US005987256A

United States Patent [19]

Wu et al.

[11] **Patent Number: 5,987,256**

[45] **Date of Patent: Nov. 16, 1999**

[54] **SYSTEM AND PROCESS FOR OBJECT RENDERING ON THIN CLIENT PLATFORMS**

[75] Inventors: **Bo Wu**; **Ling Lu**, both of San Jose, Calif.

[73] Assignee: **Enreach Technology, Inc.**, San Jose, Calif.

[21] Appl. No.: **08/922,898**

[22] Filed: **Sep. 3, 1997**

[51] **Int. Cl.**[6] **G06F 9/45**

[52] **U.S. Cl.** **395/707**

[58] **Field of Search** 395/707

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,504,917	4/1996	Austin et al.	345/522
5,513,127	4/1996	Gard et al.	395/200.53
5,551,015	8/1996	Goettelmann et al.	395/707
5,586,020	12/1996	Isozaki	395/707
5,826,089	10/1998	Ireton	395/707

OTHER PUBLICATIONS

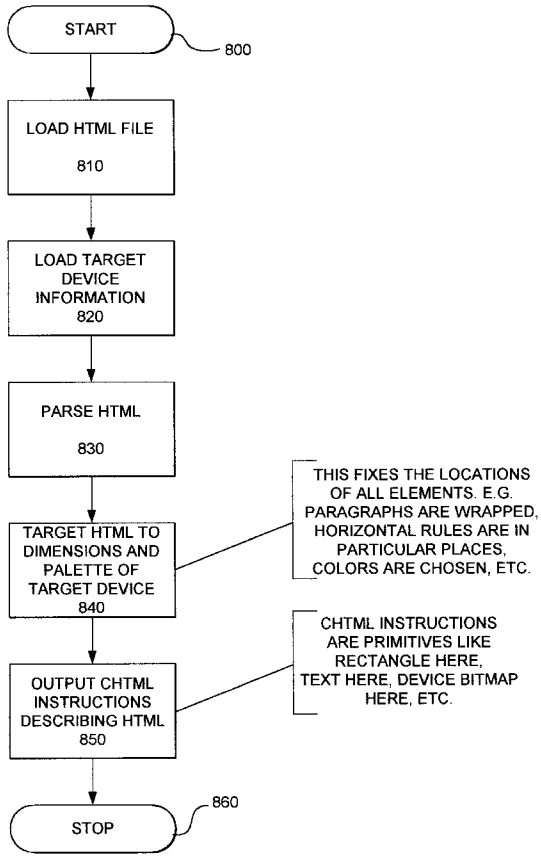
Blickstein et al. The GEM optimizing compiler system; Digital Technical Journal vol. 4 No. 4 Special Issue 1992 pp. 121–136.

Primary Examiner—Robert W. Downs
Assistant Examiner—Wei Zhen
Attorney, Agent, or Firm—Wilson Sonsini Goodrich & Rosati

[57] **ABSTRACT**

A system for processing an object specified by an object specifying language such as HTML, JAVA or other languages relying on relative positioning, that require a rendering program utilizing a minimum set of resources, translates the code for use in a target device that has limited processing resources unsuited for storage and execution of the HTML rendering program, JAVA virtual machine, or other rendering engine for the standard. Data concerning such an object is generated by a process that includes first receiving a data set specifying the object according to the object specifying language, translating the first data set into a second data set in an intermediate object language adapted for a second rendering program suitable for rendering by the target device that utilizes actual target display coordinates. The second data set is stored in a machine readable storage device, for later retrieval and execution by the thin client platform.

14 Claims, 11 Drawing Sheets



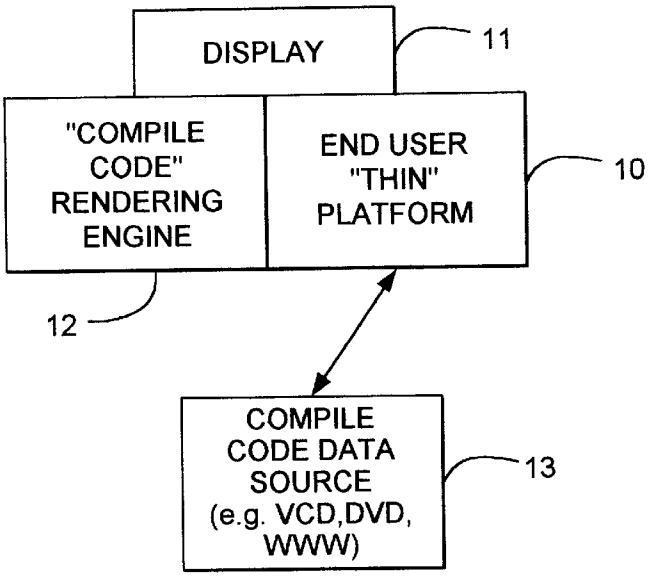

DISPLAY
11
"COMPILE CODE" RENDERING ENGINE
END USER "THIN" PLATFORM
10
12
COMPILE CODE DATA SOURCE (e.g. VCD,DVD, WWW)
13

FIG. 1

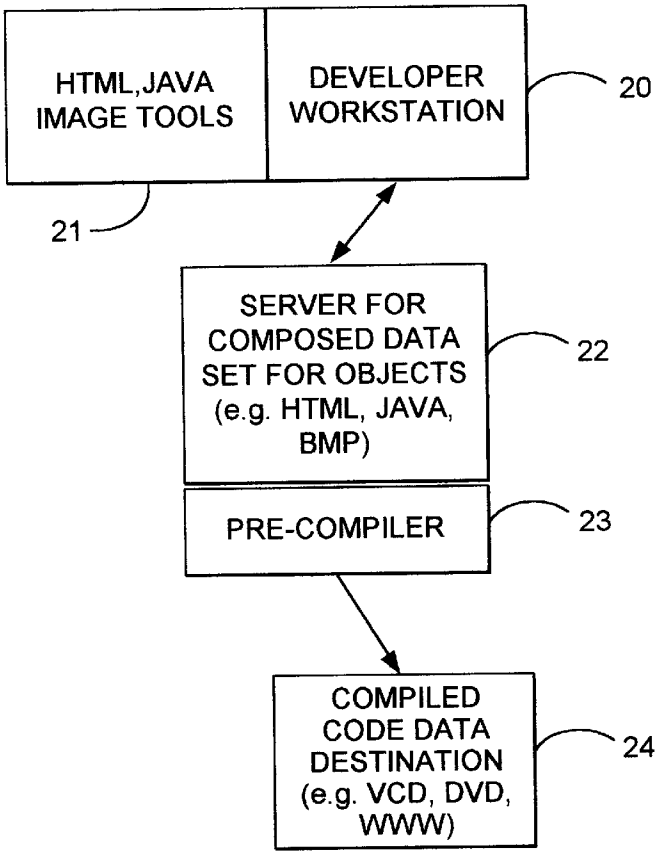

HTML,JAVA IMAGE TOOLS
DEVELOPER WORKSTATION
20
21
SERVER FOR COMPOSED DATA SET FOR OBJECTS (e.g. HTML, JAVA, BMP)
22
PRE-COMPILER
23
COMPILED CODE DATA DESTINATION (e.g. VCD, DVD, WWW)
24

FIG.2

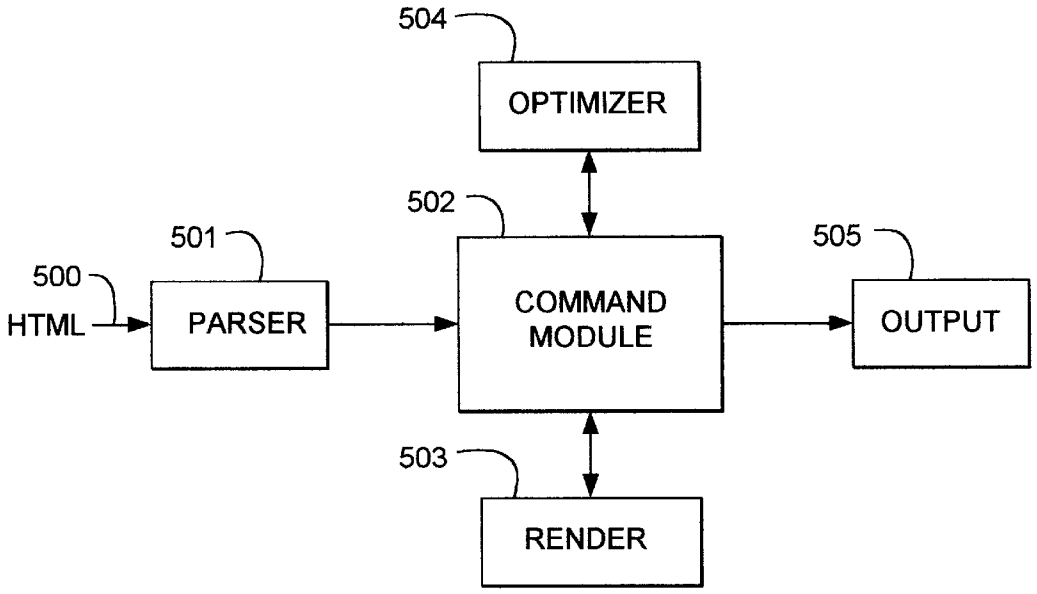

504
OPTIMIZER
502
501
505
500
HTML
PARSER
COMMAND MODULE
OUTPUT
503
RENDER

FIG. 3

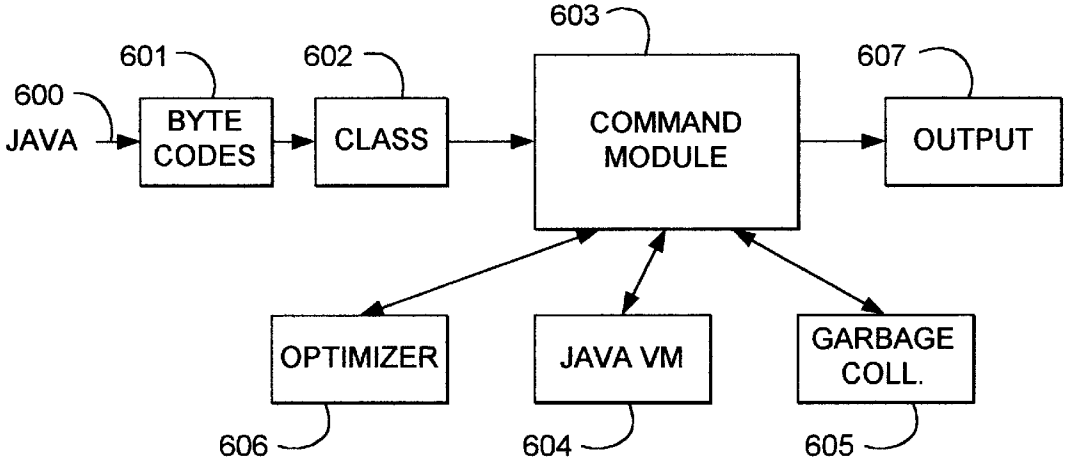

603
601
602
607
600
JAVA
BYTE CODES
CLASS
COMMAND MODULE
OUTPUT
OPTIMIZER
JAVA VM
GARBAGE COLL.
606
604
605

FIG.4

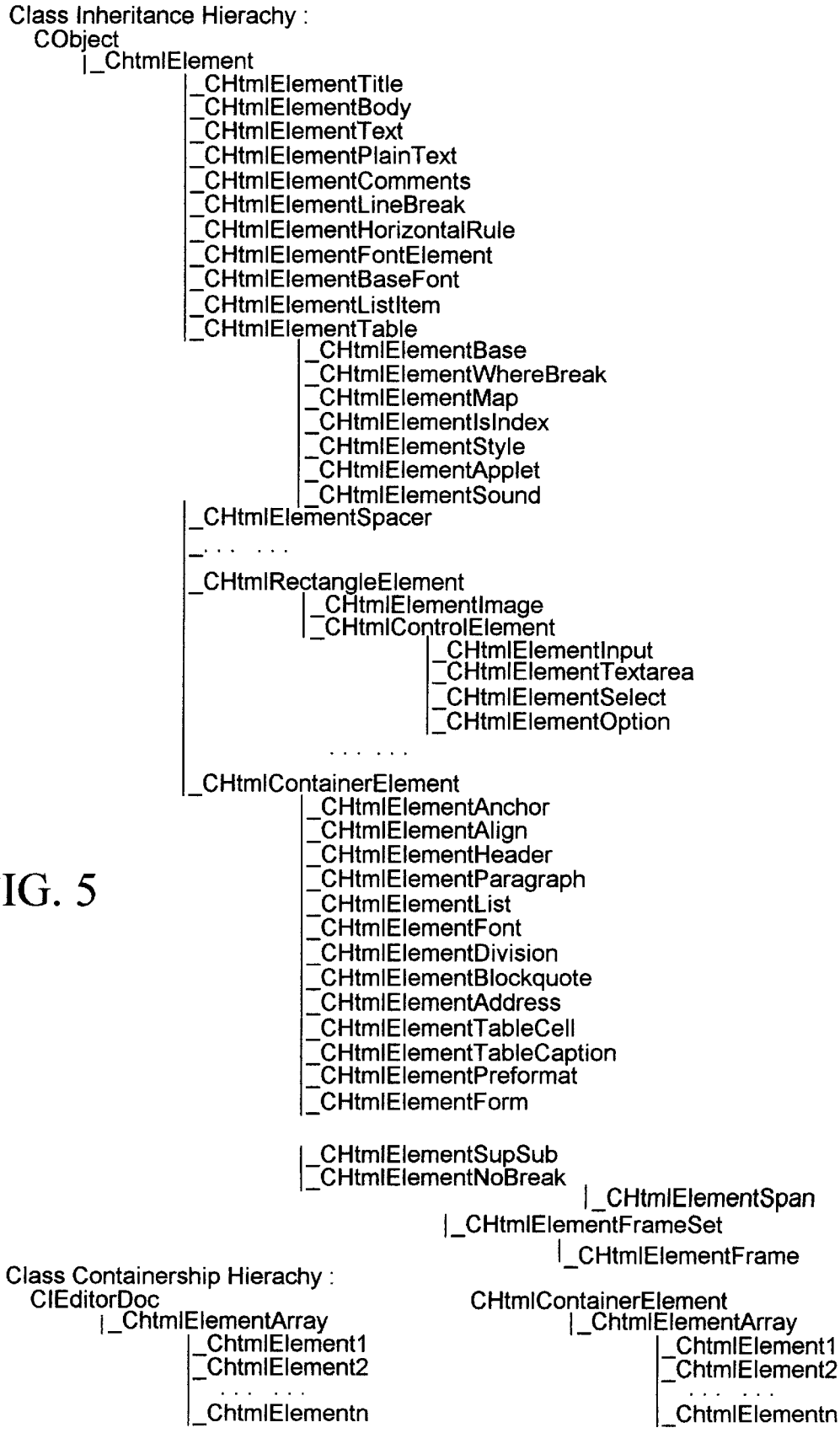

Class Inheritance Hierachy :
CObject
|_ChtmlElement
|_CHtmlElementTitle
|_CHtmlElementBody
|_CHtmlElementText
|_CHtmlElementPlainText
|_CHtmlElementComments
|_CHtmlElementLineBreak
|_CHtmlElementHorizontalRule
|_CHtmlElementFontElement
|_CHtmlElementBaseFont
|_CHtmlElementListItem
|_CHtmlElementTable
|_CHtmlElementBase
|_CHtmlElementWhereBreak
|_CHtmlElementMap
|_CHtmlElementIsIndex
|_CHtmlElementStyle
|_CHtmlElementApplet
|_CHtmlElementSound
|_CHtmlElementSpacer
|_. . . . . .
|_CHtmlRectangleElement
|_CHtmlElementImage
|_CHtmlControlElement
|_CHtmlElementInput
|_CHtmlElementTextarea
|_CHtmlElementSelect
|_CHtmlElementOption
. . . . . .
|_CHtmlContainerElement
|_CHtmlElementAnchor
|_CHtmlElementAlign
|_CHtmlElementHeader
|_CHtmlElementParagraph
|_CHtmlElementList
|_CHtmlElementFont
|_CHtmlElementDivision
|_CHtmlElementBlockquote
|_CHtmlElementAddress
|_CHtmlElementTableCell
|_CHtmlElementTableCaption
|_CHtmlElementPreformat
|_CHtmlElementForm
|_CHtmlElementSupSub
|_CHtmlElementNoBreak
|_CHtmlElementSpan
|_CHtmlElementFrameSet
|_CHtmlElementFrame
Class Containership Hierachy :
CIEditorDoc
|_ChtmlElementArray
|_ChtmlElement1
|_ChtmlElement2
. . . . . .
|_ChtmlElementn
CHtmlContainerElement
|_ChtmlElementArray
|_ChtmlElement1
|_ChtmlElement2
. . . . . .
|_ChtmlElementn

FIG. 5

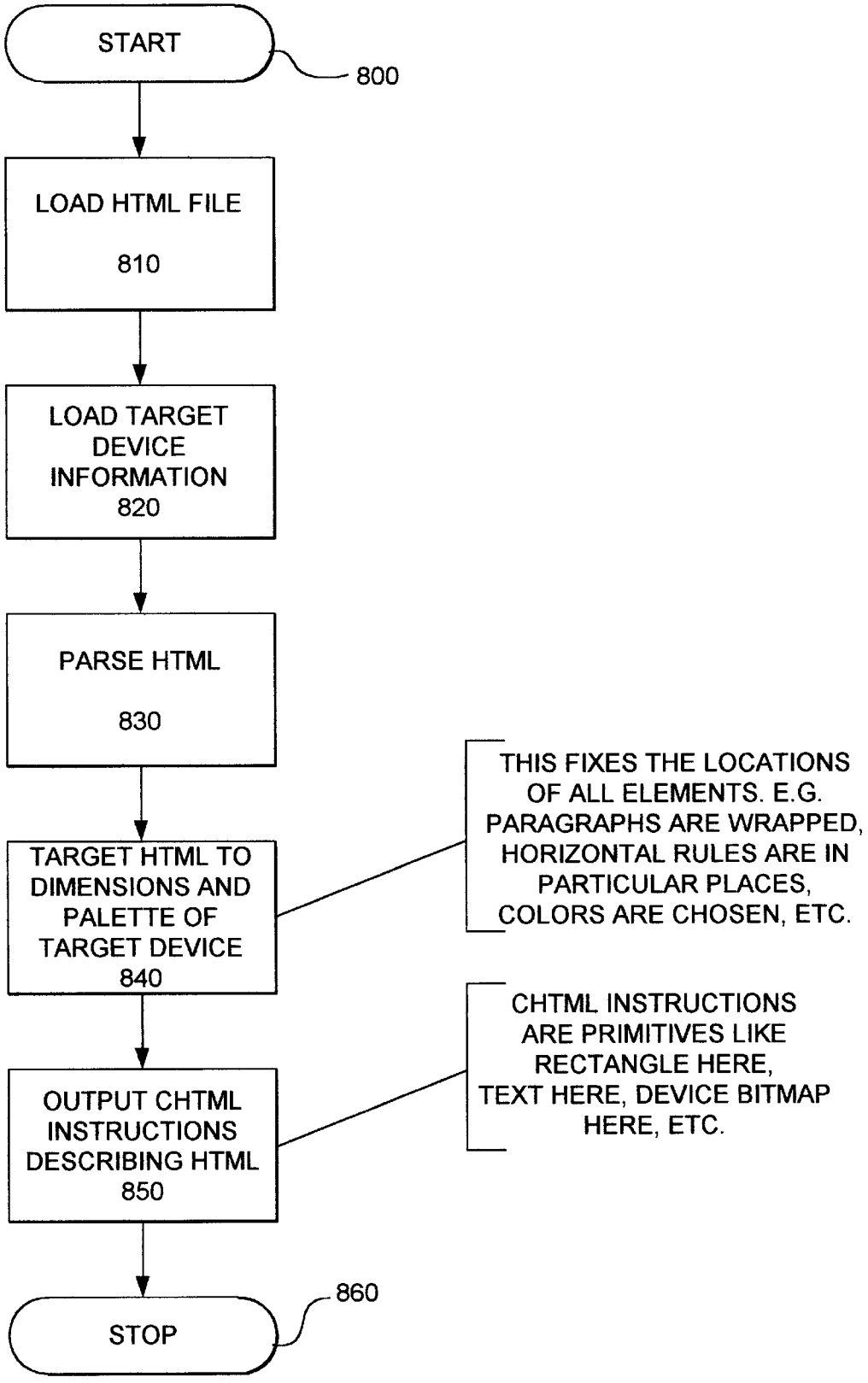

START
800
LOAD HTML FILE
810
LOAD TARGET DEVICE INFORMATION
820
PARSE HTML
830
TARGET HTML TO DIMENSIONS AND PALETTE OF TARGET DEVICE
840
THIS FIXES THE LOCATIONS OF ALL ELEMENTS. E.G. PARAGRAPHS ARE WRAPPED, HORIZONTAL RULES ARE IN PARTICULAR PLACES, COLORS ARE CHOSEN, ETC.
OUTPUT CHTML INSTRUCTIONS DESCRIBING HTML
850
CHTML INSTRUCTIONS ARE PRIMITIVES LIKE RECTANGLE HERE, TEXT HERE, DEVICE BITMAP HERE, ETC.
860
STOP

FIG. 6

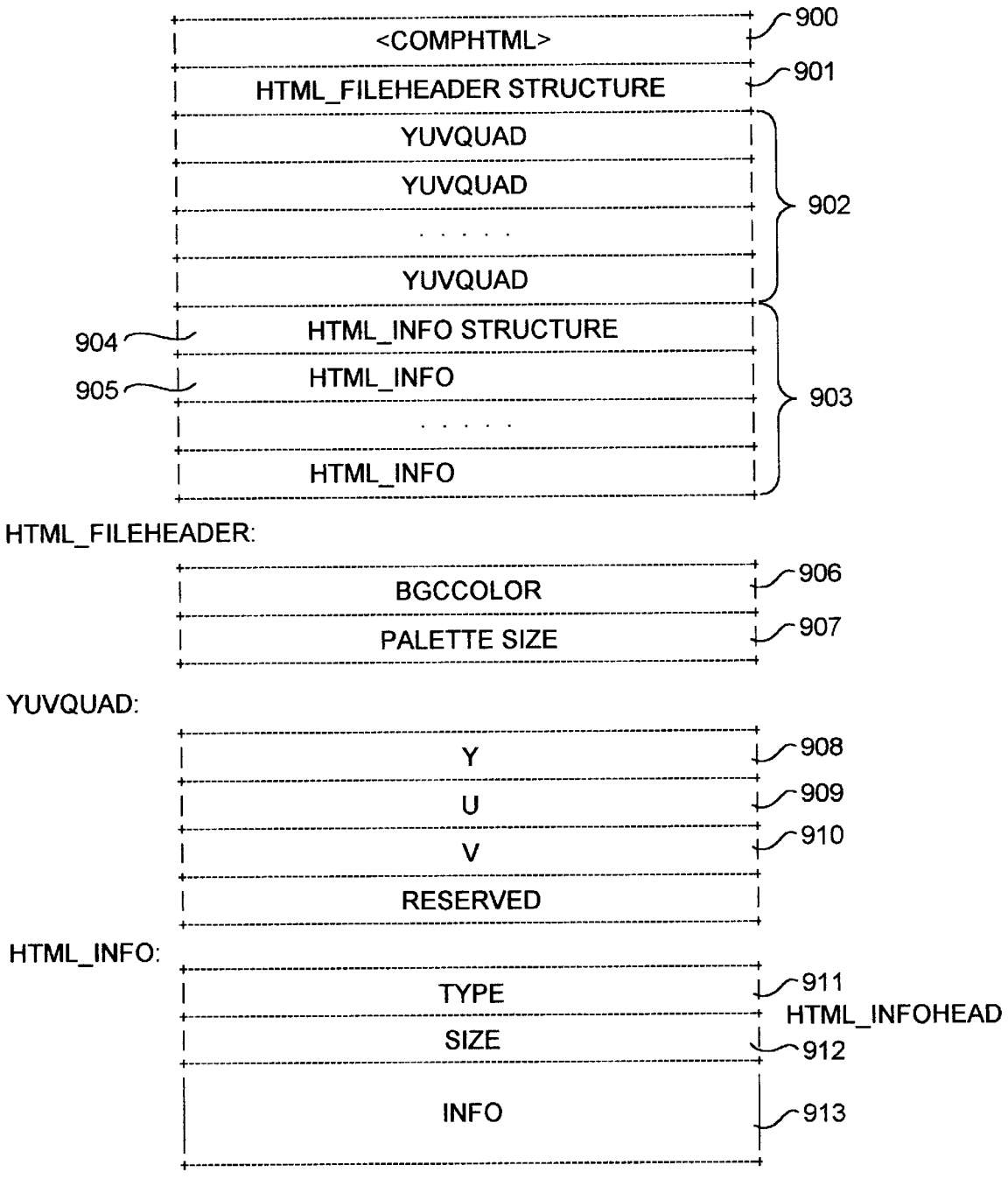

<COMPHTML>
900
HTML_FILEHEADER STRUCTURE
901
YUVQUAD
YUVQUAD
. . . . .
YUVQUAD
902
HTML_INFO STRUCTURE
904
HTML_INFO
905
. . . . .
HTML_INFO
903
HTML_FILEHEADER:
BGCCOLOR
906
PALETTE SIZE
907
YUVQUAD:
Y
908
U
909
V
910
RESERVED
HTML_INFO:
TYPE
911
HTML_INFOHEAD
SIZE
912
INFO
913

FIG. 7

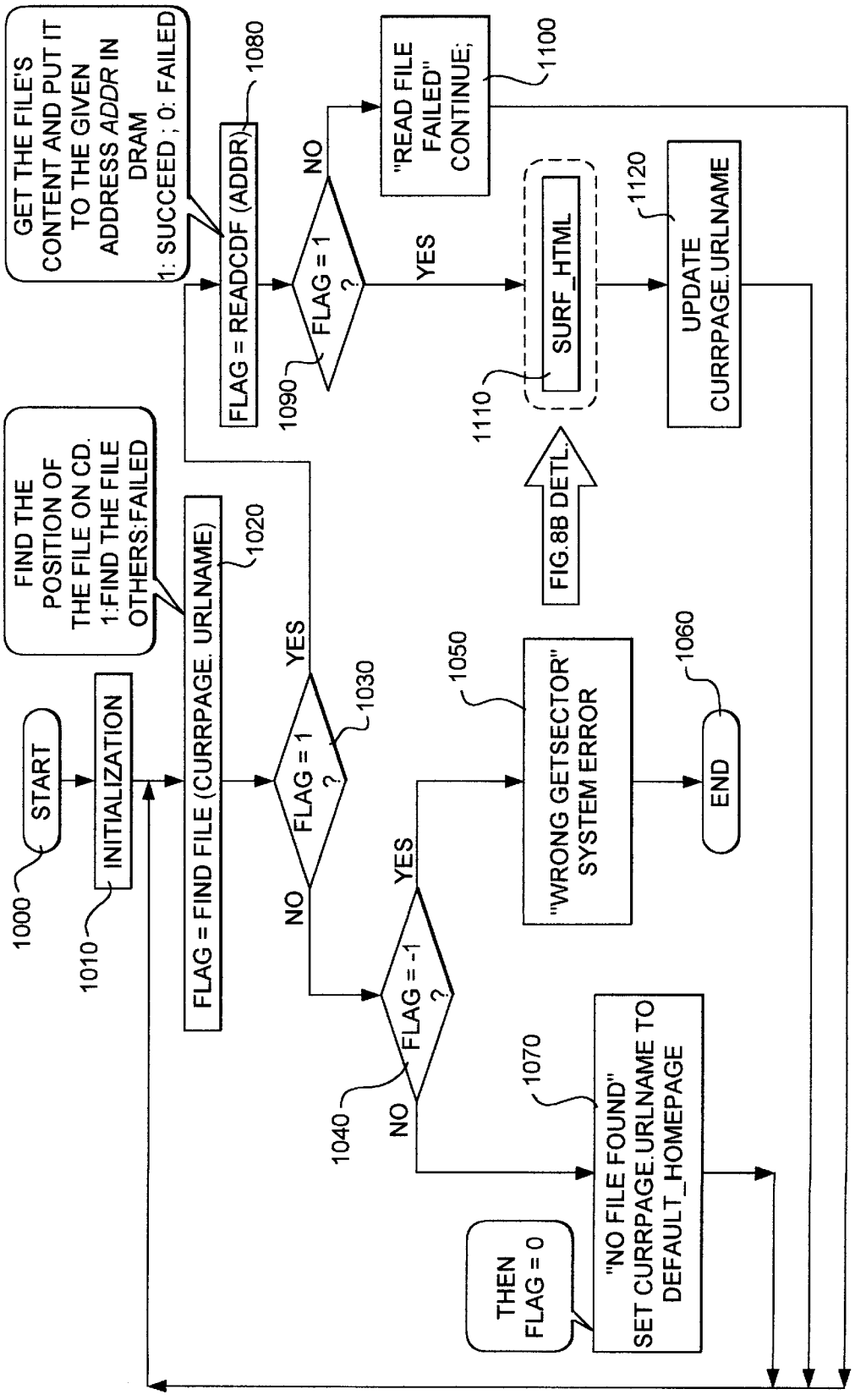

START
1000
INITIALIZATION
1010
FIND THE POSITION OF THE FILE ON CD. 1:FIND THE FILE OTHERS:FAILED
FLAG = FIND FILE (CURRPAGE. URLNAME)
1020
FLAG = 1 ?
1030
YES
NO
FLAG = -1 ?
1040
YES
NO
"WRONG GETSECTOR" SYSTEM ERROR
1050
END
1060
THEN FLAG = 0
"NO FILE FOUND" SET CURRPAGE.URLNAME TO DEFAULT_HOMEPAGE
1070
GET THE FILE'S CONTENT AND PUT IT TO THE GIVEN ADDRESS ADDR IN DRAM 1: SUCCEED ; 0: FAILED
FLAG = READCDF (ADDR)
1080
FLAG = 1 ?
1090
YES
NO
"READ FILE FAILED" CONTINUE;
1100
SURF_HTML
1110
FIG.8B DETL.
UPDATE CURRPAGE.URLNAME
1120

FIG. 8A

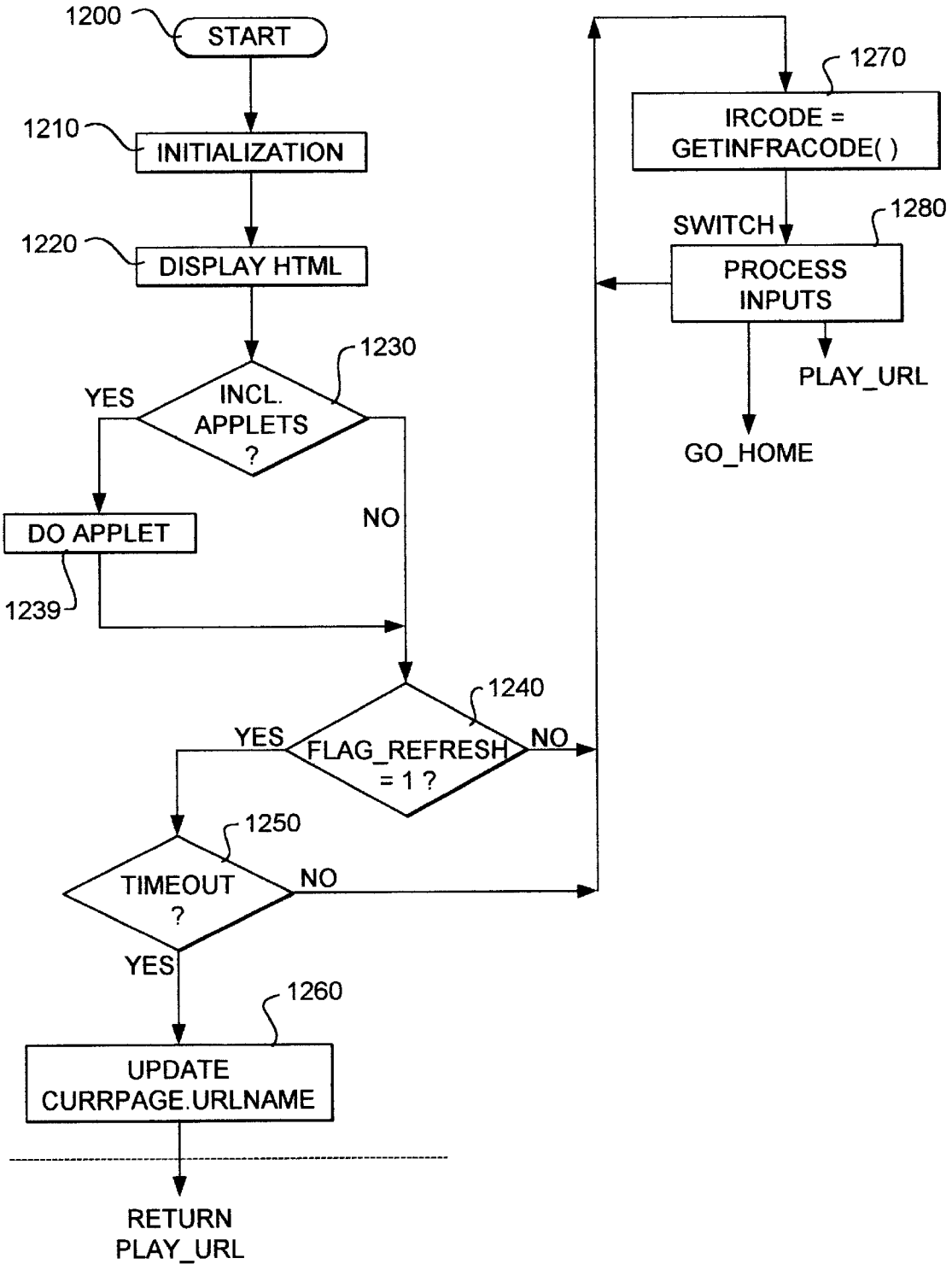

1200
START
1210
INITIALIZATION
1220
DISPLAY HTML
1230
INCL. APPLETS ?
YES
NO
DO APPLET
1239
1240
FLAG_REFRESH = 1 ?
YES
NO
1250
TIMEOUT ?
NO
YES
1260
UPDATE CURRPAGE.URLNAME
RETURN PLAY_URL
1270
IRCODE = GETINFRACODE( )
SWITCH
1280
PROCESS INPUTS
PLAY_URL
GO_HOME

FIG.8B

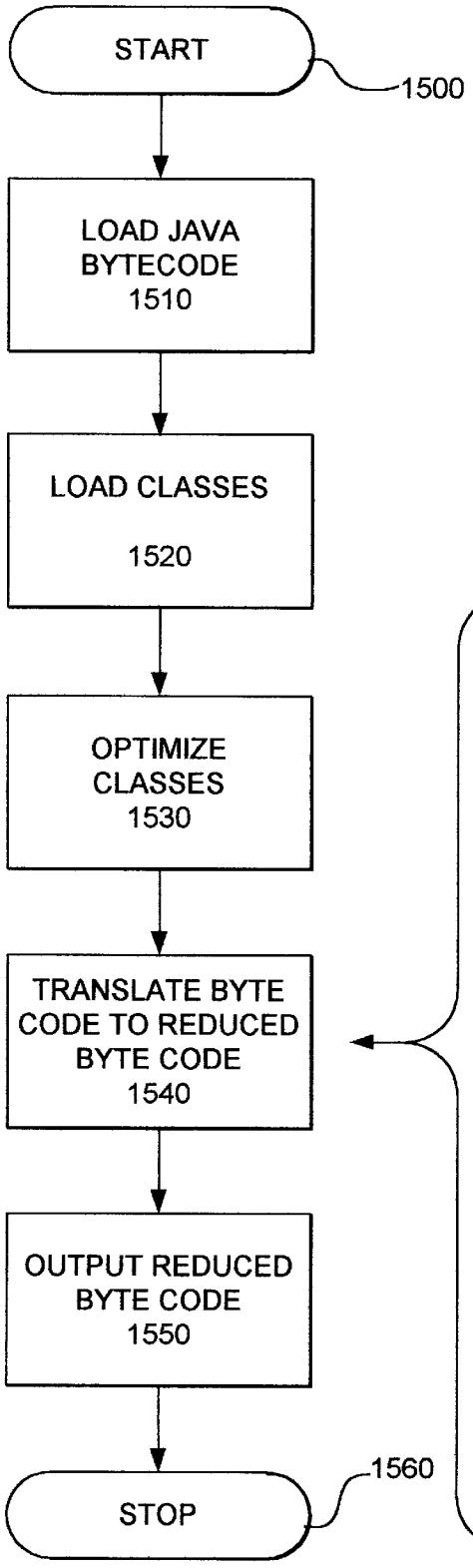

START
1500
LOAD JAVA BYTECODE 1510
LOAD CLASSES 1520
OPTIMIZE CLASSES 1530
TRANSLATE BYTE CODE TO REDUCED BYTE CODE 1540
OUTPUT REDUCED BYTE CODE 1550
STOP
1560

FIG. 9

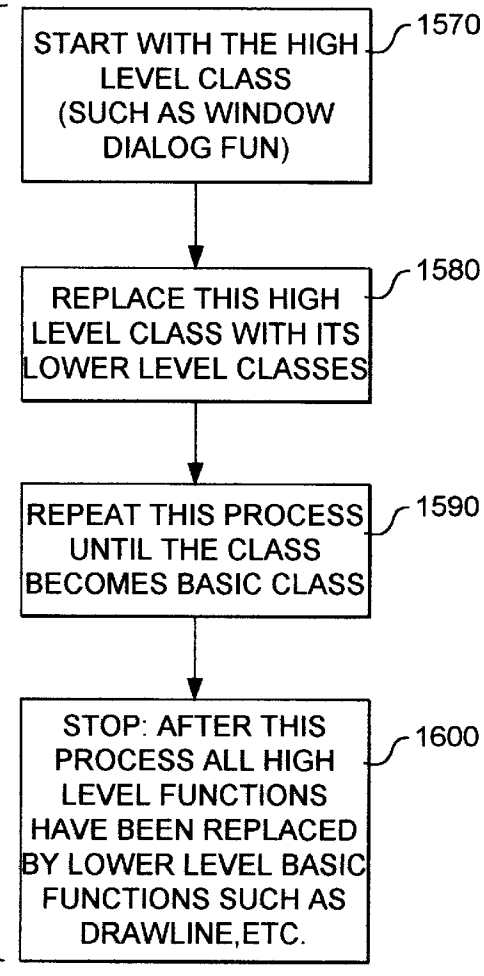

START WITH THE HIGH LEVEL CLASS (SUCH AS WINDOW DIALOG FUN)
1570
REPLACE THIS HIGH LEVEL CLASS WITH ITS LOWER LEVEL CLASSES
1580
REPEAT THIS PROCESS UNTIL THE CLASS BECOMES BASIC CLASS
1590
STOP: AFTER THIS PROCESS ALL HIGH LEVEL FUNCTIONS HAVE BEEN REPLACED BY LOWER LEVEL BASIC FUNCTIONS SUCH AS DRAWLINE,ETC.
1600

FIG. 9A

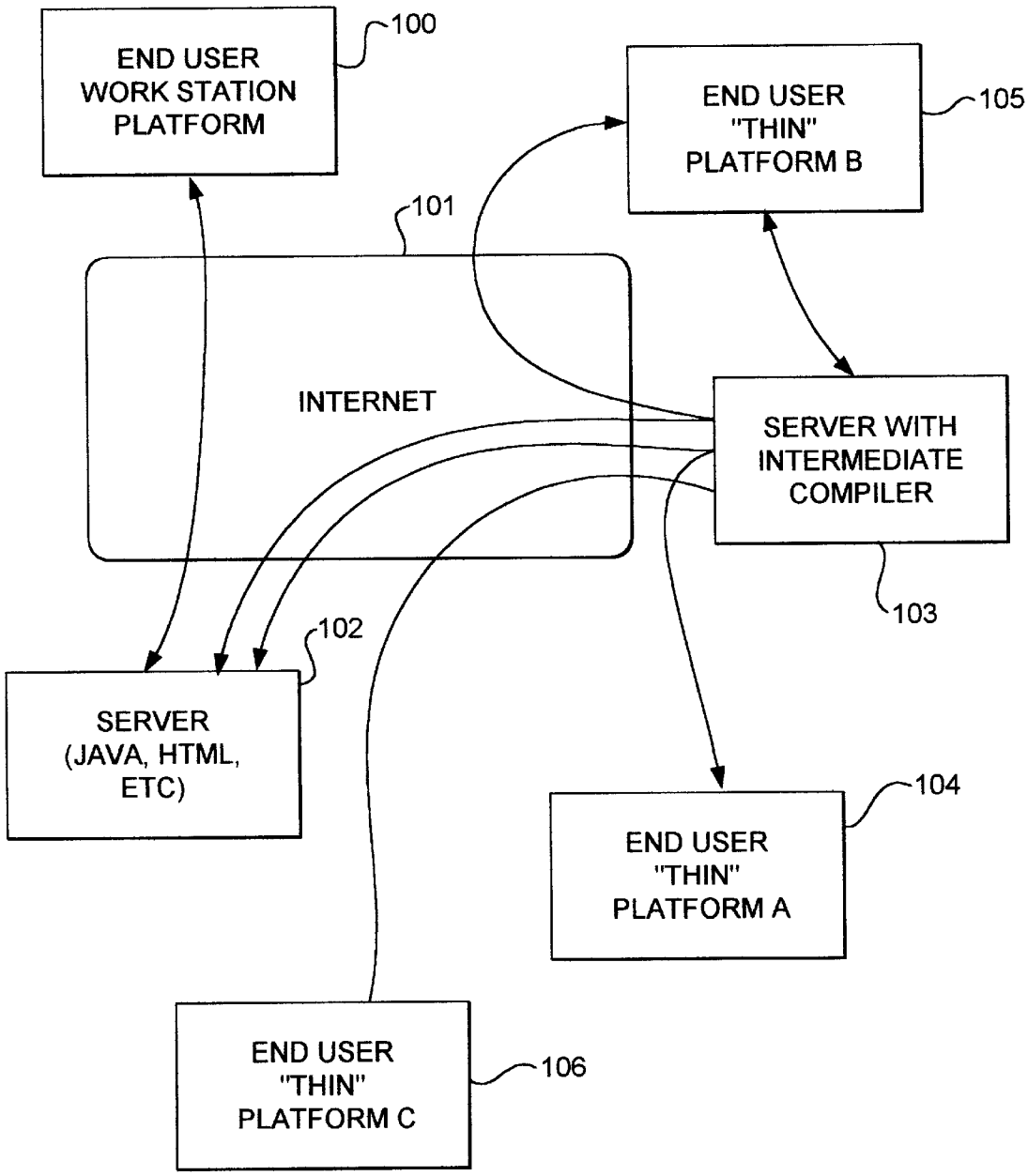

END USER
WORK STATION
PLATFORM
100
END USER
"THIN"
PLATFORM B
105
101
INTERNET
SERVER WITH
INTERMEDIATE
COMPILER
103
102
SERVER
(JAVA, HTML,
ETC)
104
END USER
"THIN"
PLATFORM A
END USER
"THIN"
PLATFORM C
106

FIG. 10

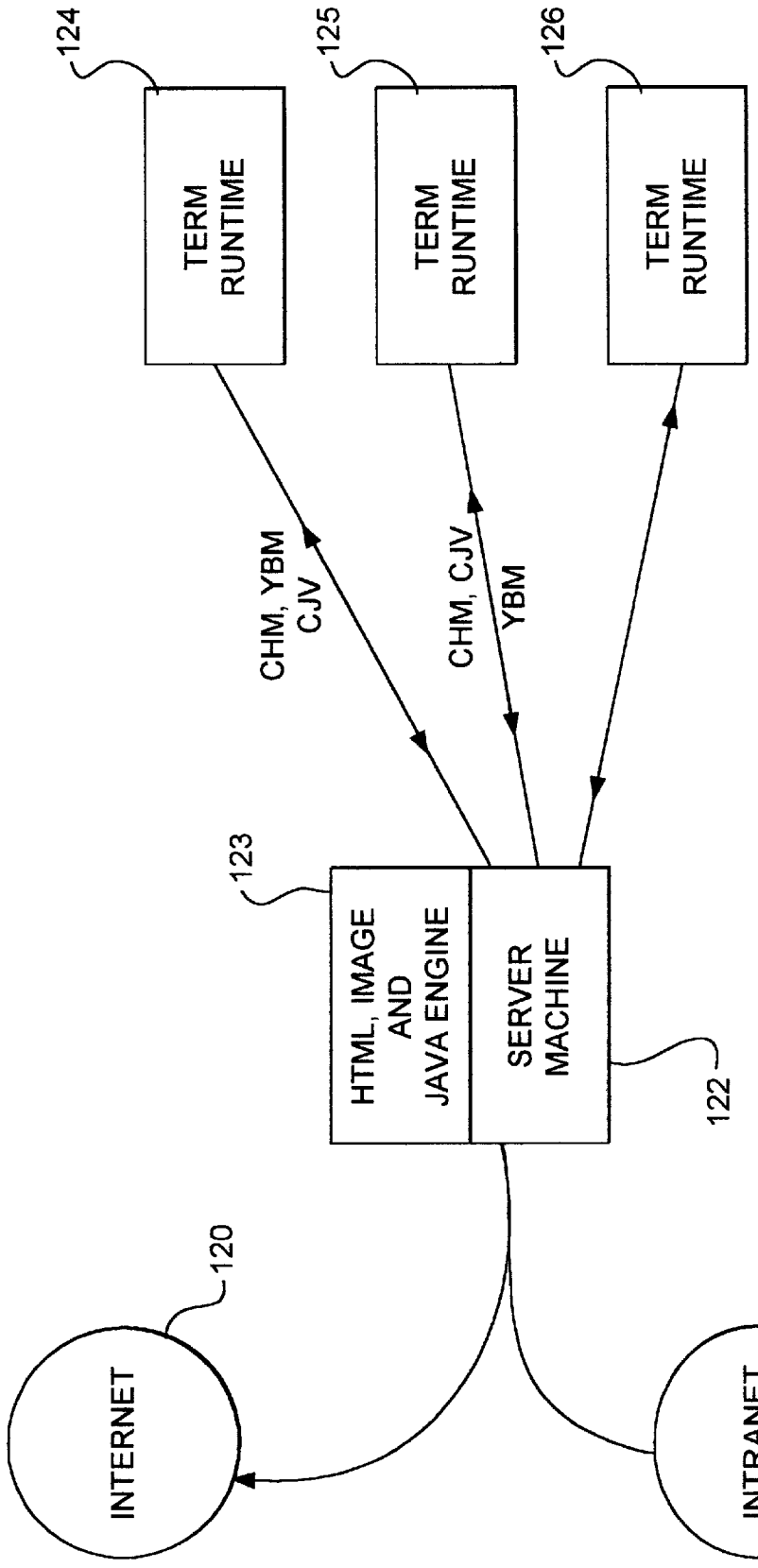

124
125
126
TERM RUNTIME
TERM RUNTIME
TERM RUNTIME
CHM, YBM CJV
CHM, CJV YBM
123
HTML, IMAGE AND JAVA ENGINE
SERVER MACHINE
122
120
121
INTERNET
INTRANET

FIG. 11

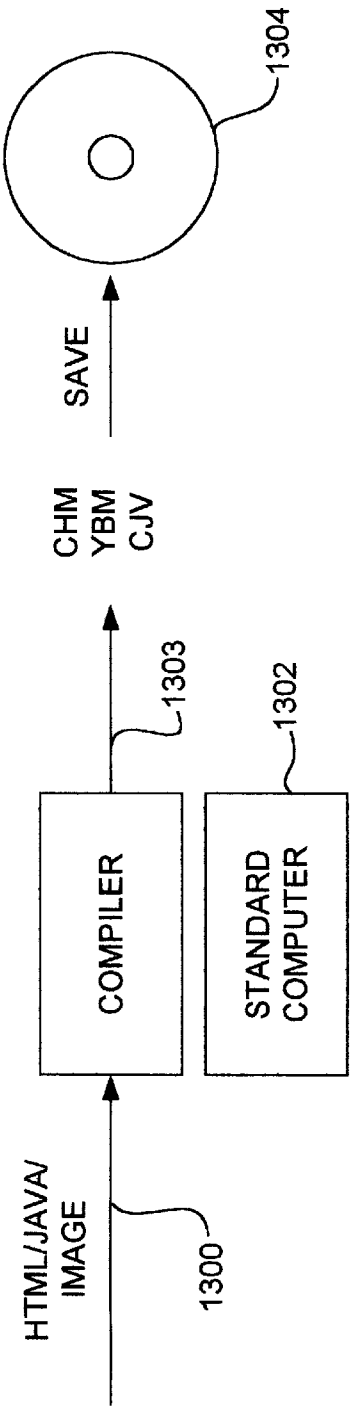


FIG. 12A

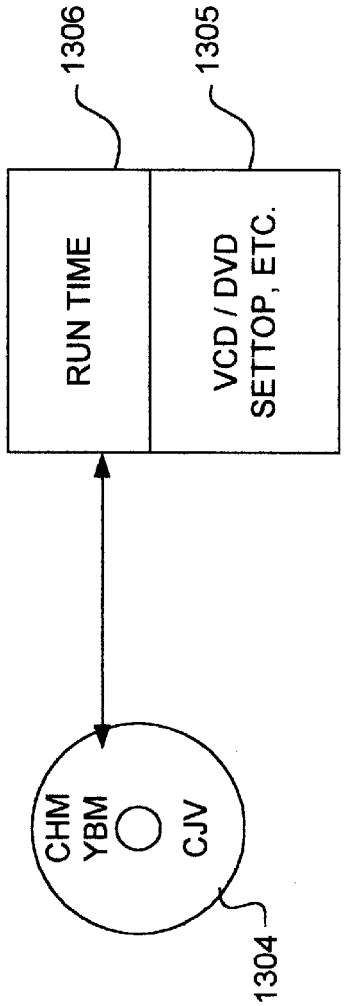


FIG. 12B

SYSTEM AND PROCESS FOR OBJECT RENDERING ON THIN CLIENT PLATFORMS

COPYRIGHT DISCLAIMER

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to a method of providing full feature program processing according to a variety of standard language codes such as HTML, JAVA and other standard languages, for execution on a thin client platform. More particularly the invention relates to methods for compiling and rendering full feature standard HTML and JAVA programs into a format which is efficient for a limited processing resource platforms.

2. Description of Related Art

Standard HTML and JAVA programs, and other hypertext languages, are designed for computers having a significant amount of data processing resources, such as CPU speed and memory bandwidth, to run well. One feature of these object specifying languages is the ability to specify a graphic object for display using relative positioning. Relative positioning enables the display of the graphic object on displays having a wide range of dimensions, resolutions, and other display characteristics. However, relative positioning of graphic objects requires that the target device have computational resources to place the graphic object on the display at specific coordinates. Thus, there are a number of environments, such as TV set top boxes, hand held devices, digital video disk DVD players, compact video disk VCD players or thin network computer environments in which these standard object specifying languages are inefficient or impractical. The original HTML and JAVA programs run very slowly, or not at all, in these types of thin client environments. To solve these problems, simpler versions of HTML and JAVA have been proposed, which have resulted in scripting out some of the features. This trades off some of the nice functionality of HTML and JAVA, which have contributed to their wide acceptance. Furthermore, use in thin client environments of the huge number of files that are already specified according to these standards, is substantially limited.

SUMMARY OF THE INVENTION

The present invention provides a system and method for processing an Display object specified by an object specifying language such as HTML, JAVA or other languages relying on relative positioning, that require a rendering program utilizing a minimum set of resources, for use in a target device that has limited processing resources unsuited for storage and execution of the HTML rendering program, JAVA virtual machine, or other rendering engine for the standard. Thus, the invention can be characterized as a method for storing data concerning such an object that includes first receiving a data set specifying the object according to the object specifying language, translating the first data set into a second data set in an intermediate object language adapted for a second rendering program suitable for rendering by the target device that utilizes actual target display coordinates. The second data set is stored in a machine readable storage device, for later retrieval and execution by the thin client platform.

The object specifying language according to alternative embodiments comprises a HTML standard language or other hypertext mark up language, a JAVA standard language or other object oriented language that includes object specifying tools.

The invention also can be characterized as a method for sending data concerning such an object to a target device having limited processing resources. This method includes receiving the first data set specifying the object according to the first object specifying language, translating the first data set to a second data set in an intermediate object language, and then sending the second data set to the target device. The target device then renders the object by a rendering engine adapted for the intermediate object language. The step of sending the second data set includes sending the second data set across a packet switched network such as the Internet or the World Wide Web to the target device. Also, the step of translating according to one aspect of the invention includes sending the first data set across a packet switched network to a translation device, and executing a translation process on the translation device to generate the second data set. The second data set is then transferred from the translation device, to the target device, or alternatively from the translation device back to the source of the data, from which it is then forwarded to the target device.

According to other aspects of the invention, the step of translating the first data set includes first identifying the object specifying language of the first data set from among a set of object specifying languages, such as HTML and JAVA. Then, a translation process is selected according to the identified object specifying language.

According to yet another aspect of the invention, before the step of translating the steps of identifying the target device from among a set of target devices, and selecting a translation process according to the identified target device, are executed.

In yet another alternative of the present invention, a method for providing data to a target device is provided. This method includes requesting for the target device a first data set from a source of data, the first data set specifying the object according to the object specifying language; translating the first data set to a second data set in an intermediate language adapted for execution according to a second rendering program by the target device. The second data set is then sent to this target device. This allows a thin platform target device to request objects specified by full function HTML, JAVA and other object specifying languages, and have them automatically translated to a format suitable for rendering in the thin environment.

Thus, the present invention provides a method which uses a computer to automatically compile standard HTML, JAVA and other programs so that such programs can run both CPU and memory efficiently on a thin client platform such as a TV set top box, a VCD/DVD player, a hand held device, a network computer or an embedded computer. The automatic compilation maintains all the benefits of full feature HTML and JAVA or other language.

The significance of the invention is evident when it is considered that in the prior art, standard HTML and JAVA were reduced in features or special standards are created for the thin client environment. Thus according to the prior art

approaches, the standard programs and image files on the Internet need to be specially modified to meet the needs of special thin client devices. This is almost impossible considering the amount of HTML and JAVA formatted files on the Web. According to the invention each HTML file, compiled JAVA class file or other object specifying language data set is processed by a standard full feature HTML browser JAVA virtual machine, or other complementary rendering engine, optimized for a target platform on the fly, and then output into a set of display oriented language codes which can be easily executed and displayed on a thin client platform. Furthermore, the technique can use in general to speed up the HTML and JAVA computing in standard platforms.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** is a simplified diagram of a end user thin platform for execution of a compiled code data source according to the present invention.

FIG. **2** is a simplified diagram of a user workstation and server for precompiling a composed data set according to the present invention.

FIG. **3** is a simplified diagram of a precompiler for a HTML formatted file.

FIG. **4** is a simplified diagram of a precompiler for a JAVA coded program.

FIG. **5** is a class inheritance hierarchy for a precompiler for HTML.

FIG. **6** is a flow chart for the HTML precompiler process.

FIG. **7** illustrates the compiled HTML structure according to one embodiment of the present invention.

FIGS. **8A–8B** illustrate a compiled HTML run time engine for execution on the thin platform according to the present invention.

FIG. **9** is a flow chart of the process for precompiling a JAVA program according to the present invention.

FIG. **9A** is a flow chart of one example process for translating the byte codes into a reduced byte code in the sequence of FIG. **9**.

FIG. **10** is a schematic diagram illustrating use of the present invention in the Internet environment.

FIG. **11** is a schematic diagram illustrating use of the present invention in a "network computer" environment.

FIG. **12A** is a schematic diagram illustrating use of the present invention in an off-line environment for producing a compiled format of the present invention and saving it to a storage medium.

FIG. **12B** illustrates the off-line environment in which the stored data is executed by thin platform.

DETAILED DESCRIPTION

A detailed description of preferred embodiments of the present invention is provided with respect to FIGS. **1–12A** and **12B**. FIGS. **1–2** illustrated simplified implementation of the present invention. FIGS. **3–9** and **9A** illustrate processes executed according to the present invention. FIGS. **10–12A** and **12B** illustrate the use of the present invention in the Internet environment or other packet switched network environment.

FIG. **1** illustrates a "thin" platform which includes a limited set of data processing resources represented by box **10**, a display **11**, and a "compiled code" rendering engine **12** for a display oriented language which relies on the data processing resources **10**. The end user platform **10** is coupled to a compiled code data source **13**. A compiled code data sources comprises, for example a VCD, a DVD, or other computer readable data storage device. Alternatively, the compiled code data source **13** consists of a connection to the World Wide Web or other packet switched or point-to-point network environment from which compiled code data is retrieved.

The limited data processing resources of the thin platform **10** include for example a microcontroller and limited memory. For example, 512k of RAM associated with a 8051 microcontroller, or a 66 MHz MIPS RISC CPU and 512k of dynamic RAM may be used in a representative thin platform. Other thin user platforms use low cost microprocessors with limited memory. In addition, other thin platforms may comprise high performance processors which have little resources available for use in rendering the compiled code data source. Coupled with the thin platform is a compiled code rendering engine **12**. This rendering engine **12** is a relatively compact program which runs efficiently on the thin platform data processing resources. The rendering engine translates the compiled code data source data set into a stream of data suitable for the display **11**. In this environment, the present invention is utilized by having the standard HTML or JAVA code preprocessed and compiled into a compiled HTML/JAVA format according to the present invention using the compiler engine described in more detail below on a more powerful computer. The compiled HTML/JAVA codes are saved on the storage media. A small compiled HTML/JAVA run time engine **12** is embedded or loaded into the thin client device. The run time engine **12** is used to play the compiled HTML/JAVA files on the thin platform **10**. This enables the use of a very small client to run full feature HTML or JAVA programs. The machine can be used both online, offline or in a hybrid mode.

FIG. **2** illustrates the environment in which the compiled code data is generated according to the present invention. Thus for example, a developer workstation **20** is coupled with image rendering tools such as HTML, JAVA, or other image tools **21**. The workstation **20** is coupled to a server for the composed data **22**. The server includes a precompiler **23** which takes the composed data and translates it into the compiled code data. Compiled code data is then sent to a destination **24** where it is stored or rendered as suits the needs of a particular environment. Thus for example, the destination may be a VCD, DVD or the World Wide Web.

According to the environment of FIG. **2** compiled HTML and JAVA "middleware" is implemented on an Internet server. Thus the thin set top box or other compiled code data destination **24** is coupled to the Internet/Intranet through the compiled HTML/JAVA middleware **22**, **23**. A small compiled HTML/JAVA run time engine is embedded in the thin destination device. All the HTML/JAVA files created in the workstation **20** go through the middleware server **22** to reach the thin client devices. The HTML/JAVA files are converted to the compiled format on the fly by the precompiler **23** on the middleware server **22**. The server **22** passes the compiled code onto the destination device. This allows for most software updates of precompiler techniques to be made in the server environment without the need to update the destination devices. Also, any changes in the run time engine that need to be executed in the destination device **24** can be provided through the link to the server **22**.

FIGS. **3** and **4** illustrate simplified diagrams of the precompilers for HTML and JAVA respectively. In FIG. **3**,

standard HTML files are received at input **500** and applied to a HTML parser **501**. The output of the parser is applied to a command module **502** which includes a HTML rendering engine **503**, and memory resident HTML objects optimizing engine **504**. The output consists of the compiled HTML output engine **505** generates the output with simplified graphics primatives.

The basic class inheritance hierarchy for the HTML precompiling is shown in FIG. **5**. The process of translating a HTML file to the compiled HTML structure of the present invention is illustrated in FIG. **6**. The process begins at point **800** in FIG. **6**. The first step involves loading the HTML file into the rendering device. Next information concerning the target device is loaded (step **820**). The HTML file is then parsed by searching for HTML tags, and based on such tags creating the class structure of FIG. **5** (step **830**).

Using the parameters of the target device, and the parsing class structure set up after the parsing process, the algorithm does HTML rendering based on a class hierarchy adapted to the dimensions and palette of the target device (step **840**). This fixes the coordinates of all the graphic objects specified by the HTML code on the screen of the target device. For example, the paragraphs are word wrapped, horizontal rules are placed in particular places, the colors are chosen, and other device specific processes are executed.

After the rendering, all the display information is saved back into the class structure of FIG. **5**. Finally the process goes through the class hierarchy and outputs the rendering information in compiled HTML format (step **850**). The compiled HTML instructions are primitives that define rectangles, text, bitmaps and the like and their respective locations. After outputting the compiled instructions, the process is finished (step **860**).

A simplified pseudo code for the HTML compilation process is provided in Table 1.

TABLE 1

Copyright EnReach 1997

```
function convert_html (input : pointer) : chtmlfile;
    // this takes a pointer to an HTML file and translates it into a CHTML binary file
begin
    deviceInfo := LoadDeviceInfo( );    // Loads size and colors of target device
    Parse HTML file                     // use a parser to break the HTML file up into
                                        // tags represented in a fashion suitable for display
    For each HTML tag (<IMG . . . > = 1 tag, <P> a paragraph </P> = 1 tag),
select a sequence of CHTML instructions to render the tag on the output device.
As instructions are selected, colors and positioning are optimized based on the
device size and palette.
        CHTML instructions include:
            TITLE       string
            TEXT        formatted text at a specific position,
                        complex formatting will
                        require multiple CHTML TEXT instructions
            IMAGE       image information including image-map,
                        animation info, image data
            ANCHOR      HTML reference
        Basic geometric instructions such as: SQUARE, FILLEDSQUARE, CIRCLE,
        FILLEDCIRCLE, and LINE, permit the complex rendering required by some
        HTML instructions to be decomposed into basic drawing instructions. For
        example, the bullets in front of lists can be described in CHTML instructions
        as squares and circles at specific locations.
        CHTML instructions including TEXT and IMAGE instructions can be
        contained within anchors. The CHTML compiler must properly code all
        instructions to indicate if an instruction is contained in an anchor.
    The CHTML instructions can then be written to the output file along with some header
    information.
end;
```

Table 2 sets forth the data structure for the precompiling process.

TABLE 2

Copyright EnReach 1997

```
/* HTML font structure */
typedef struct tagHTMLFont
{
    char name[64];
    int size;
    int bold;
    int italic;
    int underline;
    int strikeout;
} HTMLFont;
/* FG point structure */
typedef struct tagFGPoint
```

TABLE 2-continued

Copyright EnReach 1997

```
{
    int fX;
    int fY;
} FGPoint;
/* FG rectangle structure */
typedef struct tagFGRect
{
    int fLeft;
    int fTop;
    int fRight;
    int fBottom;
} FGRect;
/* html node types, used by hType attribute in HTML_InfoHead structure */
#define HTML_TYPE_TITLE     0           /* title of the html page */
#define HTML_TYPE_TEXT      1           /* text node */
#define HTML_TYPE_CHINESE   2           /* chinese text node */
#define HTML_TYPE_IMAGE     3           /* image node */
#define HTML_TYPE_SQUARE    4           /* square frame */
#define HTML_TYPE_FILLEDSQUARE 5        /* filled square */
#define HTML_TYPE_CIRCLE    6           /* circle frame */
#define HTML_TYPE_FILLEDCIRCLE 7        /* filled circle */
#define HTML_TYPE_LINE      8           /* line */
#define HTML_TYPE_ANCHOR    9           /* anchor node */
#define HTML_TYPE_ANIMATION    10       /* animation node */
#define HTML_TYPE_MAPAREA   11          /* client side image map area node */
/* header info of compiled html file */
typedef struct tagHTML_FileHead
{
    unsigned int fBgColor;                  /* background color index */
    unsigned int fPaletteSize;              /* size of palette */
} HTML_FileHead;
/* header info of each html node */
typedef struct tagHTML_InfoHead
{
    unsigned int hType;                     /* type of the node */
    unsigned int hSize;                     /* size of htmlInfo */
} HTML_InfoHead;
/* html info structure */
typedef struct tagHTML_Info
{
    HTML_InfoHead htmlHead;                 /* header info */
    unsigned char htmlInfo[1];              /* info of the html node */
} HTML_Info;
/* html title structure */
typedef struct tagHTML_Title
{
    unsigned int textLen;                   /* length of text buffer */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Title;
/* html text structure */
typedef struct tagHTML_Text
{
    FGPoint dispPos;                        /* display coordinates */
    int anchorID;                           /* anchor id if it's inside an anchor, -1 if not */
    HTMLFont textFont;                      /* font of the text */
    unsigned int textColor;                 /* color index of the text */
    unsigned int textLen;                   /* length of text buffer */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Text
/* html chinese structure */
typedef struct tagHTML_Chinese
{
    FGPNT dispPos;                          /* display coordinates */
    int anchorID;                           /* anchor id if it's inside an anchor, -1 if not */
    unsigned int textColor;                 /* color index of the text */
    unsigned int bufLen;                    /* length of the bitmap buffer (16* 16) */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Chinese;
/* html image structure */
typedef struct tagHTML_Image
{
    FGRect dispPos;                         /* display coordinates */
    int anchorID;                           /* anchor id if it's inside an anchor, -1 if not */
```

TABLE 2-continued

Copyright EnReach 1997

```
    int animationID; /* animation id if it supports animation, -1 if not */
    int animationDelay;                    /* delay time for animation */
    char mapName[64];              /* name of client side image map, empty if no
image map */
    void *data;                            /* used to store image
data */
    unsigned int fnameLen;             /* length of the image file name */
    char fname[1];                         /* image filename */
} HTML_Image;
/* square structure */
typedef strnct tagHTML_Square
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int borderColor; /* border color index */
} HTML_Square;
/* filled square structure */
typedef struct tagHTML_FilledSquare
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int brushColor; /* the inside color index */
} HTML_FilledSquare;
/* circle structure */
typedef struct tagHTML_Circle
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int borderColor; /* border color index */
} HTML_Circle
/* circle structure */
typedef struct tagHTML_FilledCircle
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int brushColor; /* the inside color index */
} HTML_FilledCircle;
/* line structure */
typedef struct tagHTML_Line
{
    FGPoint startPos;                      /* line starting position */
    FGPoint endPos;                        /* line end position */
    int style;                             /* style of the line (solid, dashed, dotted,
etc.) */
    unsigned int penColor;                 /* pen color index */
} HTML_Line;
/* anchor structure */
typedef struct tagHTML_Anchor
{
    int anchorID;                          /* id of the anchor */
    unsigned int hrefLen;.             /* length of href */
    char href[1];                          /* url of the anchor */
} HTML_Anchor;
/* animation structure */
typedef struct tagHTML_Animation
{
    int animationID;                   /* id of the animation */
    unsigned int frameTotal; /* total number of animation frames */
    long runtime;                          /* animation runtime */
} HTML_Animation;
#define SHAPE_RECTANGLE 0
#define SHAPE_CIRCLE 1
#define SHAPE_POLY 2
/* image map area structure */
typedef struct tagHTML_MapArea
{
    char mapName[64];                      /* name of client side image map
*/
    intshape;                              /* shape of the area */
    int numVer;                            /* number of vertix */
    int coords[6][2];              /* coordinates */
    unsigned int hrefLen;          /* length of href */
    char href[1];                          /* url the area pointed to */
} HTML_MapArea;
```

An example routine for reading this file into the thin platform memory follows in Table 3.

TABLE 3

Copyright EnReach 1997

```
reading this file:
#define BLOCK_SIZE 256
/* returns number of nodes */
long read_chm(const char *filename,          /* input: .chm file name */
    HTML_Info ***ppNodeList, /* output: array of (HTML_Info *)
                                                    including anchors. */
    YUVQUAD **ppPalette,                                /* output: page palette */
    unsigned int *palette_size)                         /* output: palette size */
{
    int fd;
    char head[12];
    long total_nodes = 0;
    long max_nodes = 0;
    HTML_FileHead myFileHead;
    HTML_InfoHead myInfoHead;
    HTML_Info *pNodeInfo;
    void *pNodeData;
    long i;
    HTML_InfoHead *pHead;
    if (!ppNodeList || !ppPalette || !palette_size)
        return 0;
    (*ppNodeList) = NULL;
    (*ppPalette) = NULL;
    (*palette_size) = 0
    /* open file */
    fd = _open(filename,_O_BINARY|_O_RDONLY);
    if(fd < 0)
        return 0;
    /* read header and check for file type */
    if (_read(fd, head, 10) != 10)
    {
        _close(fd);
        return 0;
    }
    if (strncmp(head, "<COMPHTML>", 10))
    {
        _close(fd);
        return 0;
    }
    /* read file header */
    if(_read(fd, &myFileHead, sizeof(HTML_FileHead)) !=
sizeof(HTML_FileHead))
    {
        _close(fd);
        return 0;
    }
    (*palette_size) = myFileHead.fpaletteSize;
    /* read the palette */
    if ((*palette_size) > 0)
    {
        (*ppPalette) = (YUVQUAD *)malloc(sizeof(YUVQUAD)*
(*palette_size));
        if (_read(fd, (*ppPalette), sizeof(YUVQUAD) * (*palette_size))
            != (int) (sizeof(YUVQUAD) * (*palette_size)))
        {
            _close(fd);
            return 0;
        }
    }
    /* read anchors along with other html nodes */
    while (1)
    {
        if (_read(fd, &myInfoHead, sizeof(HTML_InfoHead))
            != sizeof(HTML_InfoHead))
        {
            break;
        }
        if (myInfoHead.hSize > 0)
        {
            pNodeInfo = (HTML_Info *) malloc(myInfoHead.hSize +
sizeof(HTML_InfoHead));
            if (!pNodeInfo)
                break;
            memcpy(pNodeInfo, &myInfoHead,
sizeof(HTML_InfoHead));
            if (_read(fd, &pNodeInfo[sizeof(HTML_InfoHead)],
myInfoHead.hSize)
```

TABLE 3-continued

Copyright EnReach 1997

```
                        != (int)myInfoHead.hSize)
                {
                        break;
                }
                /* check if we need to do memory allocation */
                if (total_nodes >= max_nodes)
                {
                        if(!max_nodes)
                        {
                                /* no node in the list yet */
                                (*ppNodeList) = (HTML_Info **)
malloc(
                                        sizeof(HTML_Info *)*
BLOCK_SIZE);
                        }
                        else
                        {
                                (*ppNodeList) = (HTML_Info **)
realloc((*ppNodeList),
                                        max_nodes + sizeof(HTML_Info
*) * BLOCK_SIZE);
                        }
                        if (!(*ppNodeList))
                                break;
                        max_nodes += BLOCK_SIZE;
                }
                (*ppNodeList)[total_nodes] = pNodeInfo;
                total_nodes++;
        }
    }
    _close(fd);
    /* test our data */
    for (i = 0; i < total_nodes; i++)
    {
        pNodeInfo = (*ppNodeList)[i];
        pHead = (HTML_InfoHead *) pNodeInfo;
        pNodeData = pNodeInfo + sizeof(HTML_InfoHead);
        if(pHead->hType == HTML_TYPE_TEXT)
        {
                HTML_Text *pText = (HTML_Text *) pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_IMAGE)
        {
                HTML_Image *pImage = (HTML_Image *) pNodeData;
                if (pImage->fnameLen > 0)
                {
                        /* load the image file */
                        pImage->data = load_ybm(pImage->fname);
                }
        }
        else if (pHead->hType == HTML_TYPE_ANCHOR)
        {
                HTML_Anchor *pAnchor = (HTML_Anchor *)
pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_ANIMATION)
        {
                HTML_Animation *pAnimation = (HTML_Animation *)
pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_MAPAREA)
        {
                HTML_MapArea *pMapArea = (HTML_MapArea *)
pNodeData;
        }
        else if (pHead->hType == HTML_TYPE_LINE)
        {
                HTML_Line *pLine = (HTML_Line *) pNodeData;
        }
        else if(1)Head->hType == HTML_TYPE_SQUARE)
        {
                HTML_Square *pSquare = (HTML_Square *) pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_CIRCLE)
        {
                HTML_Circle *pCircle = (HTML_Circle *) pNodeData;
        }
```

TABLE 3-continued

Copyright EnReach 1997

```
            else if (pHead->hType == HTML_TYPE_FILLEDSQUARE)
            {
                HTML_FilledSquare *pFilledSquare =
(HTML_FilledSquare *) pNodeData;
            }
            else if (pHead->hType == HTML_TYPE_FILLEDCIRCLE)
            {
                HTML_FilledCircle *pFilledCircle = (HTML_FilledCircle
*) pNodeData;
            }
            else if(pHead->hType == HTML_TYPE_TITLE)
            {
                HTML_Title *pTitle = (HTML_Title *) pNodeData;
            }
    }
    return total_nodes;
}
```

The compiled HTML file structure is set forth in FIG. **7** as described in Table 2. The file structure begins with a ten character string COMPHTML **900**. This string is followed by a HTML file header structure **901**. After the file header structure, a YUV color palette is set forth in the structure **902** this consists of an array of YUVQUAD values for the target device. After the palette array, a list **903** of HTML information structures follows. Usually the first HTML information structure **904** consists of a title. Next, a refresh element typically follows at point **905**. This is optional. Next in the line is a background color and background images if they are used in this image. After that, a list of display elements is provided in proper order. The anchor node for the HTML file is always in front of the nodes that it contains. An animation node is always right before the animation image frames start. The image area nodes usually appear at the head of the list.

The HTML file header structure includes a first value BgColor at point **906** followed by palette size parameters for the target device at point **907**. The YUVQUAD values in the color palette consist of a four word structure specifying the Y, U, and V values for the particular pixel at points **908–910**. The HTML information structures in the list **903** consist of a type field **911**, a size field **912**, and the information which supports the type at field **913**. The type structures can be a HTML_Title, HTML_Text, HTML_Chinese, HTML_Xxge, HTML_Square, HTML_FilledSquare, HTML_Circle, HTML_FilledCircle, HTML_Line, HTML_Author, HTML_Animation, . . .

Functions that would enable a thin platform to support viewing of HTML-based content pre-compiled according to the present invention includes the following:

General graphics functions

int DrawPoint (int x, int y, COLOR color, MODE mode);

int DrawLine (int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int DrawRectangle(int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int FillRectangle(int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int ClearScreen(COLOR color);

Color palette

int ChangeYUVColorPalette( );

Bitmap function

int BitBlt(int dst_x1, int dst_y1, int dst_x2, int dst_y2, unsigned char *bitmap, MODE mode);

String drawing functions

int GetStringWidth(char *str, int len);

int GetStringHeight(char *str, int len);

int DrawStringOnScreen(int x, int y, char *str, int len, COLOR color, MODE mode);

Explanation

All (x, y) coordinates are based on the screen resolution of the target display device (e.g. 320×240 pixels).

COLOR is specified as an index to a palette.

MODE defines how new pixels replace currently displayed pixels (COPY, XOR, OR, AND).

Minimum support for DrawLine is a horizontal or vertical straight line, although it would be nice to have support for diagonal lines.

The ChangeYUVColorPalette function is used for every page.

BitBlt uses (x1, y1) and (x2, y2) for scaling but it is not a requirement to have this scaling functionality.

String functions are used for English text output only. Bitmaps are used for Chinese characters.

FIGS. **8**A and **8**B set forth the run time engine suitable for execution on a thin client platform for display of the compiled HTML material which includes the function outlined above in the "display" step **1220** of FIG. **8**B.

The process of FIG. **8**A starts at block **1000**. The ran time engine is initialized on the client platform by loading the appropriate elements of the run time engine and other processes known in the art (step **1010**). The next step involves identifying the position of the file, such as on the source CD or other location from which the file is to be retrieved and setting a flag (step **1020**). The flag is tested at step **1030**. If the flag is not set, then the algorithm branches to block **1040** at which the flag is tested to determine whether it is −1 or not. If the flag is −1, then the algorithm determines that a system error has occurred (step **1050**) and the process ends at step **1060**. If the flag at step **1040** is not −1, then the file has not been found (step **1070**). Thus after step **1070** the algorithm returns to step **1020** to find the next file or retry.

If at step **1030**, the flag is set to 1 indicating that the file was found, then the content of the file is retrieved using a program like that in Table 3, and it is stored at a specified address. A flag is returned if this process succeeds set equal to 1 otherwise it is set equal to 0 (step **1080**). Next the flag is tested (step **1090**). If the flag is not equal to 1 then reading

of the file failed (step **1100**). The process then returns to step **1020** to find the next file or retry.

If the flag is set to 1, indicating that the file has been successfully loaded into the dynamic RAM of the target device, then the "Surf_HTML" process is executed (step **1110**). The details of this process are illustrated in FIG. **8**B. Next the current page URL name is updated according to the HTML process (step **1120**). After updating the current URL name, the process returns to step **1020** to find the next file.

FIG. **8**B illustrates the "Surf_HTML" process of step **1110** in FIG. **8**A. This process starts at point **1200**. The first part is initialization step **1210**. A display routine is executed at step **1220** having the fixed coordinate functions of the precompiled HTML data set. First, the process determines whether applets are included in the file (step **1230**). If they are included, then the applet is executed (step **1240**). If no applets are included or after execution of the applet, then a refresh flag is tested (step **1240**). If the flag is equal to 1, then it is tested whether a timeout has occurred (step **1250**). If a timeout has occurred, then the current page is updated (step **1260**) and the process returns set **1210** of FIG. **8**B, for example.

If at block **1240** the refresh flag was not equal to 1, or at block **1250** the timeout had not expired, then the process proceeds to step **1270** to get a user supplied input code such as an infrared input signal provided by a remote control at the target device code. In response to the code, a variety of process are executed as suits a particular target platform to handle the user inputs (step **1280**). The process returns a GO_HOME, or a PLAY_URL command, for example, which result in returning the user to a home web page or to a current URL, respectively. Alternatively the process loops to step **1270** for a next input code.

As mentioned above, FIG. **4** illustrates the JAVA precompiler according to the present invention. The JAVA precompiler receives standard full feature JAVA byte codes as input on line **600**. Byte codes are parsed at block **601**. A JAVA class loader is then executed at block **602**. The classes are loaded into a command module **603** which coordinates operations of a JAVA virtual machine **604**, a JAVA garbage collection module **605**, and a JAVA objects memory mapping optimizing engine **606**. The output is applied by block **607** which consists of a compiled JAVA bytecode format according to the present invention.

The process is illustrated in FIG. **9** beginning at block **1500**. First the JAVA bytecode file is loaded (block **1510**). Next, the JAVA classes are loaded based on the interpretation of the bytecode (step **1520**). Next the classes are optimized at step **1530**. After optimizing the classes, the byte codes are translated to a reduced bytecode (step **1540**). Finally the reduced bytecode is supplied (step **1550**) and the algorithm stops at step **1560**. Basically the process receives a JAVA source code file which usually has the format of a text file with the extension JAVA. The JAVA compiler includes a JAVA virtual machine plus compiler classes such as SUN-.TOOLS.JAVAC which are commercially available from Sun Micro Systems. The JAVA class file is parsed which typically consists of byte codes with the extension .CLASS. A class loader consists of a parser and bytecode verifier and processes other class files. The class structures are processed according to the JAVA virtual machine specification, such as the constant pool, the method tables, and the like. An interpreter and compiler are then executed. The JAVA virtual machine executes byte codes in methods and outputs compiled JAVA class files starting with "Main". The process of loading and verifying classes involves first finding a class. If the class is already loaded a read pointer to the class is returned, if not, the class is found from the user specified class path or directory, in this case a flash memory chunk. After finding the class, the next step is executed. This involves loading the bytes from the class file. Next, class file bytes are put into a class structure suitable for run time use, as defined by the JAVA virtual machine specification. The process recursively loads and links the class to its super classes. Various checks and initializations are executed to verify and prepare the routine for execution. Next, initialization is executed for the method of the class. First the process ensures that all the super classes are initialized, and then cause the initialization method for the class. Finally, the class is resolved by resolving a constant pool entry the first time it is encountered. A method is executed with the interpreter and compiler by finding the method. The method may be in the current class, its super class or other classes as specified. A frame is created for the method, including a stack, local variables and a program counter. The process starts executing the bytecode instructions. The instructions can be stack operations, branch statements, loading/storing values, from/to the local variables or constant pool items, or invoking other methods. When an invoked method is a native function, the implemented platform dependent function is executed.

In FIG. **9**A, the process of translating JAVA byte codes into compiled byte codes (step **1504** of FIG. **9**) is illustrated. According to the process FIG. **9**A, the high level class byte codes are parsed from the sequence. For example, Windows dialog functions are found (**1570**). The high level class is replaced with its lower level classes (**1580**). This process is repeated until all the classes in the file become basic classes (**1590**). After this process, all the high level functions have been replaced by lower level level basic functions, such as draw a line, etc. (**1600**).

JAVA byte codes in classes include a number of high level object specifying functions such as a window drawing function and other tool sets. According to the present invention, these classes are rendered by the precompiler into a set of specific coordinate functions such as those outlined above in connection with the HTML precompiler. By precompiling the object specifying functions of the JAVA byte code data set, significant processing resources are freed up on the thin client platform for executing the other programs carried in a JAVA byte code file. Furthermore, the amount of memory required to store the run time engine and JAVA class file for the thin client platform according to the present invention which is suitable for running a JAVA byte code file is substantially reduced.

FIG. **10** illustrates one environment in which use of the present invention is advantageous. In particular, in the Internet environment a wide variety of platforms are implemented. For example, an end user workstation platform **100** is coupled to the Internet **101**. An Internet server platform **102** is also coupled to the Internet **101** and includes storage for JAVA data sets, HTML data sets, and other image files. A server **103** with an intermediate compiler according to the present invention for one or more of the data sets available in the Internet is coupled to the Internet **101** as well. A variety of "thin" platforms are also coupled to the Internet and/or the server **103**. For example, an end user thin platform A **104** is coupled to the server **103**. End user thin platform B **105** is coupled to the server **103** and to the Internet **101**. End user thin platform C **106** is coupled to the Internet **101** and via the Internet all the other platforms in the network. A variety of scenarios are thus instituted. The source of data sets for end user platform C **106** consists of the World Wide Web. When it requests a file from server

102, the file is first transferred to the intermediate compiler at server 103, and from server 103 to the end user platform 106. End user platform A 104 is coupled directly to the server 103. When it makes a request for a file, the request is transmitted to the server 103, which retrieves the file from its source at server 102, translates it to the compiled version and sends it to platform A 104. End user platform B is coupled to both the server 103 and to the Internet 101. Thus, it is capable of requesting files directly from server 102. The server 102 transmits the file to server 103 from which the translated compiled version is sent to platform B 105. Alternatively, platform B may request a file directly from server 103 which performs all retrieval and processing functions on behalf of platform B.

FIG. 11 illustrates an alternative environment for the present invention. For example, the Internet 120 and an Intranet 121 are connected together. A server 122 is coupled to the Intranet 121 and the Internet 120. The server 122 includes the HTML and JAVA intermediate compiling engines according to the present invention as represented by block 123. The server 122 acts as a source of precompiled data sets for thin client platforms 124, 125 and 126 each of which has a simplified run time engine suitable for the compiled data sets. Thus the powerful HTML/JAVA engine resides on the network server 122. The thin network computers 124, 125, 126 are connected to the server have only the simplified run time engine for the compiled image set. Thus, very small computing power is required for executing the display. Thus computing tasks are done using the network server, but displayed on a thin network computer terminals 124–126.

FIGS. 12A and 12B illustrate the off-line environment for use of the present invention. In FIG. 12A, the production of the compiled files is illustrated. Thus, a standard object file, such as an HTML or JAVA image, is input online 1300 to a compiler 1301 which runs on a standard computer 1302. The output of the compiler on line 1303 is the compiled bitmap, compiled HTML or compiled JAVA formatted file. This file is then saved on a non-volatile storage medium such as a compact disk, video compact disk or other storage medium represented by the disk 1304.

FIG. 12B illustrates the reading of the data from the disk 1304 and a thin client such as a VCD box, a DVD box or a set top box 1305. The run time engine 1306 for the compiled data is provided on the thin platform 1305.

Thus, off-line fill feature HTML and JAVA processing is provided for a run time environment on a very thin client such as a VCD/DVD player. The standard HTML/JAVA objects are pre-processed and compiled into the compiled format using the compiler engine 1301 on a more powerful computer 1302. The compiled files are saved on a storage medium such as a floppy disk, hard drive, a CD-ROM, a VCD, or a DVD disk. A small compiled run time engine is embedded or loaded into the thin client device. The run time engine is used to play the compiled files. This enables use of a very small client for running full feature HTML and JAVA programs. Thus, the machine can be used in both online, and off-line modes, or in a hybrid mode.

The foregoing description of a preferred embodiment of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

1. A method of translating a document on a first device for use on a second device, the document being in a standard HTML language, the method comprising:
   - reading the document;
   - reading a profile describing characteristics of the second device, the profile including a display resolution and a supported image format; and
   - translating the document on the first device according to the profile, the translating including
     - retrieving a plurality of images referenced by the document,
     - generating a color palette for the second platform using the plurality of images and the document,
     - executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device,
     - translating the plurality of images from respective formats to the supported image format, and
     - outputting a translated document, the translated document including at least a reference to the color palette, the plurality of images in the supported image format, and the plurality of drawing instructions.

2. The method of claim 1 wherein the reading the document further comprises retrieving the document from a world wide web (WWW) site based on a uniform resource locator (URL).

3. The method of claim 1 wherein the profile includes a maximum number of colors for the color palette.

4. The method of claim 3 wherein the generating the color palette using the plurality of images and the document comprises:
   - creating a set of colors comprised of all colors used in the plurality of images and all colors used in the document;
   - reducing the set of colors to contain no more than the maximum number of colors for the color palette.

5. The method of claim 1 wherein the document includes a plurality of references to a plurality of images, each of the plurality of references comprising a URL, and the retrieving a plurality of images referenced by the document further comprises retrieving respective images using the plurality of references.

6. The method of claim 1 wherein the executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device further comprises:
   - executing the document for display on the second device according to the display resolution;
   - positioning HTML elements in the document according to the display resolution;
   - word wrapping HTML text elements in the document according to the display resolution; and
   - generating a plurality of text drawing elements.

7. The method of claim 6 wherein the generating a plurality of text drawing elements further comprises generating a text element for each text segment, a text segment comprised of one or more characters from the document, the one or more characters sharing a font, a size, a style, and a color, and the text segment occupying not more than one line in the font at the size in the style, each text element including an absolute position at which the text segment should be displayed on the second device.

8. The method of claim 6 further comprising generating a plurality of graphics drawing elements including:

generating a plurality of line elements;

generating a plurality of rectangle elements; and

generating a plurality of circle elements.

9. The method of claim **6** further comprising generating a plurality of link elements, each link element including a URL of a corresponding linked item.

10. The method of claim **1** wherein the supported image format includes a color palette indexed bitmap format and the translating the plurality of images from respective formats to the supported image format comprises:

decoding each of the plurality of images into a red-green-blue bitmap format;

selecting a color in the color palette for pixels in each of the plurality of images; and

outputting a color palette indexed bitmap format for each of the plurality of images.

11. The method of claim **10** wherein the document comprises a plurality of Java classes, and wherein the executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device comprises:

loading and verifying the plurality of Java classes;

initializing methods associated with the plurality of Java classes; and

replacing calls to complex drawing operations with a plurality of graphics drawing elements and a plurality of text drawing elements.

12. The method of claim **1** wherein the translated document includes a plurality of text elements and a plurality of graphics drawing elements.

13. The method of claim **1** wherein the standard HTML language comprises a Java language program.

14. The method of claim **1** wherein the translating the document on the first device for use on the second device further comprises:

receiving a request at the first device over a packet switched network from the second device, the request including a URL;

retrieving the document using the URL responsive to the request; and

providing the translated document to the second device over the packet switched network.

* * * * *

Visualization of Large Terrains in Resource-Limited Computing Environments

Boris Rabinovich Craig Gotsman

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
[borisr|gotsman]@cs.technion.ac.il

Abstract

We describe a software system supporting interactive visualization of large terrains in a resource-limited environment, i.e. a low-end client computer accessing a large terrain database server through a low-bandwidth network. By "large", we mean that the size of the terrain database is orders of magnitude larger than the computer RAM. Superior performance is achieved by manipulating both geometric and texture data at a continuum of resolutions, and, at any given moment, using the best resolution dictated by the CPU and bandwidth constraints. The geometry is maintained as a Delaunay triangulation of a dynamic subset of the terrain data points, and the texture compressed by a progressive wavelet scheme.

A careful blend of algorithmic techniques enables our system to achieve superior rendering performance on a low-end computer by optimizing the number of polygons and texture pixels sent to the graphics pipeline. It guarantees a frame rate depending *only* on the size and quality of the rendered image, independent of the viewing parameters and scene database size. An efficient paging scheme minimizes data I/O, thus enabling the use of our system in a low-bandwidth client/server data-streaming scenario, such as on the Internet.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; D.4.4 [Operating Systems]: Communications Management—Network Communication.
Keywords: Terrain rendering, level-of-detail, interactive graphics

1 Introduction

Terrain visualization is an important component of many civilian and military applications [10, 3]. The input to the terrain visualization problem is usually a large Digital Terrain Map (DTM), consisting of elevation data sampled on a regular grid, and corresponding aerial and/or satellite texture data, which is mapped onto the reconstructed terrain surface. The output is rendered images of the terrain surface, usually as part of a "flythrough" sequence.

The advent of the World-Wide-Web suggests the running of this type of application over the Internet, in a client/server scenario. The server is a very large remote database, accessed by the client, usually a low-end computer, over a narrow-bandwidth line (3 KByte/sec is typical for the contemporary Internet). The two bottlenecks that have to be overcome are the bandwidth in delivering relevant terrain data from the server to the client, and the CPU power required at the client for rendering this data.

The key to efficient terrain *rendering* is efficient online manipulation of both the geometric and texture data, especially when the scene database at the server is orders of magnitude larger that the size of client system RAM. Naive terrain rendering algorithms convert each DTM cell (bounded by four adjacent grid points) into two 3D triangles, and render (send through the graphics pipeline) all such triangles in a region determined by the viewing frustum. They also map the texture data at its highest resolution onto these polygons. This is a very inefficient procedure, as for low pitch angles, the number of these triangles and texture pixels (*texels*) may be extremely large. Each individual triangle projection to image space is very small, and many texels may be condensed to one image pixel, contributing negligibly to the image. One remedy to this problem, adopted in a number of works over the past few years (e.g. [8]) is to maintain the scene data at a number of discrete *levels-of-detail*. Since terrain areas at large viewing distances project to small image areas, there is no point rendering them in full detail. At any given moment during the animation, the appropriate level-of-detail is used to render the image. To do this effectively, pieces of the scene must be taken from multiple levels (foreground areas from a high-detail version, and background areas from a low-detail version), requiring methods to "stitch" together pieces of different models in a continuous fashion, so that there are no holes or breaks along the seams. This has proven to be a major problem for the geometric data, since there usually is no topological correlation between the different levels of detail. De Berg and Dobrint [1], Cohen-Or and Levanoni [5], and Lindstrom et al. [12] have provided partial solutions to the stitching problem.

In this paper we use a different approach to maintaining the terrain geometry, proposed independently by Klein and Huttner [11] and Delepine [6]. The geometry is treated in a *continuous-resolution* fashion. We do not maintain multiple geometric models (at different levels of detail), rather continuously update one model online to represent in an optimal way the projection of the terrain contained in the viewing frustum. As a result, the number of poly-

gons in the approximation is more or less constant, independent of the viewing parameters (for a fixed frame rate). For the texture,we employ a progressive wavelet compression scheme [2], which enables the extraction of texture at a continuum of resolutions from arbitrary prefixes of the encoded bit stream.

Our ultimate goal is to render any terrain image in time proportional to the *image* resolution (in pixels), and not to the scene complexity, number of DTM points in the viewing frustrum, texture resolution, etc. We are motivated by the (simple) observation that an image of fixed resolution can contain only a bounded amount of information, therefore any algorithm rendering such an image should not use more than a bounded number of polygons and texels. Such algorithms are called *output-sensitive*. Most algorithms are not output-sensitive, and in order that they be such, require careful design. Our system contains a careful blend of techniques, some borrowed from computational geometry, which together achieve a high degree of output sensitivity, enabling adequate performance in a limited-resource environment.

Since one server may be accessed simultaneously by a large number of clients, is is crucial to minimize the amount of work the server performs per client. If this load is minimized, the server will be scalable, able to support a virtually unlimited number of clients. We adhere to this principle throughout our implementation.

Using these methods, we have developed a client application achieving terrain visualization at interactive rates on a low-end SGI (O_2)workstation, accessing a server database over a network with bandwidth comparable to the Internet. This paper describes the architecture and algorithms incorporated into our system.

2 System Overview

The large terrain scene resides on the server disk, partitioned into geometry and texture *tiles* of fixed size. A raw geometry tile contains a matrix of elevation heights, and a texture tile a matrix of texels. Tiling schemes are standard in terrain visualization applications (e.g. [4]). The server processes requests for geometry and texture data received from remote clients. In a preprocessing step at the server, applied independently to each tile (thus enabling a scene consisting of an unlimited number of tiles), the DTM points are assigned "grades" related to their importance in approximating the terrain surface. These grades are obtained from the *simplification* algorithm of Heckbert and Garland [9]. Using these grades as a third dimension, the DTM points in each tile are organized into a 3D octree, which will enable efficient answers to future geometric queries. The client maintains online a geometry cache containing DTM points from a small subset of the server's geometry tiles. Even from these tiles, only the relevant upper levels of the corresponding octrees are imported to the client. Which levels are relevant is determined on the fly by the client.

At any given moment, a subset of the geometry cache points are maintained at the client in a dynamic Delaunay triangulation, our primary geometric data structure. To maintain the triangulation, we use the algorithms of Devillers, Meiser and Teillaud [7] for efficient insertion and deletion into a 2D Delaunay triangulation. Delaunay triangulations are commonly considered to be suitable for terrain visualization purposes. A DTM point deserves to be in the triangulation if its grade is greater than a threshold, which is proportional to the distance of the point from the viewpoint. Section 3 elaborates on the details of how we handle the geometry.

The texture data is maintained at the server in tiles, compressed using the progressive wavelet scheme of Buccigrossi and Simoncelli [2]. This scheme compresses the data to approximately 30% of its raw size with negligble loss, and, more important, allows the decoding of the texture data from any prefix of the bit stream. Naturally, using more bits will result in a higher quality result. Client requests for texture data at a given resolution result in the streaming of the prefix of minimal length sufficing for the required resolution. Section 4 describes our handling of the texture in more detail.

The client graphics pipeline, sometimes supported in hardware, is fed relevant triangles and texels. This pipeline takes care of the basic rendering operations, e.g. perspective projection, hidden surface elimination, and texture mapping. The main issues we address in our implementation are the minimization of data transmitted from the server to the client caches and subsequently fed to the graphics pipeline.

Typical triangulations and rendered images generated by our client system are shown in Fig. 2.

3 Geometry Processing

3.1 Data Reduction

A typical DTM is supplied on a regular grid, and this data is usually highly redundant. If the surface is to be approximated by a piecewise-linear 2D function (a collection of planar polygons), a small number of large polygons suffice to approximate the surface well in planar regions. On the other hand, terrain areas with high curvature, such as ridges and ravines, require a large number of small polygons to achieve a satisfactory approximation (see Fig. 2). By this argument, is it obvious that some DTM points are more important than others. Heckbert and Garland [9] have described a procedure which starts off with a small number of DTM points (usually the four corners of the DTM coverage), and incrementally adds points whose contribution to the surface approximation is most significant. The contribution of a point to the approximation is quantified by its vertical distance from the piecewise-linear approximation built with all previous points. The larger this distance - the more important the point is. The incremental procedure is done efficiently using a priority queue mechanism.

We use the Heckbert and Garland procedure at the server as a preprocessing operation on each tile to assign each DTM point a numeric "grade" - precisely the vertical distance described in the previous paragraph. This grade is stored with the point, and used later to determine online whether the point is required for the terrain approximation. This decision is based on the grade and the point's distance from the viewpoint. To facilitate efficient decision-making, we build a 3D octree of the DTM points, the grade serving as the third dimension. The grid structure of the points in the XY plane facilitates a fixed quadtree structure in this plane, which, in turn, facilitates the organization of the data stored in the tile in a

record of fixed length. This hierarchical spatial data structure will enable efficient range reporting of points.

3.2 View Frustum Culling

The first step in frame generation is to determine which DTM tiles are relevant to the current view. In principle, if the terrain surface were planar, the intersection of the viewing frustum with the terrain surface (the view *footprint*) would be a trapezoid, whose four vertex positions could be easily computed (see Fig. 3). Since the terrain surface is not planar, the footprint terrain is bounded by a region which is the union of *two* trapezoids, formed on horizontal planes whose elevations coincide with the minimal and maximal elevations in the projection area, repectively.

The footprint is "scan-converted" by the client to determine which DTM tiles intersect it, and what resolution data (which levels of the octree) are required. This data is requested from the server. For every tile received, the octree structure of its points enables to efficiently determine which tile points are actually contained in the footprint. Efficiency is achieved by pruning off large sets of the points corresponding to branches of the octree close to its root. The remaining points are then tested, as described in Section 3.3, to determine if they are required for the terrain approximation and rendering.

3.3 Continuous Resolution

Each DTM point has a grade quantifying its importance in the terrain approximation. This grade is traded off with distance from the viewpoint. In other words, more distant points are considered less significant. In practice, the client considers a virtual cone centered at the viewpoint, and calculates which DTM points in the geometry cache have a grade positioning them *inside* the cone (see Fig. 3). We would like to be able to determine this set of points in time proportional mainly to their number (and not to the total number of points in the viewing frustum). In computational-geometric terminology, this is called *output-sensitive range reporting*. We achieve this again using the tile octree. The complexity of the range reporting procedure is $O(\sqrt{N}+k)$, where N is the number of points in the viewing frustum, and k the number of points in the answer to the query ([13], p.79). Using this virtual cone also implies that a small change in the viewpoint induces a small change in the DTM points used for the rendering, thus ensuring the temporal continuity of the rendered images.

3.4 Caching

Portions of geometry tiles are imported from the server on demand and stored in the client cache. Only the neccesary upper levels of the tile octree are imported, possible due to the fixed structure of the octree. Hence a typical snapshot of the client cache contents would reveal a few (foreground) tiles from which almost the entire data content has been read, and many (background) tiles with a very sparse content. A prediction mechanism, based on the viewpoint trajectory, enables the loading of tiles in advance, resulting in smooth streaming of geometry from server to client.

3.5 Dynamic Delaunay Triangulation

The piecewise linear surface induced by the *Delaunay triangulation* of the 2D projection of the DTM points is generally considered the most suitable for surface approximation. This is because the minimal angle in the triangulation is maximized, eliminating long "slivery" triangles. Hence, the client constantly maintains a Delaunay triangulation of the DTM points contributing to the approximation of the terrain in the footprint. Many $O(n \log n)$ time algorithms exist for the Delaunay triangulation of n points, but not many are able to efficiently support update of the triangulation upon insertion or deletion of points. We use the algorithm of DeVillers et al [7], which inserts points in $O(\log n)$ and deletes points in $O(\log \log n)$ average time using a hierarchical data structure. Care must be taken to slightly perturb the spatial positions of the DTM points, otherwise degeneracies in the Delaunay triangulation and unstable numerics may occur.

At the client, points which were in the footprint corresponding to the previous frame, and are no longer in the current footprint, are removed from the triangulation - the main geometric data structure maintained online by the client. New points which were previously not in the footprint, and now are, are inserted into the triangulation. The turnover of points in the triangulation depends on the viewpoint velocity. Theoretically, very large velocities could cause successive frames to see totally different regions of the terrain, requiring the formation of an entirely different triangulation between frames. In practice, however, this does not occur. Typically, 99% of the footprint areas overlap between successive frames.

Pseudo-code of the flow of control in the client while rendering a single frame appears in Fig. 1.

4 Texture Processing

The texture data must also be manipulated at multiple resolutions, since image foreground pixels contain high resolution texels, and image background pixels contain low resolution texels. The resolution of the texels contributing to any given image pixel is essentially a function of the viewing distance to that scene point. The server texture database is also organized in tiles, storing the texels compressed to approximately 30% of their original volume, using a progressive wavelet scheme. This results in a bit stream sorted by importance.

A typical low-end client computer contains a texture buffer of limited capacity (e.g. 1024x1024 pixels) with a pyramid structure on top of it. By supplying appropriate texture coordinates for the rendered triangle vertices, the graphics hardware/software maps texels from the texture buffer to the image pixels in the interior of the projected triangles. Each level of the texture pyramid contains texels representing the same terrain area, at decreasing resolutions. However, since not all texels, especially not at *all* resolutions, will contribute to the terrain image (see Fig. 4), there is no need to import them from the server. We optimize network bandwidth by loading *only* those texture tiles which intersect the view footprint, at the appropriate resolution, if they are not yet loaded. By this we mean we calculate the number of encoded bits of the texture stream required to reconstruct the texture tile at the appropriate res-

olution (the lower the required resolution, the less bits required). In any case, we use any bits available at rendering time, even though there might be less than required (if the network temporarily slows down). Which tiles are relevant can be easily determined from the geometry of the footprint. Occasionally, it is neccesary to shift the contents of the texture buffer, due to the movement of the viewpoint.

5 Experimental Results

We have implemented the procedures described in Sections 2 - 4 as a prototype client/server system, the client running on a R5000 SGI O_2, at 180MHz with 64MB RAM, based on the OpenGL API, and an X/Motif GUI. This client accesses the scene database server over a 3 KByte/sec network. The main parameters influencing the overall performance of the system are the size of the visualization window, i.e. the number of rendered image pixels, and the flight velocity. This performance is measured in the client frame rate, and the quality of the imagery delivered at that frame rate. There is an obvious tradeoff between the two, which is controlled by two independent "resolution" parameters, one for geometry, and one for texture. Increasing these parameters increases the number of triangles and/or texture bytes used for the rendering process, thus increasing the image quality, but decreasing the frame rate, due to higher rendering and bandwidth overhead. There is, however, a point beyond which the resolution parameter saturates, i.e. the marginal increase in image quality is insignificant.

The geometric resolution parameter, namely, the average number of triangles rendered per image pixel, is controlled by the angle of the cone used for culling DTM points, as described in Section 3.3. The smaller the angle, the narrower the cone, admitting less DTM points into the Delaunay triangulation, in turn implying less triangles for the same number of image pixels (see also Fig. 3). The texture resolution is controlled by specifying the fraction of the texture tile bit stream imported and decoded to texels for the foreground image pixels. The resolution of the background image pixels is derived from this.

Keeping the resolution parameters and velocity fixed causes the system to maintain a fixed frame rate. Increasing the velocity would slow down the system, as the turnover of points in the Delaunay triangulation and turnover of texture tiles in the texture buffer increases, incurring more CPU and bandwidth overhead. By trial and error, it seems that reasonable image quality is obtained at a geometric resolution of 0.06 triangles and 0.5 texture bytes per output image pixel. Any more than that imposes an unneccesary load on the system, slowing it down, and any less than that results in poor quality images (see Fig. 2). A telltale sign of insufficient geometric resolution (triangles per image pixel) is if there are "jumps" (also known as "popping") in the terrain surface during animation, due to the triangles being too large and crude. A telltale sign of insufficient texture resolution (texels per image pixel) are blurred images.

Fig. 5 shows the speed/quality tradeoffs we are able to achieve with our system at different "flight" velocity parameters, when one of the geometric/texture resolution parameters is fixed, and the other varied. Velocity is measured as the percentage of non-overlapping area between footprints corresponding to successive frames. The figure shows that approximately 3 frames/sec are achievable with reasonable quality, when the image size is fixed at 300x400 pixels, and flying at an average (3%) velocity. Higher velocities result in a larger turnover of geometry and texture, slowing down the system frame rate. Our system accesses a scene database server covering the northern part of Israel, containing a total of 10^7 DTM points and 10^8 texels. The client uses a geometry cache of size 2MB RAM, and texture buffer of 1024x1024 texels.

6 Conclusion

In the long-term, our techniques will support client/server terrain visualization applications over the Internet. A large scene database resides at a central server site, and is accessed (perhaps simultaneously) by a number of low-end clients over the Internet for visualization purposes. This application requires tight optimization of the available network bandwidth and client rendering power.

The ever-increasing user appetite for larger and richer geometric scenes has forced computer graphics practitioners to develop output-sensitive rendering algorithms whose computational complexity is not sensitive to the complexity of the input scene, rather to the complexity of the output image. We have implemented this for the terrain visualization application by rendering at geometric and texture level-of-detail which changes continuously along the spatial and temporal dimensions. Our algorithm satisfies almost all of the five requirements from such an algorithm, as formulated in [12].

Use of other sophisticated data optimization techniques, such as *occlusion culling* [14], in which large portions of the geometry inside the view frustrum are efficiently culled because they are invisible, can further reduce the rendering load.

Temporal aliasing sometimes occurs in our implementation. The use of *morphing* techniques to alleviate this, such as that of Cohen-Or and Levanoni [5], are not directly applicable, again due to the dynamic nature of our Delaunay triangulation. Alternatives are being investigated.

Acknowledgements

We thank Olivier DeVillers for providing code implementing the algorithm of [7], Paul Heckbert for code implementing the algorithm of [9], and R. Buccigrossi for code implementing the algorithm of [2].

This research was supported by the Technion V.P.R. Fund - Promotion of Sponsored Research.

References

[1] M. De Berg and K. Dobrindt. On levels of detail in terrains. In *11th Annual ACM Symposium on Computational Geometry*. ACM, 1994.

[2] R.W. Buccigrossi and E.P. Simoncelli. Progressive wavelet image coding based on a conditional probability model. In *Proceedings of Int'l Conf. Acoustics Speech and Signal Processing*. IEEE, 1997.

[3] D. Cohen and C. Gotsman. Photorealistic terrain imaging and flight simulation. *IEEE Computer Graphics and Applications*, 14(2):10–12, March 1994.

[4] D. Cohen-Or, U. Lerner, E. Rich, and V. Shenkar. A real-time photo-realistic visual flythrough. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):255–265, September 1996.

[5] D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in Delaunay triangulated terrain. In *Proceedings of Visualization '96*. IEEE Computer Society Press, 1996.

[6] T. Delepine. Online terrain level-of-detail. In *Proceedings of ITECH*, 1997.

[7] O. Devillers, S. Meiser, and M.Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2:55–80, 1992.

[8] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Computer Graphics and Applications*, 9(2):67–78, 1989.

[9] P. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, School of Computer Science,Carnegie Mellon University,Pittsburg ,PA , 15213, 1995.

[10] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, Tanaka, and Nogushi. Three-dimensional terrain modeling and display for environmental assessment. *Computer Graphics (Proceedings of SIGGRAPH'89)*, 23(3):207–214, 1989.

[11] R. Klein and T. Huttner. Simple camera-dependent approximation of terrain surfaces for fast visualization and animation. In *Proceedings of Visualization '96 (late breaking topics)*. IEEE Computer Society Press, 1996.

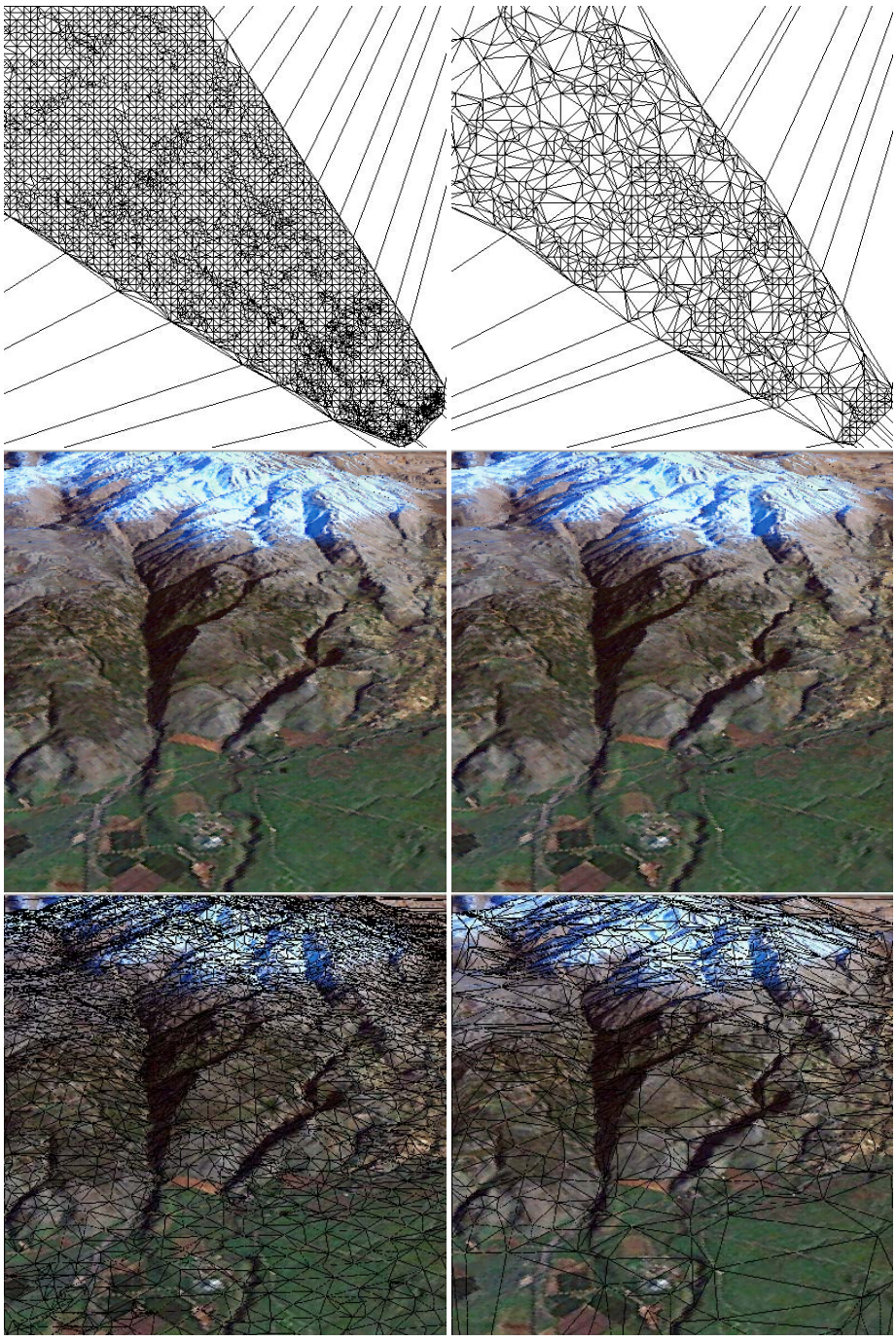
[12] P. Lindstrom, D. Koller, L.F. Hodges W. Ribarsky, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of SIGGRAPH '96*, 1996.

[13] M. Shamos and F. Preparata. *Computational Geometry*. Springer, 1989.

[14] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, 1996 (Proceedings of Eurographics, Poitiers, France, August 1996).

1. Calculate view frustum and bound terrain footprint by rectangle.
2. Scan-convert the rectangle and for each geometry tile in it:
   (a) If the tile is not in the footprint, but was in it in the previous frame, then:
      - Remove all its points from the Delaunay triangulation.
   (b) If the tile is in the footprint, but was not in the previous frame, then:
      - Request tile from server at appropriate resolution.
      - Search in tile octree for appropriate voxels.
      - Insert the points from these voxels in Delaunay triangulation.
   (c) If tile is in the footprint and was also in the previous frame, then:
      - Search in tile octree for appropriate voxels.
      - Find difference from previous frame.
      - Insert (Delete) difference points in (from) Delaunay triangulation.
3. For each texture tile in the bounding rectangle:
   (a) If the texture tile is in the footprint, but was not in the previous frame, then:
      - Calculate required resolution.
      - Request the appropriate bit stream prefix from the server.
   (b) If texture tile is in the footprint, and was also in the previous frame, then:
      - Calculate its resolution.
      - If this resolution is higher than that of the previous frame, then request more of the bit stream from the server.
4. For every tenth frame check the actual performance (frames/sec) against the required performance and adjust the geometric and/or texture resolution parameters to achieve that performance.
5. Render image.

Figure 1: Pseudo-code of the client algorithm.



(a) (b)

Figure 2: Terrain meshes (Delaunay triangulated) and views rendered at different data resolutions. (a) High resolution: 0.08 triangles/pixel and 1 texels/pixel. (b) Equivalent quality at lower resolution: 0.02 triangles and 0.8 texels/pixels. Note how more DTM points are used in foreground areas or areas of high curvature.

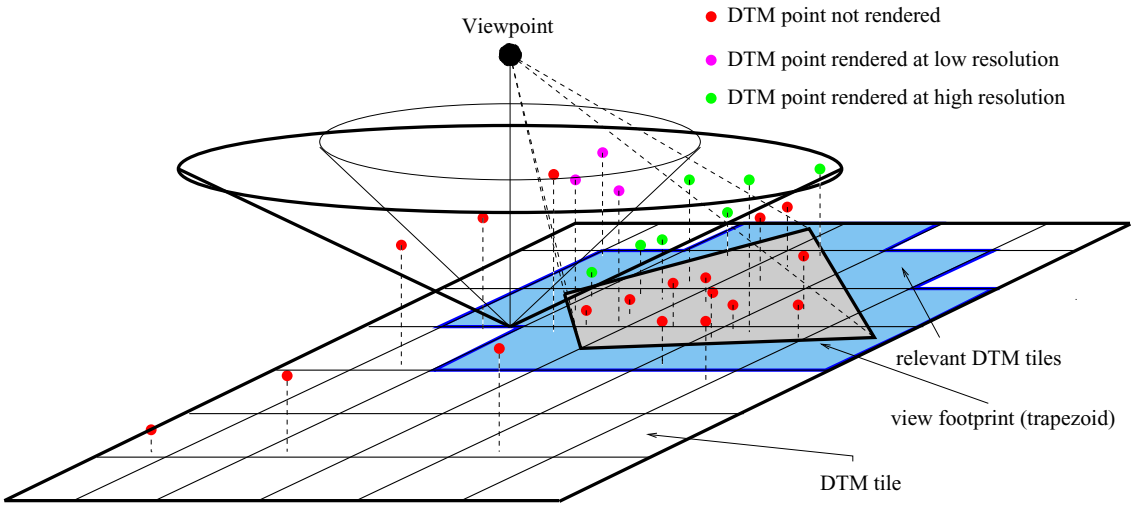


Figure 3: Determining the DTM points of the rendered Delaunay triangulation for a given view at different geometric resolutions. The narrow cone represents a low-resolution view, and the wide one a high resolution. The "elevations" of the DTM points are their precalculated grades. All points within the footprint with grade above the relevant cone are included in the triangulation. This range-reporting operation is performed efficiently using an octree structure on the points in each tile. Note that more points are admitted in the view foreground than in its background.

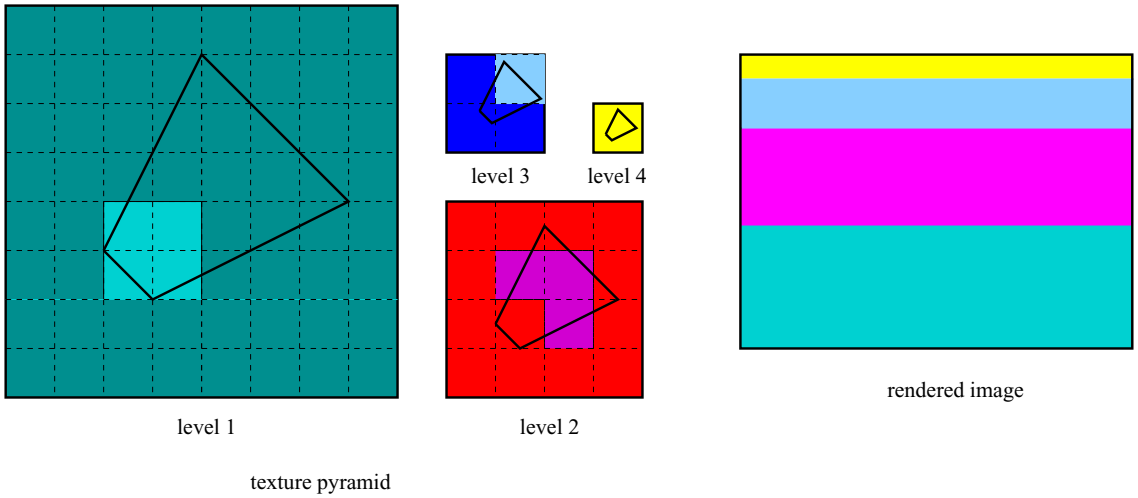
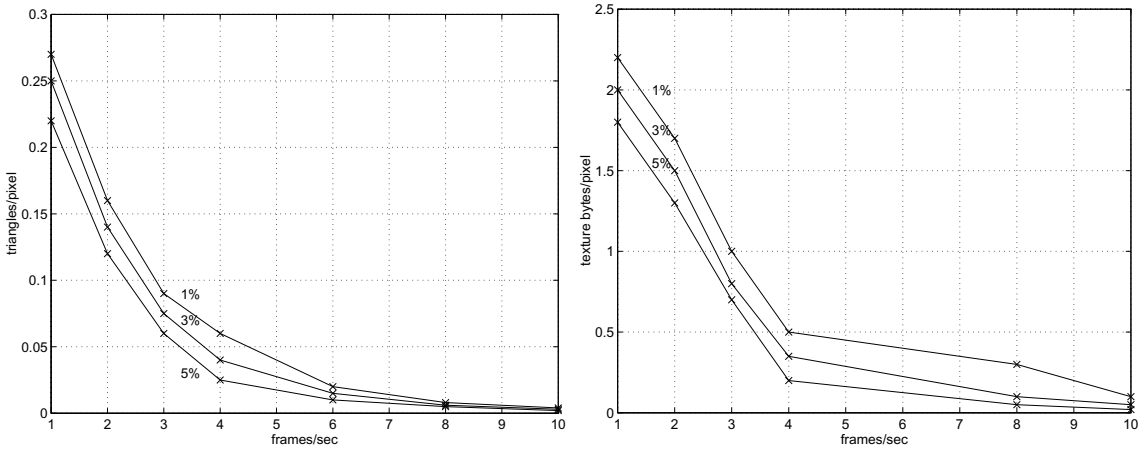


Figure 4: The contribution of individual tiles in the texture buffer to the rendered image corresponding to the marked footprint. Those tiles not contributing need not reside in the texture buffer at all, and are not streamed and decoded from the server.



(a) (b)

Figure 5: Speed/resolution tradeoff in our prototype visualization client while rendering 300x400 pixel images on a R5000 SGI O_2, accessing the scene database server over a 3 KByte/sec network. (a) Varying only geometric resolution. The texture resolution is fixed to 0.5 compressed texture bytes per pixel. (b) Varying only texture resolution. The geometric resolution is fixed to 0.06 triangles/pixel. The individual curves correspond to different flight velocities, which influence the turnover of data in system caches and bandwidth overhead.

PROCEEDINGS
Visualization '97

October 19 – 24, 1997

Phoenix, Arizona

Sponsored by
IEEE Computer Society Technical Committee on Computer Graphics

In cooperation with
ACM SIGGRAPH

The Association for Computing Machinery
1515 Broadway
New York, NY 10036

APPENDIX R

Copyright © 1997 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

ACM ISBN: 1-58113-011-2
ACM Order Number: 428978

IEEE Computer Society Press Order Number: PR08262
IEEE Catalog Number: 97CB36155
IEEE ISBN: 0-8186-8262-0
IEEE ISBN - Library Binding: 0-8186-8263-9
IEEE ISBN - Microfiche: 0-8186-8264-7
ISSN: 1070-2385

Additional copies may be ordered from:

ACM Order Department
P.O. Box 12114
Church Street Station
New York, NY 10257 USA
Tel: +1-212-626-0500
Fax: +1-212-944-1318
E-mail: orders@acm.org

ACM European Service Center
108 Cowley Road
Oxford OX4 1JF
United Kingdom
E-mail: acm_europe@acm.org

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264 USA
Tel: +1-714-821-8380
Fax: +1-714-821-4641
E-mail: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331 USA
Tel: +1-908-981-1393
Fax: +1-908-981-9667
E-mail: mis.custserv@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
Belgium
Tel: +32-2-770-2198
Fax: +32-2-770-8505
E-mail: euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
Japan
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
E-mail: tokyo.ofc@computer.org

Table of Contents

APPENDIX R

Preface 11
Conference Committee 13
Program Committee 14

Keynote Address: Global Tele-Immersion 15
Tom DeFanti

Capstone Address: Dissolving Descartes: Perception and the Construction of Reality 16
Mark Pesce

Papers

Session 2B: Volume Rendering I

A Comparison of Normal Estimation Schemes 19
Torsten Möller, Raghu Machiraju, Klaus Mueller, Roni Yagel
Color Plate 525

Collision Detection for Volumetric Models 27
Taosong He, Arie Kaufman
Color Plate 526

The VSBUFFER: Visibility Ordering of Unstructured Volume Primitives by Polygon Drawing 35
Rüdiger Westermann, Thomas Ertl
Color Plate 527

Volume Rendering of Abdominal Aortic Aneurysms 43
Roger C. Tam, Christopher G. Healey, Borys Flak, Peter Cahoon
Color Plate 528

Session 3A: Vector Fields

Auralization of Streamline Vorticity in Computational Fluid Dynamics Data 51
Christopher R. Volpe, Ephraim P. Glinert

Singularities in Nonuniform Tensor Fields 59
Yingmei Lavin, Yuval Levy, Lambertus Hesselink

Visualization of Higher Order Singularities in Vector Fields 67
Gerik Scheuermann, Hans Hagen, Heinz Krüger, Martin Menzel, Alyn Rockwood

Principal Stream Surfaces 75
Wenli Cai, Pheng-Ann Heng
Color Plate 529

Session 3B: Terrain Visualization

ROAMing Terrain: Real-time Optimally Adapting Meshes 81
Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein

Visualization of Height Field Data with Physical Models and Texture Photomapping 89
Dru Clark, Michael J. Bailey
Color Plate 530

Visualization of Large Terrains in Resource-Limited Computing Environments 95
Boris Rabinovitch, Craig Gotsman

Building and Traversing a Surface at Variable Resolution 103
Leila De Floriani, Paola Magillo, Enrico Puppo
Color Plate 531

Microsoft Corp. Exhibit 1009

Session 4A: Information Visualization

Multivariate Visualization Using Metric Scaling 111
Pak Chung Wong, R. Daniel Bergeron
Color Plate 532

Visualizing the Behavior of Higher Dimensional Dynamical Systems 119
Rainer Wegenkittl, Helwig Löffelmann, Eduard Gröller
Color Plate 533

Displaying Data in Multidimensional Relevance Space with 2D Visualization Maps 127
Jackie Assa, Daniel Cohen-Or, Tova Milo
Color Plate 534

Session 4B: MultiResolution

Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data 135
Yong Zhou, Baoquan Chen, Arie Kaufman
Color Plate 535

Haar Wavelets over Triangular Domains with Applications to Multiresolution Models for Flow over a Sphere 143
Gregory M. Nielson, Il-Hong Jung, Junwon Sung
Color Plate 536

Wavelet-based Multiresolutional Representation of Computational Field Simulation Datasets 151
Zhifan Zhu, Raghu Machiraju, Bryan Fry, Robert J. Moorhead
Color Plate 537

Session 5A: User Interfaces & Interaction

Dynamic Color Mapping of Bivariate Qualitative Data 159
Penny Rheingans
Color Plate 538

The Contour Spectrum 167
Chandrajit L. Bajaj, Valerio Pascucci, Daniel R. Schikore
Color Plate 539

Constrained 3D Navigation with 2D Controllers 175
Andrew J. Hanson, Eric A. Wernert
Color Plate 540

Session 5B: Volume Rendering II

Two-Phase Perspective Ray Casting for Interactive Volume Navigation 183
Martin L. Brady, Kenneth Jung, HT Nguyen, Thinh Nguyen
Color Plate 541

Accelerated Volume Rendering Using Homogenous Region Encoding 191
Jason L. Freund, Kenneth Sloan
Color Plates 542-543

An Anti-Aliasing Technique for Splatting 197
J. Edward Swan II, Klaus Mueller, Torsten Möller, Naeem Shareef, Roger A. Crawfis, Roni Yagel
Color Plate 544

Microsoft Corp. Exhibit 1009

A Topology Modifying Progressive Decimation Algorithm 205
William J. Schroeder
Color Plate 545

Efficient Subdivision of Finite-Element Datasets into Consistent Tetrahedra 213
Guy Albertelli, Roger A. Crawfis

Interval Volume Tetrahedrization 221
Gregory M. Nielson, Junwon Sung
Color Plate 546

Computing the Separating Surface for Segmented Data 229
Gregory M. Nielson, Richard Franke

Session 6B: Visualization Systems

Application-Controlled Demand Paging for Out-of-Core Visualization 235
Michael B. Cox, David Ellsworth
Color Plate 547

GADGET: Goal-Oriented Application Design Guidance for Modular Visualization Environments 245
Issei Fujishiro, Yuriko Takeshima, Yoshihiko Ichikawa, Kyoko Nakamura
Color Plate 548

Collaborative Visualization 253
Jason D. Wood, Helen Wright, Ken W. Brodlie
Color Plate 549

VizWiz: A Java Applet for Interactive 3D Scientific Visualization on the Web 261
Cherilyn K. Michaels, Michael J. Bailey
Color Plate 550

Session 7A: Data Extraction

Image Synthesis From A Sparse Set of Views 269
Qian Chen, Gérard G. Medioni
Color Plate 551

Virtual*ized* Reality: Constructing Time-Varying Virtual Worlds from Real World Events 277
Peter W. Rander, PJ Narayanan, Takeo Kanade
Color Plate 552

Extracting Feature Lines from 3D Unstructured Grids 285
Kwan-Liu Ma, Victoria L. Interrante
Color Plate 553

I/O Optimal Isosurface Extraction 293
Yi-Jen Chiang, Cláudio T. Silva
Color Plate 554

Microsoft Corp. Exhibit 1009

CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the CAVE Virtual Reality Theater 301
Vijendra S. Jaswal
Color Plate 555

Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet 309
Rainer Wegenkittl, Eduard Gröller

UFLIC: A Line Integral Convolution Algorithm For Visualizing Unsteady Flows 317
Han-Wei Shen, David L. Kao
Color Plate 556

The Motion Map: Efficient Computation of Steady Flow Animations 323
Bruno Jobard, Wilfrid Lefer

Session 8A: Compression

Integrated Volume Compression and Visualization 329
Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, Arie Kaufman
Color Plate 557

Multiresolution Compression And Reconstruction 337
Oliver G. Staadt, Markus H. Gross, Roger Weber
Color Plate 558

Optimized Geometry Compression for Real-time Rendering 347
Mike M. Chow
Color Plate 559

Session 9A: Polygonal Surfaces

Architectural Walkthroughs Using Portal Textures 355
Daniel G. Aliaga, Anselmo A. Lastra
Color Plate 560

Repairing CAD Models 363
Gill Barequet, Subodh Kumar
Color Plate 561

Dynamic Smooth Subdivision Surfaces for Data Visualization 371
Chhandomay Mandal, Hong Qin, Baba C. Vemuri
Color Plate 562

Session 10A: Surface Simplification

Smooth Hierarchical Surface Triangulations 379
Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schlussmann, Isaac J. Trotts

The Multilevel Finite Element Method for Adaptive Mesh Optimization and Visualization of Volume Data 387
Roberto Grosso, Christoph Lürig, Thomas Ertl
Color Plate 563

Simplifying Polygonal Models Using Successive Mappings 395
Jonathan Cohen, Dinesh Manocha, Marc Olano
Color Plate 564

Controlled Simplification of Genus for Polygonal Models 403
Jihad El-Sana, Amitabh Varshney
Color Plate 565

Microsoft Corp. Exhibit 1009

Case Studies

Session 2C: Flow Visualization

Vortex Identification - Applications in Aerodynamics 413
David Kenwright, Robert Haimes
Color Plate 566

exVis 1.0: Developing a Wind Tunnel Data Visualization Tool 417
Samuel P. Uselton
Color Plate 567

Strategies for Effectively Visualizing 3D Flow with Volume LIC 421
Victoria Interrante, Chester Grosch
Color Plate 568

Towards Efficient Visualization Support for Single-block and Multi-block Datasets 425
Jean M. Favre
Color Plate 569

Session 3C: Medical Visualization

Brushing Techniques for Exploring Volume Datasets 429
Pak Chung Wong, R. Daniel Bergeron
Color Plate 570

Interactive Volume Rendering for Virtual Colonoscopy 433
Suya You, Lichan Hong, Ming Wan, Kittiboon Junyaprasert, Arie Kaufman, Shigeru Muraki, Yong Zhou, Mark Wax, Zhengrong Liang
Color Plate 571

DNA Visual And Analytic Data Mining 437
Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, Eugene Stanley
Color Plate 572

An Interactive Cerebral Blood Vessel Exploration System 443
Anna Puig, Dani Tost, Isabel Navazo
Color Plate 573

Session 5C: Educational Visualization

Instructional Software for Visualizing Optical Phenomena 447
David C. Banks, John T. Foley, Kiril N. Vidimce, Ming-Hoe Kiu
Color Plate 574

Wildfire Visualization 451
James Ahrens, Patrick McCormick, James Bossert, Jon Reisner, Judith Winterkamp
Color Plate 575

Visualization of Geometric Algorithms in an Electronic Classroom 455
Maria Shneerson, Ayellet Tal
Color Plate 576

Microsoft Corp. Exhibit 1009

Session 6C: Web & Virtual Reality

Collaborative Augmented Reality: Exploring Dynamical Systems 459
Anton Fuhrmann, Helwig Löffelmann, Dieter Schmalstieg
Color Plate 577

Visualizing Customer Segmentations Produced by Self Organizing Maps 463
Holly Rushmeier, Richard Lawrence, George Almasi
Color Plate 578

Pearls Found on the way to the Ideal Interface for Scanned-probe Microscopes 467
Russell M. Taylor II, Jun Chen, Shoji Okimoto, Noel Llopis-Artime, Vernon L. Chi, Fredrick P. Brooks Jr., Mike Falvo, Scott Paulson, Pichet Thiansanthaporn, Dave Glick, Sean Washburn, Richard Superfine
Color Plate 579

Viewing IGES Files Through VRML 471
Jed Marti

Session 7C: Engineering and Computational Geometry

Visualization of Plant Growth 475
Jeremy J. Loomis, Xiuwen Liu, Zhaohua Ding, Kikuo Fujimura, Michael L. Evans, Hideo Ishikawa
Color Plate 580

Determination of Unknown Particle Charges in a Thunder Cloud Based Upon Detected Electric Field Vectors .479
Dan Drake, Thomas Simpson, Larry Smithmeir, Penny Rheingans
Color Plate 581

Interactive Visualization of Aircraft and Power Generation Engines 483
Lisa Sobierajski Avila, William Schroeder
Color Plate 582

Efficient visualization of physical and structural properties in crash-worthiness simulations 487
Sven Kuschfeldt, Thomas Ertl, Michael Holzner
Color Plate 583

Session 9B: Math & Statistics

Visualization of Rotation Fields 491
Mark A. Livingston
Color Plate 584

Isosurface Extraction Using Particle Systems 495
Patricia Crossno, Edward Angel
Color Plate 585

A Visualization of Music 499
Sean M. Smith, Glen M. Williams

Panels

Terascale Visualization: Approaches, Pitfalls, and Issues 507
Organizers: *Carol Hunter, Roger Crawfis*
Panelists: *Michael Cox, Roger Crawfis, Bernd Hamann, Chuck Hansen, Carol Hunter, Mark Miller*

Information Exploration Shootout Project and Benchmark Data Sets:
Evaluating how Visualization does in Analyzing Real-World Data Analysis Problems 511
Organizer: *Georges Grinstein*
Panelists: *Sharon Laskowski, Bernice Rogowitz, Graham Wills*

Perceptual Measures for Effective Visualizations 515
Organizer: *Holly Rushmeier*
Panelists: *Harrison Barrett, Penny Rheingans, Sam Uselton, Andrew Watson*

Author Index 519
Cover Image Credits 521
Color Plate Section 523

Microsoft Corp. Exhibit 1009

APPENDIX S

User Datagram Protocol (UDP) (Windows CE 5.0)

Windows CE 5.0

Send Feedback

UDP provides a connectionless, unreliable transport service. Connectionless means that a communication session between hosts is not established before exchanging data. UDP is often used for one-to-many communications that use broadcast or multicast IP datagrams. The UDP connectionless datagram delivery service is unreliable because it does not guarantee data packet delivery and no notification is sent if a packet is not delivered. Also, UDP does not guarantee that packets are delivered in the same order in which they were sent.

Because delivery of UDP datagrams is not guaranteed, applications using UDP must supply their own mechanisms for reliability, if needed. Although UDP appears to have some limitations, it is useful in certain situations. For example, Winsock IP multicasting is implemented with UDP datagram type sockets. UDP is very efficient because of low overhead. Microsoft networking uses UDP for logon, browsing, and name resolution. UDP can also be used to carry IP multicast streams for applications such as Microsoft® Windows Media®.

See Also

Core Protocol Stack for IPv4 | User Datagram Protocol (UDP) and Name Resolution for IPv4

Send Feedback on this topic to the authors

Feedback FAQs

© 2006 Microsoft Corporation. All rights reserved.

© 2015 Microsoft

The OpenGL® Graphics System: A Specification (Version 1.2.1)

Mark Segal
Kurt Akeley

Editor (version 1.1): Chris Frazier
Editor (versions 1.2, 1.2.1): Jon Leech

Microsoft Corp. Exhibit 1009

Copyright © 1992-1999 Silicon Graphics, Inc.

This document contains unpublished information of
Silicon Graphics, Inc.

This document is protected by copyright, and contains information proprietary to Silicon Graphics, Inc. Any copying, adaptation, distribution, public performance, or public display of this document without the express written consent of Silicon Graphics, Inc. is strictly prohibited. The receipt or possession of this document does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

U.S. Government Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or in similar or successor clauses in the FAR or the DOD or NASA FAR Supplement. Unpublished rights reserved under the copyright laws of the United States. Contractor/manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.

OpenGL is a registered trademark of Silicon Graphics, Inc.
Unix is a registered trademark of The Open Group.
The "X" device and X Windows System are trademarks of
The Open Group.

Microsoft Corp. Exhibit 1009

Contents

1 Introduction **1**
1.1 Formatting of Optional Features 1
1.2 What is the OpenGL Graphics System? 1
1.3 Programmer's View of OpenGL 2
1.4 Implementor's View of OpenGL 2
1.5 Our View 3

2 OpenGL Operation **4**
2.1 OpenGL Fundamentals 4
2.1.1 Floating-Point Computation 6
2.2 GL State 6
2.3 GL Command Syntax 7
2.4 Basic GL Operation 9
2.5 GL Errors 11
2.6 Begin/End Paradigm 12
2.6.1 Begin and End Objects 15
2.6.2 Polygon Edges 18
2.6.3 GL Commands within **Begin**/**End** 19
2.7 Vertex Specification 19
2.8 Vertex Arrays 21
2.9 Rectangles 28
2.10 Coordinate Transformations 28
2.10.1 Controlling the Viewport 30
2.10.2 Matrices 31
2.10.3 Normal Transformation 34
2.10.4 Generating Texture Coordinates 36
2.11 Clipping 38
2.12 Current Raster Position 40
2.13 Colors and Coloring 43

Microsoft Corp. Exhibit 1009

2.13.1 Lighting 44
2.13.2 Lighting Parameter Specification 49
2.13.3 **ColorMaterial** 51
2.13.4 Lighting State 53
2.13.5 Color Index Lighting 53
2.13.6 Clamping or Masking 54
2.13.7 Flatshading 54
2.13.8 Color and Texture Coordinate Clipping 55
2.13.9 Final Color Processing 56

3 Rasterization **57**
3.1 Invariance 59
3.2 Antialiasing 59
3.3 Points 60
3.3.1 Point Rasterization State 62
3.4 Line Segments 62
3.4.1 Basic Line Segment Rasterization 64
3.4.2 Other Line Segment Features 66
3.4.3 Line Rasterization State 69
3.5 Polygons 70
3.5.1 Basic Polygon Rasterization 70
3.5.2 Stippling 72
3.5.3 Antialiasing 72
3.5.4 Options Controlling Polygon Rasterization 73
3.5.5 Depth Offset 73
3.5.6 Polygon Rasterization State 75
3.6 Pixel Rectangles 75
3.6.1 Pixel Storage Modes 75
3.6.2 The Imaging Subset 76
3.6.3 Pixel Transfer Modes 78
3.6.4 Rasterization of Pixel Rectangles 88
3.6.5 Pixel Transfer Operations 100
3.7 Bitmaps 110
3.8 Texturing 111
3.8.1 Texture Image Specification 112
3.8.2 Alternate Texture Image Specification Commands . . 118
3.8.3 Texture Parameters 123
3.8.4 Texture Wrap Modes 124
3.8.5 Texture Minification 125
3.8.6 Texture Magnification 131

Microsoft Corp. Exhibit 1009

3.8.7 Texture State and Proxy State . . . 131
3.8.8 Texture Objects . . . 132
3.8.9 Texture Environments and Texture Functions . . . 135
3.8.10 Texture Application . . . 138
3.9 Color Sum . . . 138
3.10 Fog . . . 138
3.11 Antialiasing Application . . . 140

4 Per-Fragment Operations and the Framebuffer 141
4.1 Per-Fragment Operations . . . 142
4.1.1 Pixel Ownership Test . . . 142
4.1.2 Scissor test . . . 143
4.1.3 Alpha test . . . 143
4.1.4 Stencil test . . . 144
4.1.5 Depth buffer test . . . 145
4.1.6 Blending . . . 146
4.1.7 Dithering . . . 149
4.1.8 Logical Operation . . . 150
4.2 Whole Framebuffer Operations . . . 150
4.2.1 Selecting a Buffer for Writing . . . 150
4.2.2 Fine Control of Buffer Updates . . . 152
4.2.3 Clearing the Buffers . . . 153
4.2.4 The Accumulation Buffer . . . 155
4.3 Drawing, Reading, and Copying Pixels . . . 156
4.3.1 Writing to the Stencil Buffer . . . 156
4.3.2 Reading Pixels . . . 156
4.3.3 Copying Pixels . . . 162
4.3.4 Pixel Draw/Read state . . . 162

5 Special Functions 164
5.1 Evaluators . . . 164
5.2 Selection . . . 170
5.3 Feedback . . . 173
5.4 Display Lists . . . 175
5.5 Flush and Finish . . . 179
5.6 Hints . . . 179

6 State and State Requests 181
6.1 Querying GL State . . . 181
6.1.1 Simple Queries . . . 181

Microsoft Corp. Exhibit 1009

6.1.2 Data Conversions . . . 182
6.1.3 Enumerated Queries . . . 182
6.1.4 Texture Queries . . . 184
6.1.5 Stipple Query . . . 185
6.1.6 Color Matrix Query . . . 185
6.1.7 Color Table Query . . . 185
6.1.8 Convolution Query . . . 186
6.1.9 Histogram Query . . . 187
6.1.10 Minmax Query . . . 188
6.1.11 Pointer and String Queries . . . 189
6.1.12 Saving and Restoring State . . . 189
6.2 State Tables . . . 193

A Invariance **218**
A.1 Repeatability . . . 218
A.2 Multi-pass Algorithms . . . 219
A.3 Invariance Rules . . . 219
A.4 What All This Means . . . 221

B Corollaries **222**

C Version 1.1 **225**
C.1 Vertex Array . . . 225
C.2 Polygon Offset . . . 226
C.3 Logical Operation . . . 226
C.4 Texture Image Formats . . . 226
C.5 Texture Replace Environment . . . 226
C.6 Texture Proxies . . . 227
C.7 Copy Texture and Subtexture . . . 227
C.8 Texture Objects . . . 227
C.9 Other Changes . . . 227
C.10 Acknowledgements . . . 228

D Version 1.2 **230**
D.1 Three-Dimensional Texturing . . . 230
D.2 BGRA Pixel Formats . . . 230
D.3 Packed Pixel Formats . . . 230
D.4 Normal Rescaling . . . 231
D.5 Separate Specular Color . . . 231
D.6 Texture Coordinate Edge Clamping . . . 231

Microsoft Corp. Exhibit 1009

D.7 Texture Level of Detail Control . . . 232
D.8 Vertex Array Draw Element Range . . . 232
D.9 Imaging Subset . . . 232
D.9.1 Color Tables . . . 232
D.9.2 Convolution . . . 233
D.9.3 Color Matrix . . . 233
D.9.4 Pixel Pipeline Statistics . . . 234
D.9.5 Constant Blend Color . . . 234
D.9.6 New Blending Equations . . . 234
D.10 Acknowledgements . . . 234

E Version 1.2.1 238

F ARB Extensions 239
F.1 Naming Conventions . . . 239
F.2 Multitexture . . . 240
F.2.1 Dependencies . . . 240
F.2.2 Issues . . . 240
F.2.3 Changes to Section 2.6 (Begin/End Paradigm) . . . 240
F.2.4 Changes to Section 2.7 (Vertex Specification) . . . 241
F.2.5 Changes to Section 2.8 (Vertex Arrays) . . . 243
F.2.6 Changes to Section 2.10.2 (Matrices) . . . 244
F.2.7 Changes to Section 2.10.4 (Generating Texture Coordinates) . . . 245
F.2.8 Changes to Section 2.12 (Current Raster Position) . . . 246
F.2.9 Changes to Section 3.8 (Texturing) . . . 246
F.2.10 Changes to Section 3.8.5 (Texture Minification) . . . 248
F.2.11 Changes to Section 3.8.8 (Texture Objects) . . . 248
F.2.12 Changes to Section 3.8.10 (Texture Application) . . . 249
F.2.13 Changes to Section 5.1 (Evaluators) . . . 249
F.2.14 Changes to Section 5.3 (Feedback) . . . 249
F.2.15 Changes to Section 6.1.2 (Data Conversions) . . . 251
F.2.16 Changes to Section 6.1.12 (Saving and Restoring State) 251

Index of OpenGL Commands 256

Microsoft Corp. Exhibit 1009

List of Figures

2.1 Block diagram of the GL. 9
2.2 Creation of a processed vertex from a transformed vertex and current values. 13
2.3 Primitive assembly and processing. 13
2.4 Triangle strips, fans, and independent triangles. 16
2.5 Quadrilateral strips and independent quadrilaterals. 17
2.6 Vertex transformation sequence. 28
2.7 Current raster position. 41
2.8 Processing of RGBA colors. 43
2.9 Processing of color indices. 43
2.10 ColorMaterial operation. 51

3.1 Rasterization. 57
3.2 Rasterization of non-antialiased wide points. 61
3.3 Rasterization of antialiased wide points. 61
3.4 Visualization of Bresenham's algorithm. 64
3.5 Rasterization of non-antialiased wide lines. 67
3.6 The region used in rasterizing an antialiased line segment. . . 69
3.7 Operation of **DrawPixels**. 88
3.8 Selecting a subimage from an image 93
3.9 A bitmap and its associated parameters. 110
3.10 A texture image and the coordinates used to access it. 118

4.1 Per-fragment operations. 142
4.2 Operation of **ReadPixels**. 156
4.3 Operation of **CopyPixels**. 162

5.1 Map Evaluation. 166
5.2 Feedback syntax. 176

Microsoft Corp. Exhibit 1009

F.1 Creation of a processed vertex from a transformed vertex and current values. 241
F.2 Current raster position. 246
F.3 Multitexture pipeline. 249

Microsoft Corp. Exhibit 1009

List of Tables

2.1 GL command suffixes 8
2.2 GL data types 10
2.3 Summary of GL errors 13
2.4 Vertex array sizes (values per vertex) and data types 22
2.5 Variables that direct the execution of **InterleavedArrays**. . 26
2.6 Component conversions 44
2.7 Summary of lighting parameters. 46
2.8 Correspondence of lighting parameter symbols to names. . . . 50
2.9 Polygon flatshading color selection. 55

3.1 **PixelStore** parameters pertaining to one or more of **DrawPixels**, **TexImage1D**, **TexImage2D**, and **TexImage3D**. . 76
3.2 **PixelTransfer** parameters. 78
3.3 **PixelMap** parameters. 79
3.4 Color table names. 80
3.5 **DrawPixels** and **ReadPixels** types 91
3.6 **DrawPixels** and **ReadPixels** formats. 92
3.7 Swap Bytes Bit ordering. 92
3.8 Packed pixel formats. 94
3.9 `UNSIGNED_BYTE` formats. Bit numbers are indicated for each component. 95
3.10 `UNSIGNED_SHORT` formats 96
3.11 `UNSIGNED_INT` formats 97
3.12 Packed pixel field assignments 98
3.13 Color table lookup. 103
3.14 Computation of filtered color components. 104
3.15 Conversion from RGBA pixel components to internal texture, table, or filter components. 114
3.16 Correspondence of sized internal formats to base internal formats. 115

Microsoft Corp. Exhibit 1009

3.17 Texture parameters and their values. 124
3.18 Replace and modulate texture functions. 136
3.19 Decal and blend texture functions. 137

4.1 Values controlling the source blending function and the source blending values they compute. $f = \min(A_s, 1 - A_d)$. 148
4.2 Values controlling the destination blending function and the destination blending values they compute. 148
4.3 Arguments to **LogicOp** and their corresponding operations. . 151
4.4 Arguments to **DrawBuffer** and the buffers that they indicate. 152
4.5 **PixelStore** parameters pertaining to **ReadPixels**, **GetTexImage1D**, **GetTexImage2D**, **GetTexImage3D**, **GetColorTable**, **GetConvolutionFilter**, **GetSeparableFilter**, **GetHistogram**, and **GetMinmax**. 158
4.6 **ReadPixels** index masks. 160
4.7 **ReadPixels** GL Data Types and Reversed component conversion formulas. 161

5.1 Values specified by the *target* to **Map1**. 165
5.2 Correspondence of feedback type to number of values per vertex. 174

6.1 Texture, table, and filter return values. 185
6.2 Attribute groups 191
6.3 State variable types 192
6.4 GL Internal begin-end state variables (inaccessible) 194
6.5 Current Values and Associated Data 195
6.6 Vertex Array Data 196
6.7 Transformation state 197
6.8 Coloring 198
6.9 Lighting (see also Table 2.7 for defaults) 199
6.10 Lighting (cont.) 200
6.11 Rasterization 201
6.12 Texture Objects 202
6.13 Texture Objects (cont.) 203
6.14 Texture Environment and Generation 204
6.15 Pixel Operations 205
6.16 Framebuffer Control 206
6.17 Pixels 207
6.18 Pixels (cont.) 208
6.19 Pixels (cont.) 209

Microsoft Corp. Exhibit 1009

APPENDIX T

6.20 Pixels (cont.) . . . 210
6.21 Pixels (cont.) . . . 211
6.22 Evaluators (**GetMap** takes a map name) . . . 212
6.23 Hints . . . 213
6.24 Implementation Dependent Values . . . 214
6.25 More Implementation Dependent Values . . . 215
6.26 Implementation Dependent Pixel Depths . . . 216
6.27 Miscellaneous . . . 217

F.1 Changes to State Tables . . . 252
F.2 Changes to State Tables (cont.) . . . 253
F.3 New State Introduced by Multitexture . . . 254
F.4 New Implementation-Dependent Values Introduced by Multitexture . . . 255

Microsoft Corp. Exhibit 1009

Chapter 1

Introduction

This document describes the OpenGL graphics system: what it is, how it acts, and what is required to implement it. We assume that the reader has at least a rudimentary understanding of computer graphics. This means familiarity with the essentials of computer graphics algorithms as well as familiarity with basic graphics hardware and associated terms.

1.1 Formatting of Optional Features

Starting with version 1.2 of OpenGL, some features in the specification are considered optional; an OpenGL implementation may or may not choose to provide them (see section 3.6.2).

Portions of the specification which are optional are so labelled where they are defined. Additionally, those portions are typeset in gray, and state table entries which are optional are typeset against a gray background.

1.2 What is the OpenGL Graphics System?

OpenGL (for "Open Graphics Library") is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

Most of OpenGL requires that the graphics hardware contain a framebuffer. Many OpenGL calls pertain to drawing objects such as points, lines, polygons, and bitmaps, but the way that some of this drawing occurs (such as when antialiasing or texturing is enabled) relies on the existence of a

Microsoft Corp. Exhibit 1009

framebuffer. Further, some of OpenGL is specifically concerned with framebuffer manipulation.

1.3 Programmer's View of OpenGL

To the programmer, OpenGL is a set of commands that allow the specification of geometric objects in two or three dimensions, together with commands that control how these objects are rendered into the framebuffer. For the most part, OpenGL provides an immediate-mode interface, meaning that specifying an object causes it to be drawn.

A typical program that uses OpenGL begins with calls to open a window into the framebuffer into which the program will draw. Then, calls are made to allocate a GL context and associate it with the window. Once a GL context is allocated, the programmer is free to issue OpenGL commands. Some calls are used to draw simple geometric objects (i.e. points, line segments, and polygons), while others affect the rendering of these primitives including how they are lit or colored and how they are mapped from the user's two- or three-dimensional model space to the two-dimensional screen. There are also calls to effect direct control of the framebuffer, such as reading and writing pixels.

1.4 Implementor's View of OpenGL

To the implementor, OpenGL is a set of commands that affect the operation of graphics hardware. If the hardware consists only of an addressable framebuffer, then OpenGL must be implemented almost entirely on the host CPU. More typically, the graphics hardware may comprise varying degrees of graphics acceleration, from a raster subsystem capable of rendering two-dimensional lines and polygons to sophisticated floating-point processors capable of transforming and computing on geometric data. The OpenGL implementor's task is to provide the CPU software interface while dividing the work for each OpenGL command between the CPU and the graphics hardware. This division must be tailored to the available graphics hardware to obtain optimum performance in carrying out OpenGL calls.

OpenGL maintains a considerable amount of state information. This state controls how objects are drawn into the framebuffer. Some of this state is directly available to the user: he or she can make calls to obtain its value. Some of it, however, is visible only by the effect it has on what is drawn. One of the main goals of this specification is to make OpenGL state

Microsoft Corp. Exhibit 1009

information explicit, to elucidate how it changes, and to indicate what its effects are.

1.5 Our View

We view OpenGL as a state machine that controls a set of specific drawing operations. This model should engender a specification that satisfies the needs of both programmers and implementors. It does not, however, necessarily provide a model for implementation. An implementation must produce results conforming to those produced by the specified methods, but there may be ways to carry out a particular computation that are more efficient than the one specified.

Microsoft Corp. Exhibit 1009

Chapter 2

OpenGL Operation

2.1 OpenGL Fundamentals

OpenGL (henceforth, the "GL") is concerned only with rendering into a framebuffer (and reading values stored in that framebuffer). There is no support for other peripherals sometimes associated with graphics hardware, such as mice and keyboards. Programmers must rely on other mechanisms to obtain user input.

The GL draws *primitives* subject to a number of selectable modes. Each primitive is a point, line segment, polygon, or pixel rectangle. Each mode may be changed independently; the setting of one does not affect the settings of others (although many modes may interact to determine what eventually ends up in the framebuffer). Modes are set, primitives specified, and other GL operations described by sending *commands* in the form of function or procedure calls.

Primitives are defined by a group of one or more *vertices*. A vertex defines a point, an endpoint of an edge, or a corner of a polygon where two edges meet. Data (consisting of positional coordinates, colors, normals, and texture coordinates) are associated with a vertex and each vertex is processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be *clipped* so that the indicated primitive fits within a specified region; in this case vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before the effects of a command are realized. This means, for example, that one primitive must be

Microsoft Corp. Exhibit 1009

drawn completely before any subsequent one can affect the framebuffer. It also means that queries and pixel read operations return state consistent with complete execution of all previously invoked GL commands. In general, the effects of a GL command on either GL modes or the framebuffer must be complete before any subsequent command can have any such effects.

In the GL, data binding occurs on call. This means that data passed to a command are interpreted when that command is received. Even if the command requires a pointer to data, those data are interpreted when the call is made, and any subsequent changes to the data have no effect on the GL (unless the same pointer is used in a subsequent command).

The GL provides direct control over the fundamental operations of 3D and 2D graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. It does not provide a means for describing or modeling complex geometric objects. Another way to describe this situation is to say that the GL provides mechanisms to describe how complex geometric objects are to be rendered rather than mechanisms to describe the complex objects themselves.

The model for interpretation of GL commands is client-server. That is, a program (the client) issues commands, and these commands are interpreted and processed by the GL (the server). The server may or may not operate on the same computer as the client. In this sense, the GL is "network-transparent." A server may maintain a number of GL *contexts*, each of which is an encapsulation of current GL state. A client may choose to *connect* to any one of these contexts. Issuing GL commands when the program is not *connected* to a *context* results in undefined behavior.

The effects of GL commands on the framebuffer are ultimately controlled by the window system that allocates framebuffer resources. It is the window system that determines which portions of the framebuffer the GL may access at any given time and that communicates to the GL how those portions are structured. Therefore, there are no GL commands to configure the framebuffer or initialize the GL. Similarly, display of framebuffer contents on a CRT monitor (including the transformation of individual framebuffer values by such techniques as gamma correction) is not addressed by the GL. Framebuffer configuration occurs outside of the GL in conjunction with the window system; the initialization of a GL context occurs when the window system allocates a window for GL rendering.

The GL is designed to be run on a range of graphics platforms with varying graphics capabilities and performance. To accommodate this variety, we specify ideal behavior instead of actual behavior for certain GL operations.

Microsoft Corp. Exhibit 1009

In cases where deviation from the ideal is allowed, we also specify the rules that an implementation must obey if it is to approximate the ideal behavior usefully. This allowed variation in GL behavior implies that two distinct GL implementations may not agree pixel for pixel when presented with the same input even when run on identical framebuffer configurations.

Finally, command names, constants, and types are prefixed in the GL (by **gl**, `GL_`, and `GL`, respectively in `C`) to reduce name clashes with other packages. The prefixes are omitted in this document for clarity.

2.1.1 Floating-Point Computation

The GL must perform a number of floating-point operations during the course of its operation. We do not specify how floating-point numbers are to be represented or how operations on them are to be performed. We require simply that numbers' floating-point parts contain enough bits and that their exponent fields are large enough so that individual results of floating-point operations are accurate to about 1 part in 10^5. The maximum representable magnitude of a floating-point number used to represent positional or normal coordinates must be at least 2^{32}; the maximum representable magnitude for colors or texture coordinates must be at least 2^{10}. The maximum representable magnitude for all other floating-point values must be at least 2^{32}. $x \cdot 0 = 0 \cdot x = 0$ for any non-infinite and non-NaN x. $1 \cdot x = x \cdot 1 = x$. $x + 0 = 0 + x = x$. $0^0 = 1$. (Occasionally further requirements will be specified.) Most single-precision floating-point formats meet these requirements.

Any representable floating-point value is legal as input to a GL command that requires floating-point data. The result of providing a value that is not a floating-point number to such a command is unspecified, but must not lead to GL interruption or termination. In IEEE arithmetic, for example, providing a negative zero or a denormalized number to a GL command yields predictable results, while providing a NaN or an infinity yields unspecified results.

Some calculations require division. In such cases (including implied divisions required by vector normalizations), a division by zero produces an unspecified result but must not lead to GL interruption or termination.

2.2 GL State

The GL maintains considerable state. This document enumerates each state variable and describes how each variable can be changed. For purposes of discussion, state variables are categorized somewhat arbitrarily by their

Microsoft Corp. Exhibit 1009

function. Although we describe the operations that the GL performs on the framebuffer, the framebuffer is not a part of GL state.

We distinguish two types of state. The first type of state, called GL *server state*, resides in the GL server. The majority of GL state falls into this category. The second type of state, called GL *client state*, resides in the GL client. Unless otherwise specified, all state referred to in this document is GL server state; GL client state is specifically identified. Each instance of a GL context implies one complete set of GL server state; each connection from a client to a server implies a set of both GL client state and GL server state.

While an implementation of the GL may be hardware dependent, this discussion is independent of the specific hardware on which a GL is implemented. We are therefore concerned with the state of graphics hardware only when it corresponds precisely to GL state.

2.3 GL Command Syntax

GL commands are functions or procedures. Various groups of commands perform the same operation but differ in how arguments are supplied to them. To conveniently accommodate this variation, we adopt a notation for describing commands and their arguments.

GL commands are formed from a *name* followed, depending on the particular command, by up to 4 characters. The first character indicates the number of values of the indicated type that must be presented to the command. The second character or character pair indicates the specific type of the arguments: 8-bit integer, 16-bit integer, 32-bit integer, single-precision floating-point, or double-precision floating-point. The final character, if present, is `v`, indicating that the command takes a pointer to an array (a vector) of values rather than a series of individual arguments. Two specific examples come from the **Vertex** command:

```
void Vertex3f( float x, float y, float z );
```

and

```
void Vertex2sv( short v[2] );
```

These examples show the ANSI `C` declarations for these commands. In general, a command declaration has the form[1]

[1]The declarations shown in this document apply to ANSI `C`. Languages such as `C++`

Microsoft Corp. Exhibit 1009

Letter	Corresponding GL Type
b	`byte`
s	`short`
i	`int`
f	`float`
d	`double`
ub	`ubyte`
us	`ushort`
ui	`uint`

Table 2.1: Correspondence of command suffix letters to GL argument types. Refer to Table 2.2 for definitions of the GL types.

rtype **Name**{ϵ**1234**}{ϵ **b s i f d ub us ui**}{ϵ**v**}
(*[args ,] T arg1 , ... , T argN [, args]*);

rtype is the return type of the function. The braces ({}) enclose a series of characters (or character pairs) of which one is selected. ϵ indicates no character. The arguments enclosed in brackets (*[args ,]* and *[, args]*) may or may not be present. The N arguments *arg1* through *argN* have type T, which corresponds to one of the type letters or letter pairs as indicated in Table 2.1 (if there are no letters, then the arguments' type is given explicitly). If the final character is not **v**, then N is given by the digit **1**, **2**, **3**, or **4** (if there is no digit, then the number of arguments is fixed). If the final character is **v**, then only *arg1* is present and it is an array of N values of the indicated type. Finally, we indicate an `unsigned` type by the shorthand of prepending a `u` to the beginning of the type name (so that, for instance, `unsigned char` is abbreviated `uchar`).

For example,

`void` **Normal3**{**fd**}(`T` *arg*);

indicates the two declarations

`void` **Normal3f**(`float` *arg1*, `float` *arg2*, `float` *arg3*);
`void` **Normal3d**(`double` *arg1*, `double` *arg2*, `double` *arg3*);

while

and Ada that allow passing of argument type information admit simpler declarations and fewer entry points.

Microsoft Corp. Exhibit 1009

```
void Normal3{fd}v( T arg );
```

means the two declarations

```
void Normal3fv( float arg[3] );
void Normal3dv( double arg[3] );
```

Arguments whose type is fixed (i.e. not indicated by a suffix on the command) are of one of 14 types (or pointers to one of these). These types are summarized in Table 2.2.

2.4 Basic GL Operation

Figure 2.1 shows a schematic diagram of the GL. Commands enter the GL on the left. Some commands specify geometric objects to be drawn while others control how the objects are handled by the various stages. Most commands may be accumulated in a *display list* for processing by the GL at a later time. Otherwise, commands are effectively sent through a processing pipeline.

The first stage provides an efficient means for approximating curve and surface geometry by evaluating polynomial functions of input values. The next stage operates on geometric primitives described by vertices: points, line segments, and polygons. In this stage vertices are transformed and lit, and primitives are clipped to a viewing volume in preparation for the next stage, rasterization. The rasterizer produces a series of framebuffer addresses and values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed to the next stage that performs operations on individual fragments before they finally alter the framebuffer. These operations include conditional updates into the framebuffer based on incoming and previously stored depth values (to effect depth buffering), blending of incoming fragment colors with stored colors, as well as masking and other logical operations on fragment values.

Finally, there is a way to bypass the vertex processing portion of the pipeline to send a block of fragments directly to the individual fragment operations, eventually causing a block of pixels to be written to the framebuffer; values may also be read back from the framebuffer or copied from one portion of the framebuffer to another. These transfers may include some type of decoding or encoding.

This ordering is meant only as a tool for describing the GL, not as a strict rule of how the GL is implemented, and we present it only as a means to

Microsoft Corp. Exhibit 1009

GL Type	Minimum Number of Bits	Description
`boolean`	1	Boolean
`byte`	8	signed 2's complement binary integer
`ubyte`	8	unsigned binary integer
`short`	16	signed 2's complement binary integer
`ushort`	16	unsigned binary integer
`int`	32	signed 2's complement binary integer
`uint`	32	unsigned binary integer
`sizei`	32	Non-negative binary integer size
`enum`	32	Enumerated binary integer value
`bitfield`	32	Bit field
`float`	32	Floating-point value
`clampf`	32	Floating-point value clamped to $[0, 1]$
`double`	64	Floating-point value
`clampd`	64	Floating-point value clamped to $[0, 1]$

Table 2.2: GL data types. GL types are not C types. Thus, for example, GL type `int` is referred to as `GLint` outside this document, and is not necessarily equivalent to the C type `int`. An implementation may use more bits than the number indicated in the table to represent a GL type. Correct interpretation of integer values outside the minimum range is not required, however.

Microsoft Corp. Exhibit 1009

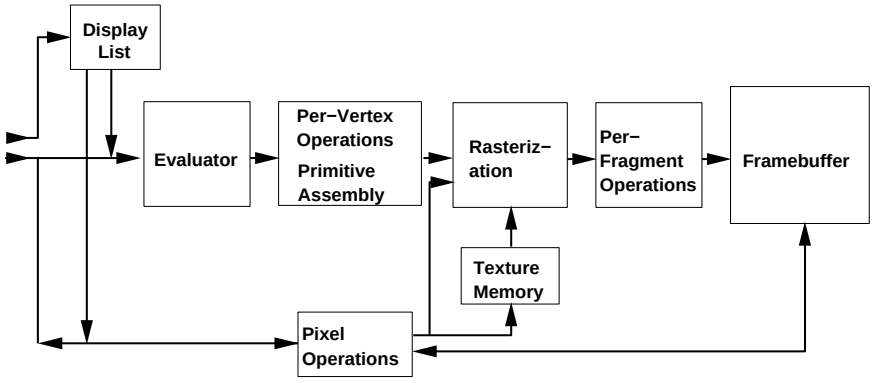


Figure 2.1. Block diagram of the GL.

organize the various operations of the GL. Objects such as curved surfaces, for instance, may be transformed before they are converted to polygons.

2.5 GL Errors

The GL detects only a subset of those conditions that could be considered errors. This is because in many cases error checking would adversely impact the performance of an error-free program.

The command

```
enum GetError( void );
```

is used to obtain error information. Each detectable error is assigned a numeric code. When an error is detected, a flag is set and the code is recorded. Further errors, if they occur, do not affect this recorded code. When **GetError** is called, the code is returned and the flag is cleared, so that a further error will again record its code. If a call to **GetError** returns `NO_ERROR`, then there has been no detectable error since the last call to **GetError** (or since the GL was initialized).

To allow for distributed implementations, there may be several flag-code pairs. In this case, after a call to **GetError** returns a value other than `NO_ERROR` each subsequent call returns the non-zero code of a distinct flag-code pair (in unspecified order), until all non-`NO_ERROR` codes have been

Microsoft Corp. Exhibit 1009

returned. When there are no more non-NO_ERROR error codes, all flags are reset. This scheme requires some positive number of pairs of a flag bit and an integer. The initial state of all flags is cleared and the initial value of all codes is NO_ERROR.

Table 2.3 summarizes GL errors. Currently, when an error flag is set, results of GL operation are undefined only if OUT_OF_MEMORY has occurred. In other cases, the command generating the error is ignored so that it has no effect on GL state or framebuffer contents. If the generating command returns a value, it returns zero. If the generating command modifies values through a pointer argument, no change is made to these values. These error semantics apply only to GL errors, not to system errors such as memory access errors. This behavior is the current behavior; the action of the GL in the presence of errors is subject to change.

Three error generation conditions are implicit in the description of every GL command. First, if a command that requires an enumerated value is passed a symbolic constant that is not one of those specified as allowable for that command, the error INVALID_ENUM results. This is the case even if the argument is a pointer to a symbolic constant if that value is not allowable for the given command. Second, if a negative number is provided where an argument of type sizei is specified, the error INVALID_VALUE results. Finally, if memory is exhausted as a side effect of the execution of a command, the error OUT_OF_MEMORY may be generated. Otherwise errors are generated only for conditions that are explicitly described in this specification.

2.6 Begin/End Paradigm

In the GL, most geometric objects are drawn by enclosing a series of coordinate sets that specify vertices and optionally normals, texture coordinates, and colors between **Begin**/**End** pairs. There are ten geometric objects that are drawn this way: points, line segments, line segment loops, separated line segments, polygons, triangle strips, triangle fans, separated triangles, quadrilateral strips, and separated quadrilaterals.

Each vertex is specified with two, three, or four coordinates. In addition, a *current normal*, *current texture coordinates*, and *current color* may be used in processing each vertex. Normals are used by the GL in lighting calculations; the current normal is a three-dimensional vector that may be set by sending three coordinates that specify it. Texture coordinates determine how a texture image is mapped onto a primitive.

Primary and secondary colors are associated with each vertex (see sec-

Microsoft Corp. Exhibit 1009

Error	Description	Offending command ignored?
INVALID_ENUM	enum argument out of range	Yes
INVALID_VALUE	Numeric argument out of range	Yes
INVALID_OPERATION	Operation illegal in current state	Yes
STACK_OVERFLOW	Command would cause a stack overflow	Yes
STACK_UNDERFLOW	Command would cause a stack underflow	Yes
OUT_OF_MEMORY	Not enough memory left to execute command	Unknown
TABLE_TOO_LARGE	The specified table is too large	Yes

Table 2.3: Summary of GL errors

tion 3.9). These *associated* colors are either based on the current color or produced by lighting, depending on whether or not lighting is enabled. Texture coordinates are similarly associated with each vertex. Figure 2.2 summarizes the association of auxiliary data with a transformed vertex to produce a *processed vertex*.

The current values are part of GL state. Vertices and normals are transformed, colors may be affected or replaced by lighting, and texture coordinates are transformed and possibly affected by a texture coordinate generation function. The processing indicated for each current value is applied for each vertex that is sent to the GL.

The methods by which vertices, normals, texture coordinates, and colors are sent to the GL, as well as how normals are transformed and how vertices are mapped to the two-dimensional screen, are discussed later.

Before colors have been assigned to a vertex, the state required by a vertex is the vertex's coordinates, the current normal, the current edge flag (see section 2.6.2), the current material properties (see section 2.13.2), and the current texture coordinates. Because color assignment is done vertex-by-vertex, a processed vertex comprises the vertex's coordinates, its edge flag, its assigned colors, and its texture coordinates.

Figure 2.3 shows the sequence of operations that builds a *primitive* (point, line segment, or polygon) from a sequence of vertices. After a primi-

Microsoft Corp. Exhibit 1009

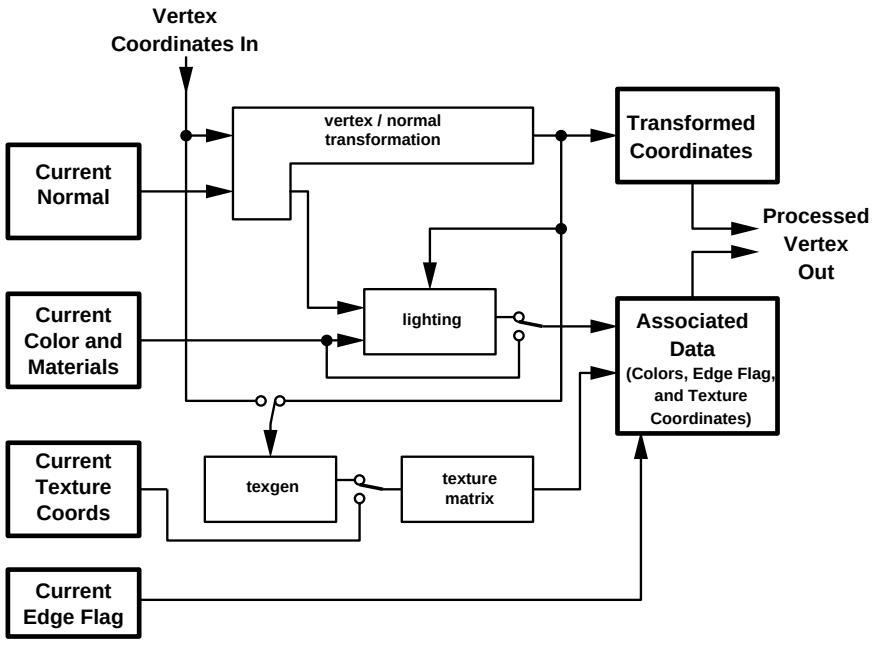


Figure 2.2. Association of current values with a vertex. The heavy lined boxes represent GL state.

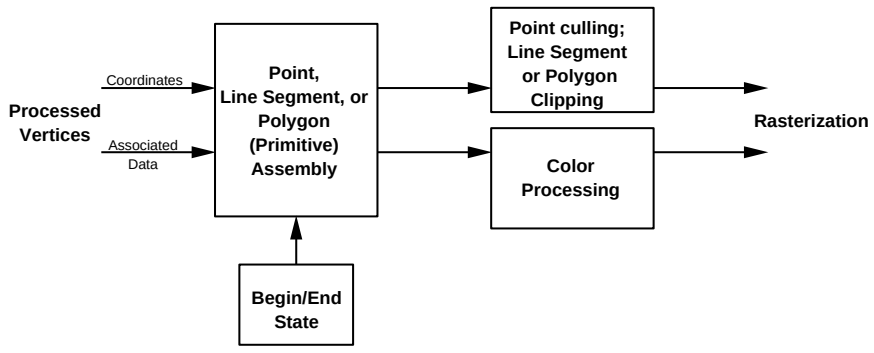


Figure 2.3. Primitive assembly and processing.

Microsoft Corp. Exhibit 1009

tive is formed, it is clipped to a viewing volume. This may alter the primitive by altering vertex coordinates, texture coordinates, and colors. In the case of a polygon primitive, clipping may insert new vertices into the primitive. The vertices defining a primitive to be rasterized have texture coordinates and colors associated with them.

2.6.1 Begin and End Objects

Begin and **End** require one state variable with eleven values: one value for each of the ten possible **Begin**/**End** objects, and one other value indicating that no **Begin**/**End** object is being processed. The two relevant commands are

```
void Begin( enum mode );
void End( void );
```

There is no limit on the number of vertices that may be specified between a **Begin** and an **End**.

Points. A series of individual points may be specified by calling **Begin** with an argument value of `POINTS`. No special state need be kept between **Begin** and **End** in this case, since each point is independent of previous and following points.

Line Strips. A series of one or more connected line segments is specified by enclosing a series of two or more endpoints within a **Begin**/**End** pair when **Begin** is called with `LINE_STRIP`. In this case, the first vertex specifies the first segment's start point while the second vertex specifies the first segment's endpoint and the second segment's start point. In general, the ith vertex (for $i > 1$) specifies the beginning of the ith segment and the end of the $i-1$st. The last vertex specifies the end of the last segment. If only one vertex is specified between the **Begin**/**End** pair, then no primitive is generated.

The required state consists of the processed vertex produced from the last vertex that was sent (so that a line segment can be generated from it to the current vertex), and a boolean flag indicating if the current vertex is the first vertex.

Line Loops. Line loops, specified with the `LINE_LOOP` argument value to **Begin**, are the same as line strips except that a final segment is added from the final specified vertex to the first vertex. The additional state consists of the processed first vertex.

Separate Lines. Individual line segments, each specified by a pair of vertices, are generated by surrounding vertex pairs with **Begin** and **End**

Microsoft Corp. Exhibit 1009

when the value of the argument to **Begin** is LINES. In this case, the first two vertices between a **Begin** and **End** pair define the first segment, with subsequent pairs of vertices each defining one more segment. If the number of specified vertices is odd, then the last one is ignored. The state required is the same as for lines but it is used differently: a vertex holding the first vertex of the current segment, and a boolean flag indicating whether the current vertex is odd or even (a segment start or end).

Polygons. A polygon is described by specifying its boundary as a series of line segments. When **Begin** is called with POLYGON, the bounding line segments are specified in the same way as line loops. Depending on the current state of the GL, a polygon may be rendered in one of several ways such as outlining its border or filling its interior. A polygon described with fewer than three vertices does not generate a primitive.

Only convex polygons are guaranteed to be drawn correctly by the GL. If a specified polygon is nonconvex when projected onto the window, then the rendered polygon need only lie within the convex hull of the projected vertices defining its boundary.

The state required to support polygons consists of at least two processed vertices (more than two are never required, although an implementation may use more); this is because a convex polygon can be rasterized as its vertices arrive, before all of them have been specified. The order of the vertices is significant in lighting and polygon rasterization (see sections 2.13.1 and 3.5.1).

Triangle strips. A triangle strip is a series of triangles connected along shared edges. A triangle strip is specified by giving a series of defining vertices between a **Begin**/**End** pair when **Begin** is called with TRIANGLE_STRIP. In this case, the first three vertices define the first triangle (and their order is significant, just as for polygons). Each subsequent vertex defines a new triangle using that point along with two vertices from the previous triangle. A **Begin**/**End** pair enclosing fewer than three vertices, when TRIANGLE_STRIP has been supplied to **Begin**, produces no primitive. See Figure 2.4.

The state required to support triangle strips consists of a flag indicating if the first triangle has been completed, two stored processed vertices, (called vertex A and vertex B), and a one bit pointer indicating which stored vertex will be replaced with the next vertex. After a **Begin**(TRIANGLE_STRIP), the pointer is initialized to point to vertex A. Each vertex sent between a **Begin**/**End** pair toggles the pointer. Therefore, the first vertex is stored as vertex A, the second stored as vertex B, the third stored as vertex A, and so on. Any vertex after the second one sent forms a triangle from vertex A, vertex B, and the current vertex (in that order).

Triangle fans. A triangle fan is the same as a triangle strip with one

Microsoft Corp. Exhibit 1009

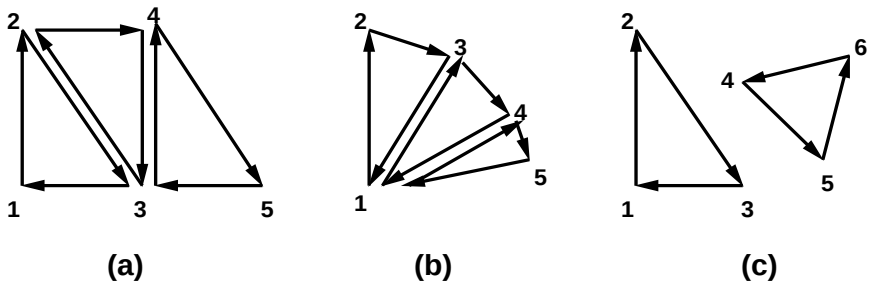


Figure 2.4. (a) A triangle strip. (b) A triangle fan. (c) Independent triangles. The numbers give the sequencing of the vertices between **Begin** and **End**. Note that in (a) and (b) triangle edge ordering is determined by the first triangle, while in (c) the order of each triangle's edges is independent of the other triangles.

exception: each vertex after the first always replaces vertex B of the two stored vertices. The vertices of a triangle fan are enclosed between **Begin** and **End** when the value of the argument to **Begin** is `TRIANGLE_FAN`.

Separate Triangles. Separate triangles are specified by placing vertices between **Begin** and **End** when the value of the argument to **Begin** is `TRIANGLES`. In this case, The $3i+1$st, $3i+2$nd, and $3i+3$rd vertices (in that order) determine a triangle for each $i = 0, 1, \ldots, n-1$, where there are $3n+k$ vertices between the **Begin** and **End**. k is either 0, 1, or 2; if k is not zero, the final k vertices are ignored. For each triangle, vertex A is vertex $3i$ and vertex B is vertex $3i+1$. Otherwise, separate triangles are the same as a triangle strip.

The rules given for polygons also apply to each triangle generated from a triangle strip, triangle fan or from separate triangles.

Quadrilateral (quad) strips. Quad strips generate a series of edge-sharing quadrilaterals from vertices appearing between **Begin** and **End**, when **Begin** is called with `QUAD_STRIP`. If the m vertices between the **Begin** and **End** are $v_1, \ldots, v_m$, where v_j is the jth specified vertex, then quad i has vertices (in order) v_{2i}, v_{2i+1}, v_{2i+3}, and v_{2i+2} with $i = 0, \ldots, \lfloor m/2 \rfloor$. The state required is thus three processed vertices, to store the last two vertices of the previous quad along with the third vertex (the first new vertex) of the current quad, a flag to indicate when the first quad has been completed, and a one-bit counter to count members of a vertex pair. See Figure 2.5.

Microsoft Corp. Exhibit 1009

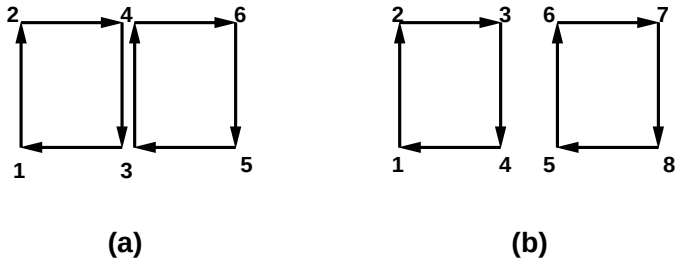


Figure 2.5. (a) A quad strip. (b) Independent quads. The numbers give the sequencing of the vertices between **Begin** and **End**.

A quad strip with fewer than four vertices generates no primitive. If the number of vertices specified for a quadrilateral strip between **Begin** and **End** is odd, the final vertex is ignored.

Separate Quadrilaterals Separate quads are just like quad strips except that each group of four vertices, the $4j+1$st, the $4j+2$nd, the $4j+3$rd, and the $4j+4$th, generate a single quad, for $j = 0, 1, \ldots, n-1$. The total number of vertices between **Begin** and **End** is $4n+k$, where $0 \leq k \leq 3$; if k is not zero, the final k vertices are ignored. Separate quads are generated by calling **Begin** with the argument value `QUADS`.

The rules given for polygons also apply to each quad generated in a quad strip or from separate quads.

2.6.2 Polygon Edges

Each edge of each primitive generated from a polygon, triangle strip, triangle fan, separate triangle set, quadrilateral strip, or separate quadrilateral set, is flagged as either *boundary* or *non-boundary*. These classifications are used during polygon rasterization; some modes affect the interpretation of polygon boundary edges (see section 3.5.4). By default, all edges are boundary edges, but the flagging of polygons, separate triangles, or separate quadrilaterals may be altered by calling

```
void EdgeFlag( boolean flag );
void EdgeFlagv( boolean *flag );
```

to change the value of a flag bit. If *flag* is zero, then the flag bit is set to `FALSE`; if *flag* is non-zero, then the flag bit is set to `TRUE`.

Microsoft Corp. Exhibit 1009

When **Begin** is supplied with one of the argument values `POLYGON`, `TRIANGLES`, or `QUADS`, each vertex specified within a **Begin** and **End** pair begins an edge. If the edge flag bit is `TRUE`, then each specified vertex begins an edge that is flagged as boundary. If the bit is `FALSE`, then induced edges are flagged as non-boundary.

The state required for edge flagging consists of one current flag bit. Initially, the bit is `TRUE`. In addition, each processed vertex of an assembled polygonal primitive must be augmented with a bit indicating whether or not the edge beginning on that vertex is boundary or non-boundary.

2.6.3 GL Commands within Begin/End

The only GL commands that are allowed within any **Begin**/**End** pairs are the commands for specifying vertex coordinates, vertex color, normal coordinates, and texture coordinates (**Vertex**, **Color**, **Index**, **Normal**, **TexCoord**), the **ArrayElement** command (see section 2.8), the **EvalCoord** and **EvalPoint** commands (see section 5.1), commands for specifying lighting material parameters (**Material** commands; see section 2.13.2), display list invocation commands (**CallList** and **CallLists**; see section 5.4), and the **EdgeFlag** command. Executing any other GL command between the execution of **Begin** and the corresponding execution of **End** results in the error `INVALID_OPERATION`. Executing **Begin** after **Begin** has already been executed but before an **End** is executed generates the `INVALID_OPERATION` error, as does executing **End** without a previous corresponding **Begin**.

Execution of the commands **EnableClientState**, **DisableClientState**, **PushClientAttrib**, **PopClientAttrib**, **EdgeFlagPointer**, **TexCoordPointer**, **ColorPointer**, **IndexPointer**, **NormalPointer**, **VertexPointer**, **InterleavedArrays**, and **PixelStore**, is not allowed within any **Begin**/**End** pair, but an error may or may not be generated if such execution occurs. If an error is not generated, GL operation is undefined. (These commands are described in sections 2.8, 3.6.1, and Chapter 6.)

2.7 Vertex Specification

Vertices are specified by giving their coordinates in two, three, or four dimensions. This is done using one of several versions of the **Vertex** command:

```
void Vertex{234}{sifd}( T coords );
void Vertex{234}{sifd}v( T coords );
```


Microsoft Corp. Exhibit 1009

A call to any **Vertex** command specifies four coordinates: x, y, z, and w. The x coordinate is the first coordinate, y is second, z is third, and w is fourth. A call to **Vertex2** sets the x and y coordinates; the z coordinate is implicitly set to zero and the w coordinate to one. **Vertex3** sets x, y, and z to the provided values and w to one. **Vertex4** sets all four coordinates, allowing the specification of an arbitrary point in projective three-space. Invoking a **Vertex** command outside of a **Begin**/**End** pair results in undefined behavior.

Current values are used in associating auxiliary data with a vertex as described in section 2.6. A current value may be changed at any time by issuing an appropriate command. The commands

```
void TexCoord{1234}{sifd}( T coords );
void TexCoord{1234}{sifd}v( T coords );
```

specify the current homogeneous texture coordinates, named s, t, r, and q. The **TexCoord1** family of commands set the s coordinate to the provided single argument while setting t and r to 0 and q to 1. Similarly, **TexCoord2** sets s and t to the specified values, r to 0 and q to 1; **TexCoord3** sets s, t, and r, with q set to 1, and **TexCoord4** sets all four texture coordinates.

The current normal is set using

```
void Normal3{bsifd}( T coords );
void Normal3{bsifd}v( T coords );
```

Byte, short, or integer values passed to **Normal** are converted to floating-point values as indicated for the corresponding (signed) type in Table 2.6.

Finally, there are several ways to set the current color. The GL stores both a current single-valued *color index*, and a current four-valued RGBA color. One or the other of these is significant depending as the GL is in *color index mode* or *RGBA mode*. The mode selection is made when the GL is initialized.

The command to set RGBA colors is

```
void Color{34}{bsifd ubusui}( T components );
void Color{34}{bsifd ubusui}v( T components );
```

The **Color** command has two major variants: **Color3** and **Color4**. The four value versions set all four values. The three value versions set R, G, and B to the provided values; A is set to 1.0. (The conversion of integer color components (R, G, B, and A) to floating-point values is discussed in section 2.13.)

Microsoft Corp. Exhibit 1009

Versions of the **Color** command that take floating-point values accept values nominally between 0.0 and 1.0. 0.0 corresponds to the minimum while 1.0 corresponds to the maximum (machine dependent) value that a component may take on in the framebuffer (see section 2.13 on colors and coloring). Values outside $[0, 1]$ are not clamped.

The command

```
void Index{sifd ub}( T index );
void Index{sifd ub}v( T index );
```

updates the current (single-valued) color index. It takes one argument, the value to which the current color index should be set. Values outside the (machine-dependent) representable range of color indices are not clamped.

The state required to support vertex specification consists of four floating-point numbers to store the current texture coordinates s, t, r, and q, three floating-point numbers to store the three coordinates of the current normal, four floating-point values to store the current RGBA color, and one floating-point value to store the current color index. There is no notion of a current vertex, so no state is devoted to vertex coordinates. The initial values of s, t, and r of the current texture coordinates are zero; the initial value of q is one. The initial current normal has coordinates $(0, 0, 1)$. The initial RGBA color is $(\mathrm{R}, \mathrm{G}, \mathrm{B}, \mathrm{A}) = (1, 1, 1, 1)$. The initial color index is 1.

2.8 Vertex Arrays

The vertex specification commands described in section 2.7 accept data in almost any format, but their use requires many command executions to specify even simple geometry. Vertex data may also be placed into arrays that are stored in the client's address space. Blocks of data in these arrays may then be used to specify multiple geometric primitives through the execution of a single GL command. The client may specify up to six arrays: one each to store edge flags, texture coordinates, colors, color indices, normals, and vertices. The commands

```
void EdgeFlagPointer( sizei stride, void *pointer );

void TexCoordPointer( int size, enum type, sizei stride,
    void *pointer );

void ColorPointer( int size, enum type, sizei stride,
    void *pointer );
```


Microsoft Corp. Exhibit 1009

Command	Sizes	Types
VertexPointer	2,3,4	`short`, `int`, `float`, `double`
NormalPointer	3	`byte`, `short`, `int`, `float`, `double`
ColorPointer	3,4	`byte`, `ubyte`, `short`, `ushort`, `int`, `uint`, `float`, `double`
IndexPointer	1	`ubyte`, `short`, `int`, `float`, `double`
TexCoordPointer	1,2,3,4	`short`, `int`, `float`, `double`
EdgeFlagPointer	1	`boolean`

Table 2.4: Vertex array sizes (values per vertex) and data types.

```
void IndexPointer( enum type, sizei stride,
    void *pointer );

void NormalPointer( enum type, sizei stride,
    void *pointer );

void VertexPointer( int size, enum type, sizei stride,
    void *pointer );
```

describe the locations and organizations of these arrays. For each command, *type* specifies the data type of the values stored in the array. Because edge flags are always type `boolean`, **EdgeFlagPointer** has no *type* argument. *size*, when present, indicates the number of values per vertex that are stored in the array. Because normals are always specified with three values, **NormalPointer** has no *size* argument. Likewise, because color indices and edge flags are always specified with a single value, **IndexPointer** and **EdgeFlagPointer** also have no *size* argument. Table 2.4 indicates the allowable values for *size* and *type* (when present). For *type* the values `BYTE`, `SHORT`, `INT`, `FLOAT`, and `DOUBLE` indicate types `byte`, `short`, `int`, `float`, and `double`, respectively; and the values `UNSIGNED_BYTE`, `UNSIGNED_SHORT`, and `UNSIGNED_INT` indicate types `ubyte`, `ushort`, and `uint`, respectively. The error `INVALID_VALUE` is generated if *size* is specified with a value other than that indicated in the table.

The one, two, three, or four values in an array that correspond to a single vertex comprise an array *element*. The values within each array element are stored sequentially in memory. If *stride* is specified as zero, then array elements are stored sequentially as well. Otherwise pointers to the ith and $(i+1)$st elements of an array differ by *stride* basic machine units (typically

Microsoft Corp. Exhibit 1009

unsigned bytes), the pointer to the $(i+1)$st element being greater. For each command, *pointer* specifies the location in memory of the first value of the first element of the array being specified.

An individual array is enabled or disabled by calling one of

```
void EnableClientState( enum array );
void DisableClientState( enum array );
```

with *array* set to `EDGE_FLAG_ARRAY`, `TEXTURE_COORD_ARRAY`, `COLOR_ARRAY`, `INDEX_ARRAY`, `NORMAL_ARRAY`, or `VERTEX_ARRAY`, for the edge flag, texture coordinate, color, color index, normal, or vertex array, respectively.

The ith element of every enabled array is transferred to the GL by calling

```
void ArrayElement( int i );
```

For each enabled array, it is as though the corresponding command from section 2.7 or section 2.6.2 were called with a pointer to element i. For the vertex array, the corresponding command is **Vertex**[*size*][*type*]**v**, where *size* is one of [2,3,4], and *type* is one of [s,i,f,d], corresponding to array types `short`, `int`, `float`, and `double` respectively. The corresponding commands for the edge flag, texture coordinate, color, color index, and normal arrays are **EdgeFlagv**, **TexCoord**[*size*][*type*]**v**, **Color**[*size*][*type*]**v**, **Index**[*type*]**v**, and **Normal**[*type*]**v**, respectively. If the vertex array is enabled, it is as though **Vertex**[*size*][*type*]**v** is executed last, after the executions of the other corresponding commands.

Changes made to array data between the execution of **Begin** and the corresponding execution of **End** may affect calls to **ArrayElement** that are made within the same **Begin**/**End** period in non-sequential ways. That is, a call to **ArrayElement** that precedes a change to array data may access the changed data, and a call that follows a change to array data may access original data.

The command

```
void DrawArrays( enum mode, int first, sizei count );
```

constructs a sequence of geometric primitives using elements $first$ through $first+count-1$ of each enabled array. *mode* specifies what kind of primitives are constructed; it accepts the same token values as the *mode* parameter of the **Begin** command. The effect of

DrawArrays $(mode, first, count)$;

Microsoft Corp. Exhibit 1009

APPENDIX T

is the same as the effect of the command sequence

```
if (mode or count is invalid )
   generate appropriate error
else {
   int i;
   Begin(mode);
   for (i=0; i < count ; i++)
      ArrayElement(first+ i);
   End();
}
```

with one exception: the current edge flag, texture coordinates, color, color index, and normal coordinates are each indeterminate after the execution of **DrawArrays**, if the corresponding array is enabled. Current values corresponding to disabled arrays are not modified by the execution of **DrawArrays**.

The command

```
void DrawElements( enum mode, sizei count, enum type,
   void *indices );
```

constructs a sequence of geometric primitives using the *count* elements whose indices are stored in *indices*. *type* must be one of `UNSIGNED_BYTE`, `UNSIGNED_SHORT`, or `UNSIGNED_INT`, indicating that the values in *indices* are indices of GL type `ubyte`, `ushort`, or `uint` respectively. *mode* specifies what kind of primitives are constructed; it accepts the same token values as the *mode* parameter of the **Begin** command. The effect of

DrawElements (*mode*, *count*, *type*, *indices*);

is the same as the effect of the command sequence

```
if (mode, count, or type is invalid )
   generate appropriate error
else {
   int i;
   Begin(mode);
   for (i=0; i < count ; i++)
      ArrayElement(indices[i]);
   End();
}
```


Microsoft Corp. Exhibit 1009

with one exception: the current edge flag, texture coordinates, color, color index, and normal coordinates are each indeterminate after the execution of **DrawElements**, if the corresponding array is enabled. Current values corresponding to disabled arrays are not modified by the execution of **DrawElements**.

The command

```
void DrawRangeElements( enum mode, uint start,
    uint end, sizei count, enum type, void *indices );
```

is a restricted form of **DrawElements**. *mode*, *count*, *type*, and *indices* match the corresponding arguments to **DrawElements**, with the additional constraint that all values in the array *indices* must lie between *start* and *end* inclusive.

Implementations denote recommended maximum amounts of vertex and index data, which may be queried by calling **GetIntegerv** with the symbolic constants `MAX_ELEMENTS_VERTICES` and `MAX_ELEMENTS_INDICES`. If $end-start+1$ is greater than the value of `MAX_ELEMENTS_VERTICES`, or if *count* is greater than the value of `MAX_ELEMENTS_INDICES`, then the call may operate at reduced performance. There is no requirement that all vertices in the range $[start, end]$ be referenced. However, the implementation may partially process unused vertices, reducing performance from what could be achieved with an optimal index set.

The error `INVALID_VALUE` is generated if $end < start$. Invalid *mode*, *count*, or *type* parameters generate the same errors as would the corresponding call to **DrawElements**. It is an error for indices to lie outside the range $[start, end]$, but implementations may not check for this. Such indices will cause implementation-dependent behavior.

The command

```
void InterleavedArrays( enum format, sizei stride,
    void *pointer );
```

efficiently initializes the six arrays and their enables to one of 14 configurations. *format* must be one of 14 symbolic constants: `V2F`, `V3F`, `C4UB_V2F`, `C4UB_V3F`, `C3F_V3F`, `N3F_V3F`, `C4F_N3F_V3F`, `T2F_V3F`, `T4F_V4F`, `T2F_C4UB_V3F`, `T2F_C3F_V3F`, `T2F_N3F_V3F`, `T2F_C4F_N3F_V3F`, or `T4F_C4F_N3F_V4F`.

The effect of

InterleavedArrays(*format*, *stride*, *pointer*);

is the same as the effect of the command sequence

Microsoft Corp. Exhibit 1009

format	e_t	e_c	e_n	s_t	s_c	s_v	t_c
V2F	*False*	*False*	*False*			2	
V3F	*False*	*False*	*False*			3	
C4UB_V2F	*False*	*True*	*False*		4	2	UNSIGNED_BYTE
C4UB_V3F	*False*	*True*	*False*		4	3	UNSIGNED_BYTE
C3F_V3F	*False*	*True*	*False*		3	3	FLOAT
N3F_V3F	*False*	*False*	*True*			3	
C4F_N3F_V3F	*False*	*True*	*True*		4	3	FLOAT
T2F_V3F	*True*	*False*	*False*	2		3	
T4F_V4F	*True*	*False*	*False*	4		4	
T2F_C4UB_V3F	*True*	*True*	*False*	2	4	3	UNSIGNED_BYTE
T2F_C3F_V3F	*True*	*True*	*False*	2	3	3	FLOAT
T2F_N3F_V3F	*True*	*False*	*True*	2		3	
T2F_C4F_N3F_V3F	*True*	*True*	*True*	2	4	3	FLOAT
T4F_C4F_N3F_V4F	*True*	*True*	*True*	4	4	4	FLOAT

format	p_c	p_n	p_v	s
V2F			0	$2f$
V3F			0	$3f$
C4UB_V2F	0		c	$c+2f$
C4UB_V3F	0		c	$c+3f$
C3F_V3F	0		$3f$	$6f$
N3F_V3F		0	$3f$	$6f$
C4F_N3F_V3F	0	$4f$	$7f$	$10f$
T2F_V3F			$2f$	$5f$
T4F_V4F			$4f$	$8f$
T2F_C4UB_V3F	$2f$		$c+2f$	$c+5f$
T2F_C3F_V3F	$2f$		$5f$	$8f$
T2F_N3F_V3F		$2f$	$5f$	$8f$
T2F_C4F_N3F_V3F	$2f$	$6f$	$9f$	$12f$
T4F_C4F_N3F_V4F	$4f$	$8f$	$11f$	$15f$

Table 2.5: Variables that direct the execution of **InterleavedArrays**. f is `sizeof(FLOAT)`. c is 4 times `sizeof(UNSIGNED_BYTE)`, rounded up to the nearest multiple of f. All pointer arithmetic is performed in units of `sizeof(UNSIGNED_BYTE)`.

Microsoft Corp. Exhibit 1009

```
if (format or stride is invalid)
   generate appropriate error
else {
   int str;
   set e_t, e_c, e_n, s_t, s_c, s_v, t_c, p_c, p_n, p_v, and s as a function
      of Table 2.5 and the value of format.
   str = stride;
   if (str is zero)
      str = s;
   DisableClientState(EDGE_FLAG_ARRAY);
   DisableClientState(INDEX_ARRAY);
   if (e_t) {
      EnableClientState(TEXTURE_COORD_ARRAY);
      TexCoordPointer(s_t, FLOAT, str, pointer);
   } else {
      DisableClientState(TEXTURE_COORD_ARRAY);
   }
   if (e_c) {
      EnableClientState(COLOR_ARRAY);
      ColorPointer(s_c, t_c, str, pointer + p_c);
   } else {
      DisableClientState(COLOR_ARRAY);
   }
   if (e_n) {
      EnableClientState(NORMAL_ARRAY);
      NormalPointer(FLOAT, str, pointer + p_n);
   } else {
      DisableClientState(NORMAL_ARRAY);
   }
   EnableClientState(VERTEX_ARRAY);
   VertexPointer(s_v, FLOAT, str, pointer + p_v);
}
```

The client state required to implement vertex arrays consists of six boolean values, six memory pointers, six integer stride values, five symbolic constants representing array types, and three integers representing values per element. In the initial state the boolean values are each disabled, the memory pointers are each null, the strides are each zero, the array types are each FLOAT, and the integers representing values per element are each four.

Microsoft Corp. Exhibit 1009

2.9 Rectangles

There is a set of GL commands to support efficient specification of rectangles as two corner vertices.

```
void Rect{sifd}( T x1, T y1, T x2, T y2 );
void Rect{sifd}v( T v1[2], T v2[2] );
```

Each command takes either four arguments organized as two consecutive pairs of (x, y) coordinates, or two pointers to arrays each of which contains an x value followed by a y value. The effect of the **Rect** command

```
Rect (x1, y1, x2, y2);
```

is exactly the same as the following sequence of commands:

```
Begin(POLYGON);
   Vertex2(x1, y1);
   Vertex2(x2, y1);
   Vertex2(x2, y2);
   Vertex2(x1, y2);
End();
```

The appropriate **Vertex2** command would be invoked depending on which of the **Rect** commands is issued.

2.10 Coordinate Transformations

Vertices, normals, and texture coordinates are transformed before their coordinates are used to produce an image in the framebuffer. We begin with a description of how vertex coordinates are transformed and how this transformation is controlled.

Figure 2.6 diagrams the sequence of transformations that are applied to vertices. The vertex coordinates that are presented to the GL are termed *object coordinates*. The *model-view* matrix is applied to these coordinates to yield *eye* coordinates. Then another matrix, called the *projection* matrix, is applied to eye coordinates to yield *clip* coordinates. A perspective division is carried out on clip coordinates to yield *normalized device* coordinates. A final *viewport* transformation is applied to convert these coordinates into *window coordinates*.

Microsoft Corp. Exhibit 1009

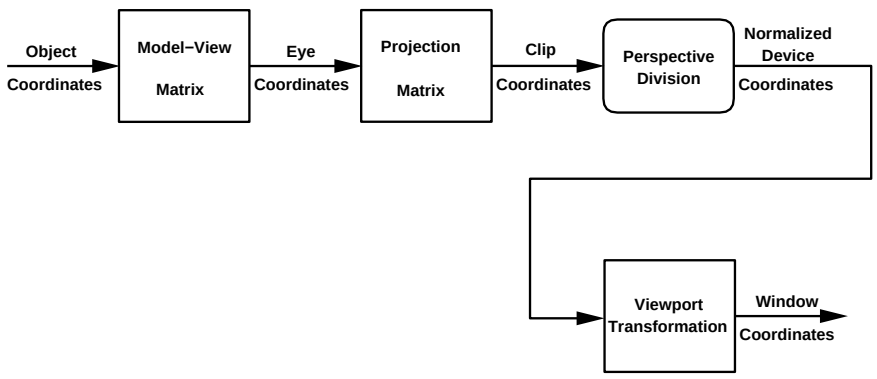


Figure 2.6. Vertex transformation sequence.

Object coordinates, eye coordinates, and clip coordinates are four-dimensional, consisting of x, y, z, and w coordinates (in that order). The model-view and perspective matrices are thus 4×4.

If a vertex in object coordinates is given by $\begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}$ and the model-view matrix is M, then the vertex's eye coordinates are found as

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = M \begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}.$$

Similarly, if P is the projection matrix, then the vertex's clip coordinates are

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = P \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}.$$

The vertex's normalized device coordinates are then

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{pmatrix}.$$

Microsoft Corp. Exhibit 1009

2.10.1 Controlling the Viewport

The viewport transformation is determined by the viewport's width and height in pixels, p_x and p_y, respectively, and its center (o_x, o_y) (also in pixels). The vertex's window coordinates, $\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}$, are given by

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} (p_x/2)x_d + o_x \\ (p_y/2)y_d + o_y \\ [(f-n)/2]z_d + (n+f)/2 \end{pmatrix}.$$

The factor and offset applied to z_d encoded by n and f are set using

`void` **DepthRange**(`clampd` n, `clampd` f);

Each of n and f are clamped to lie within $[0, 1]$, as are all arguments of type `clampd` or `clampf`. z_w is taken to be represented in fixed-point with at least as many bits as there are in the depth buffer of the framebuffer. We assume that the fixed-point representation used represents each value $k/(2^m - 1)$, where $k \in \{0, 1, \ldots, 2^m - 1\}$, as k (e.g. 1.0 is represented in binary as a string of all ones).

Viewport transformation parameters are specified using

`void` **Viewport**(`int` x, `int` y, `sizei` w, `sizei` h);

where x and y give the x and y window coordinates of the viewport's lower-left corner and w and h give the viewport's width and height, respectively. The viewport parameters shown in the above equations are found from these values as $o_x = x + w/2$ and $o_y = y + h/2$; $p_x = w$, $p_y = h$.

Viewport width and height are clamped to implementation-dependent maximums when specified. The maximum width and height may be found by issuing an appropriate **Get** command (see Chapter 6). The maximum viewport dimensions must be greater than or equal to the visible dimensions of the display being rendered to. `INVALID_VALUE` is generated if either w or h is negative.

The state required to implement the viewport transformation is 6 integers. In the initial state, w and h are set to the width and height, respectively, of the window into which the GL is to do its rendering. o_x and o_y are set to $w/2$ and $h/2$, respectively. n and f are set to 0.0 and 1.0, respectively.

Microsoft Corp. Exhibit 1009

2.10.2 Matrices

The projection matrix and model-view matrix are set and modified with a variety of commands. The affected matrix is determined by the current matrix mode. The current matrix mode is set with

`void` **MatrixMode**(`enum` *mode*);

which takes one of the pre-defined constants `TEXTURE`, `MODELVIEW`, `COLOR`, or `PROJECTION` as the argument value. `TEXTURE` is described later in section 2.10.2, and `COLOR`is described in section 3.6.3. If the current matrix mode is `MODELVIEW`, then matrix operations apply to the model-view matrix; if `PROJECTION`, then they apply to the projection matrix.

The two basic commands for affecting the current matrix are

`void` **LoadMatrix{fd}**(`T` *m[16]*);
`void` **MultMatrix{fd}**(`T` *m[16]*);

LoadMatrix takes a pointer to a 4×4 matrix stored in column-major order as 16 consecutive floating-point values, i.e. as

$$\begin{pmatrix} a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \\ a_4 & a_8 & a_{12} & a_{16} \end{pmatrix}.$$

(This differs from the standard row-major `C` ordering for matrix elements. If the standard ordering is used, all of the subsequent transformation equations are transposed, and the columns representing vectors become rows.)

The specified matrix replaces the current matrix with the one pointed to. **MultMatrix** takes the same type argument as **LoadMatrix**, but multiplies the current matrix by the one pointed to and replaces the current matrix with the product. If C is the current matrix and M is the matrix pointed to by **MultMatrix**'s argument, then the resulting current matrix, C', is

$$C' = C \cdot M.$$

The command

`void` **LoadIdentity**(`void`);

Microsoft Corp. Exhibit 1009

APPENDIX T

effectively calls **LoadMatrix** with the identity matrix:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

There are a variety of other commands that manipulate matrices. **Rotate**, **Translate**, **Scale**, **Frustum**, and **Ortho** manipulate the current matrix. Each computes a matrix and then invokes **MultMatrix** with this matrix. In the case of

`void` **Rotate{fd}**(`T` θ, `T` x, `T` y, `T` z);

θ gives an angle of rotation in degrees; the coordinates of a vector $\mathbf{v}$ are given by $\mathbf{v} = (x\ y\ z)^T$. The computed matrix is a counter-clockwise rotation about the line through the origin with the specified axis when that axis is pointing up (i.e. the right-hand rule determines the sense of the rotation angle). The matrix is thus

$$\begin{pmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Let $\mathbf{u} = \mathbf{v}/||\mathbf{v}|| = (\,x' \quad y' \quad z'\,)^T$. If

$$S = \begin{pmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{pmatrix}$$

then

$$R = \mathbf{u}\mathbf{u}^T + \cos\theta(I - \mathbf{u}\mathbf{u}^T) + \sin\theta S.$$

The arguments to

`void` **Translate{fd}**(`T` x, `T` y, `T` z);

give the coordinates of a translation vector as $(x\ y\ z)^T$. The resulting matrix is a translation by the specified vector:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Microsoft Corp. Exhibit 1009

`void` **Scale{fd}**(`T` x, `T` y, `T` z);

produces a general scaling along the x-, y-, and z- axes. The corresponding matrix is

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

For

`void` **Frustum**(`double` l, `double` r, `double` b, `double` t, `double` n, `double` f);

the coordinates $(l\ b\ -n)^T$ and $(r\ t\ -n)^T$ specify the points on the near clipping plane that are mapped to the lower-left and upper-right corners of the window, respectively (assuming that the eye is located at $(0\ 0\ 0)^T$). f gives the distance from the eye to the far clipping plane. If either n or f is less than or equal to zero, l is equal to r, b is equal to t, or n is equal to f, the error `INVALID_VALUE` results. The corresponding matrix is

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

`void` **Ortho**(`double` l, `double` r, `double` b, `double` t, `double` n, `double` f);

describes a matrix that produces parallel projection. $(l\ b\ -n)^T$ and $(r\ t\ -n)^T$ specify the points on the near clipping plane that are mapped to the lower-left and upper-right corners of the window, respectively. f gives the distance from the eye to the far clipping plane. If l is equal to r, b is equal to t, or n is equal to f, the error `INVALID_VALUE` results. The corresponding matrix is

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

There is another 4×4 matrix that is applied to texture coordinates. This matrix is applied as

Microsoft Corp. Exhibit 1009

$$\begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix} \begin{pmatrix} s \\ t \\ r \\ q \end{pmatrix},$$

where the left matrix is the current texture matrix. The matrix is applied to the coordinates resulting from texture coordinate generation (which may simply be the current texture coordinates), and the resulting transformed coordinates become the texture coordinates associated with a vertex. Setting the matrix mode to `TEXTURE` causes the already described matrix operations to apply to the texture matrix.

There is a stack of matrices for each of the matrix modes. For `MODELVIEW` mode, the stack depth is at least 32 (that is, there is a stack of at least 32 model-view matrices). For the other modes, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

`void` **PushMatrix**(`void`);

pushes the stack down by one, duplicating the current matrix in both the top of the stack and the entry below it.

`void` **PopMatrix**(`void`);

pops the top entry off of the stack, replacing the current matrix with the matrix that was the second entry in the stack. The pushing or popping takes place on the stack corresponding to the current matrix mode. Popping a matrix off a stack with only one entry generates the error `STACK_UNDERFLOW`; pushing a matrix onto a full stack generates `STACK_OVERFLOW`.

The state required to implement transformations consists of a four-valued integer indicating the current matrix mode, a stack of at least two 4×4 matrices for each of `COLOR`, `PROJECTION`, and `TEXTURE` with associated stack pointers, and a stack of at least 32 4×4 matrices with an associated stack pointer for `MODELVIEW`. Initially, there is only one matrix on each stack, and all matrices are set to the identity. The initial matrix mode is `MODELVIEW`.

2.10.3 Normal Transformation

Finally, we consider how the model-view matrix and transformation state affect normals. Before use in lighting, normals are transformed to eye coordinates by a matrix derived from the model-view matrix. Rescaling and normalization operations are performed on the transformed normals to make

Microsoft Corp. Exhibit 1009

them unit length prior to use in lighting. Rescaling and normalization are controlled by

`void` **Enable**(`enum` *target*);

and

`void` **Disable**(`enum` *target*);

with *target* equal to `RESCALE_NORMAL` or `NORMALIZE`. This requires two bits of state. The initial state is for normals not to be rescaled or normalized.

If the model-view matrix is M, then the normal is transformed to eye coordinates by:

$$\begin{pmatrix} n_x' & n_y' & n_z' & q' \end{pmatrix} = \begin{pmatrix} n_x & n_y & n_z & q \end{pmatrix} \cdot M^{-1}$$

where, if $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$ are the associated vertex coordinates, then

$$q = \begin{cases} 0, & w = 0, \\ \dfrac{-\begin{pmatrix} n_x & n_y & n_z \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}}{w}, & w \neq 0 \end{cases} \tag{2.1}$$

Implementations may choose instead to transform $\begin{pmatrix} n_x & n_y & n_z \end{pmatrix}$ to eye coordinates using

$$\begin{pmatrix} n_x' & n_y' & n_z' \end{pmatrix} = \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} \cdot {M_u}^{-1}$$

where M_u is the upper leftmost 3x3 matrix taken from M.

Rescale multiplies the transformed normals by a scale factor

$$\begin{pmatrix} n_x'' & n_y'' & n_z'' \end{pmatrix} = f \begin{pmatrix} n_x' & n_y' & n_z' \end{pmatrix}$$

If rescaling is disabled, then $f = 1$. If rescaling is enabled, then f is computed as (m_{ij} denotes the matrix element in row i and column j of M^{-1}, numbering the topmost row of the matrix as row 1 and the leftmost column as column 1)

$$f = \frac{1}{\sqrt{{m_{31}}^2 + {m_{32}}^2 + {m_{33}}^2}}$$

Microsoft Corp. Exhibit 1009

Note that if the normals sent to GL were unit length and the model-view matrix uniformly scales space, then rescale makes the transformed normals unit length.

Alternatively, an implementation may chose f as

$$f = \frac{1}{\sqrt{{n_x'}^2 + {n_y'}^2 + {n_z'}^2}}$$

recomputing f for each normal. This makes all non-zero length normals unit length regardless of their input length and the nature of the model-view matrix.

After rescaling, the final transformed normal used in lighting, n_f, is computed as

$$n_f = m \begin{pmatrix} {n_x}'' & {n_y}'' & {n_z}'' \end{pmatrix}$$

If normalization is disabled, then $m = 1$. Otherwise

$$m = \frac{1}{\sqrt{{n_x''}^2 + {n_y''}^2 + {n_z''}^2}}$$

Because we specify neither the floating-point format nor the means for matrix inversion, we cannot specify behavior in the case of a poorly-conditioned (nearly singular) model-view matrix M. In case of an exactly singular matrix, the transformed normal is undefined. If the GL implementation determines that the model-view matrix is uninvertible, then the entries in the inverted matrix are arbitrary. In any case, neither normal transformation nor use of the transformed normal may lead to GL interruption or termination.

2.10.4 Generating Texture Coordinates

Texture coordinates associated with a vertex may either be taken from the current texture coordinates or generated according to a function dependent on vertex coordinates. The command

```
void TexGen{ifd}( enum coord, enum pname, T param );
void TexGen{ifd}v( enum coord, enum pname, T params );
```

controls texture coordinate generation. *coord* must be one of the constants `S`, `T`, `R`, or `Q`, indicating that the pertinent coordinate is the s, t, r, or q

Microsoft Corp. Exhibit 1009

coordinate, respectively. In the first form of the command, *param* is a symbolic constant specifying a single-valued texture generation parameter; in the second form, *params* is a pointer to an array of values that specify texture generation parameters. *pname* must be one of the three symbolic constants `TEXTURE_GEN_MODE`, `OBJECT_PLANE`, or `EYE_PLANE`. If *pname* is `TEXTURE_GEN_MODE`, then either *params* points to or *param* is an integer that is one of the symbolic constants `OBJECT_LINEAR`, `EYE_LINEAR`, or `SPHERE_MAP`.

If `TEXTURE_GEN_MODE` indicates `OBJECT_LINEAR`, then the generation function for the coordinate indicated by *coord* is

$$g = p_1 x_o + p_2 y_o + p_3 z_o + p_4 w_o.$$

x_o, y_o, z_o, and w_o are the object coordinates of the vertex. $p_1, \ldots, p_4$ are specified by calling **TexGen** with *pname* set to `OBJECT_PLANE` in which case *params* points to an array containing $p_1, \ldots, p_4$. There is a distinct group of plane equation coefficients for each texture coordinate; *coord* indicates the coordinate to which the specified coefficients pertain.

If `TEXTURE_GEN_MODE` indicates `EYE_LINEAR`, then the function is

$$g = p'_1 x_e + p'_2 y_e + p'_3 z_e + p'_4 w_e$$

where

$$\begin{pmatrix} p'_1 & p'_2 & p'_3 & p'_4 \end{pmatrix} = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \end{pmatrix} M^{-1}$$

x_e, y_e, z_e, and w_e are the eye coordinates of the vertex. $p_1, \ldots, p_4$ are set by calling **TexGen** with *pname* set to `EYE_PLANE` in correspondence with setting the coefficients in the `OBJECT_PLANE` case. M is the model-view matrix in effect when $p_1, \ldots, p_4$ are specified. Computed texture coordinates may be inaccurate or undefined if M is poorly conditioned or singular.

When used with a suitably constructed texture image, calling **TexGen** with `TEXTURE_GEN_MODE` indicating `SPHERE_MAP` can simulate the reflected image of a spherical environment on a polygon. `SPHERE_MAP` texture coordinates are generated as follows. Denote the unit vector pointing from the origin to the vertex (in eye coordinates) by $\mathbf{u}$. Denote the current normal, after transformation to eye coordinates, by $\mathbf{n}'$. Let $\mathbf{r} = \begin{pmatrix} r_x & r_y & r_z \end{pmatrix}^T$, the reflection vector, be given by

$$\mathbf{r} = \mathbf{u} - 2\mathbf{n}'^T \left(\mathbf{n}'\mathbf{u}\right),$$

and let $m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$. Then the value assigned to an s coordinate (the first **TexGen** argument value is `S`) is $s = r_x/m + \frac{1}{2}$; the value

Microsoft Corp. Exhibit 1009

assigned to a t coordinate is $t = r_y/m + \frac{1}{2}$. Calling **TexGen** with a *coord* of either `R` or `Q` when *pname* indicates `SPHERE_MAP` generates the error `INVALID_ENUM`.

A texture coordinate generation function is enabled or disabled using **Enable** and **Disable** with an argument of `TEXTURE_GEN_S`, `TEXTURE_GEN_T`, `TEXTURE_GEN_R`, or `TEXTURE_GEN_Q` (each indicates the corresponding texture coordinate). When enabled, the specified texture coordinate is computed according to the current `EYE_LINEAR`, `OBJECT_LINEAR` or `SPHERE_MAP` specification, depending on the current setting of `TEXTURE_GEN_MODE` for that coordinate. When disabled, subsequent vertices will take the indicated texture coordinate from the current texture coordinates.

The state required for texture coordinate generation comprises a three-valued integer for each coordinate indicating coordinate generation mode, and a bit for each coordinate to indicate whether texture coordinate generation is enabled or disabled. In addition, four coefficients are required for the four coordinates for each of `EYE_LINEAR` and `OBJECT_LINEAR`. The initial state has the texture generation function disabled for all texture coordinates. The initial values of p_i for s are all 0 except p_1 which is one; for t all the p_i are zero except p_2, which is 1. The values of p_i for r and q are all 0. These values of p_i apply for both the `EYE_LINEAR` and `OBJECT_LINEAR` versions. Initially all texture generation modes are `EYE_LINEAR`.

2.11 Clipping

Primitives are clipped to the *clip volume*. In clip coordinates, the *view volume* is defined by

$$\begin{aligned} -w_c \leq x_c \leq w_c \\ -w_c \leq y_c \leq w_c \\ -w_c \leq z_c \leq w_c \end{aligned} .$$

This view volume may be further restricted by as many as n client-defined clip planes to generate the clip volume. (n is an implementation dependent maximum that must be at least 6.) Each client-defined plane specifies a half-space. The clip volume is the intersection of all such half-spaces with the view volume (if there no client-defined clip planes are enabled, the clip volume is the view volume).

A client-defined clip plane is specified with

`void` **ClipPlane**(`enum` *p*, `double` *eqn[4]*);

Microsoft Corp. Exhibit 1009

The value of the first argument, p, is a symbolic constant, CLIP_PLANEi, where i is an integer between 0 and $n-1$, indicating one of n client-defined clip planes. *eqn* is an array of four double-precision floating-point values. These are the coefficients of a plane equation in object coordinates: p_1, p_2, p_3, and p_4 (in that order). The inverse of the current model-view matrix is applied to these coefficients, at the time they are specified, yielding

$$(\,p'_1 \quad p'_2 \quad p'_3 \quad p'_4\,) = (\,p_1 \quad p_2 \quad p_3 \quad p_4\,)\,M^{-1}$$

(where M is the current model-view matrix; the resulting plane equation is undefined if M is singular and may be inaccurate if M is poorly-conditioned) to obtain the plane equation coefficients in eye coordinates. All points with eye coordinates $(\,x_e \quad y_e \quad z_e \quad w_e\,)^T$ that satisfy

$$(\,p'_1 \quad p'_2 \quad p'_3 \quad p'_4\,) \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} \geq 0$$

lie in the half-space defined by the plane; points that do not satisfy this condition do not lie in the half-space.

Client-defined clip planes are enabled with the generic **Enable** command and disabled with the **Disable** command. The value of the argument to either command is CLIP_PLANEi where i is an integer between 0 and n; specifying a value of i enables or disables the plane equation with index i. The constants obey CLIP_PLANEi = CLIP_PLANE0 + i.

If the primitive under consideration is a point, then clipping passes it unchanged if it lies within the clip volume; otherwise, it is discarded. If the primitive is a line segment, then clipping does nothing to it if it lies entirely within the clip volume and discards it if it lies entirely outside the volume. If part of the line segment lies in the volume and part lies outside, then the line segment is clipped and new vertex coordinates are computed for one or both vertices. A clipped line segment endpoint lies on both the original line segment and the boundary of the clip volume.

This clipping produces a value, $0 \leq t \leq 1$, for each clipped vertex. If the coordinates of a clipped vertex are $\mathbf{P}$ and the original vertices' coordinates are $\mathbf{P}_1$ and $\mathbf{P}_2$, then t is given by

$$\mathbf{P} = t\mathbf{P}_1 + (1-t)\mathbf{P}_2.$$

The value of t is used in color and texture coordinate clipping (section 2.13.8).

Microsoft Corp. Exhibit 1009

If the primitive is a polygon, then it is passed if every one of its edges lies entirely inside the clip volume and either clipped or discarded otherwise. Polygon clipping may cause polygon edges to be clipped, but because polygon connectivity must be maintained, these clipped edges are connected by new edges that lie along the clip volume's boundary. Thus, clipping may require the introduction of new vertices into a polygon. Edge flags are associated with these vertices so that edges introduced by clipping are flagged as boundary (edge flag `TRUE`), and so that original edges of the polygon that become cut off at these vertices retain their original flags.

If it happens that a polygon intersects an edge of the clip volume's boundary, then the clipped polygon must include a point on this boundary edge. This point must lie in the intersection of the boundary edge and the convex hull of the vertices of the original polygon. We impose this requirement because the polygon may not be exactly planar.

A line segment or polygon whose vertices have w_c values of differing signs may generate multiple connected components after clipping. GL implementations are not required to handle this situation. That is, only the portion of the primitive that lies in the region of $w_c > 0$ need be produced by clipping.

Primitives rendered with clip planes must satisfy a complementarity criterion. Suppose a single clip plane with coefficients $(\, p'_1 \quad p'_2 \quad p'_3 \quad p'_4 \,)$ (or a number of similarly specified clip planes) is enabled and a series of primitives are drawn. Next, suppose that the original clip plane is respecified with coefficients $(\, -p'_1 \quad -p'_2 \quad -p'_3 \quad -p'_4 \,)$ (and correspondingly for any other clip planes) and the primitives are drawn again (and the GL is otherwise in the same state). In this case, primitives must not be missing any pixels, nor may any pixels be drawn twice in regions where those primitives are cut by the clip planes.

The state required for clipping is at least 6 sets of plane equations (each consisting of four double-precision floating-point coefficients) and at least 6 corresponding bits indicating which of these client-defined plane equations are enabled. In the initial state, all client-defined plane equation coefficients are zero and all planes are disabled.

2.12 Current Raster Position

The *current raster position* is used by commands that directly affect pixels in the framebuffer. These commands, which bypass vertex transformation and primitive assembly, are described in the next chapter. The current raster position, however, shares some of the characteristics of a vertex.

Microsoft Corp. Exhibit 1009

The state required for the current raster position consists of three window coordinates x_w, y_w, and z_w, a clip coordinate w_c value, an eye coordinate distance, a valid bit, and associated data consisting of a color and texture coordinates. It is set using one of the **RasterPos** commands:

```
void RasterPos{234}{sifd}( T coords );
void RasterPos{234}{sifd}v( T coords );
```

RasterPos4 takes four values indicating x, y, z, and w. **RasterPos3** (or **RasterPos2**) is analogous, but sets only x, y, and z with w implicitly set to 1 (or only x and y with z implicitly set to 0 and w implicitly set to 1).

The coordinates are treated as if they were specified in a **Vertex** command. The x, y, z, and w coordinates are transformed by the current model-view and perspective matrices. These coordinates, along with current values, are used to generate a color and texture coordinates just as is done for a vertex. The color and texture coordinates so produced replace the color and texture coordinates stored in the current raster position's associated data. The distance from the origin of the eye coordinate system to the vertex as transformed by only the current model-view matrix replaces the current raster distance. This distance can be approximated (see section 3.10).

The transformed coordinates are passed to clipping as if they represented a point. If the "point" is not culled, then the projection to window coordinates is computed (section 2.10) and saved as the current raster position, and the valid bit is set. If the "point" is culled, the current raster position and its associated data become indeterminate and the valid bit is cleared. Figure 2.7 summarizes the behavior of the current raster position.

The current raster position requires five single-precision floating-point values for its x_w, y_w, and z_w window coordinates, its w_c clip coordinate, and its eye coordinate distance, a single valid bit, a color (RGBA and color index), and texture coordinates for associated data. In the initial state, the coordinates and texture coordinates are both $(0, 0, 0, 1)$, the eye coordinate distance is 0, the valid bit is set, the associated RGBA color is $(1, 1, 1, 1)$ and the associated color index color is 1. In RGBA mode, the associated color index always has its initial value; in color index mode, the RGBA color always maintains its initial value.

Microsoft Corp. Exhibit 1009

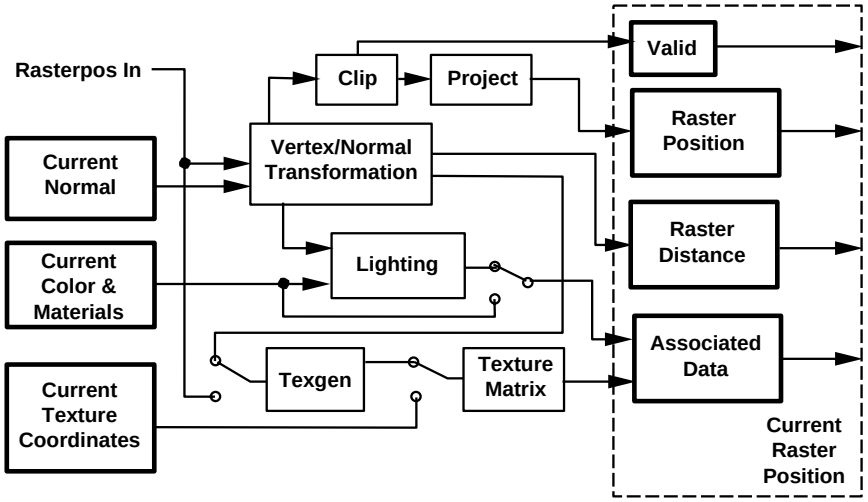


Figure 2.7. The current raster position and how it is set.

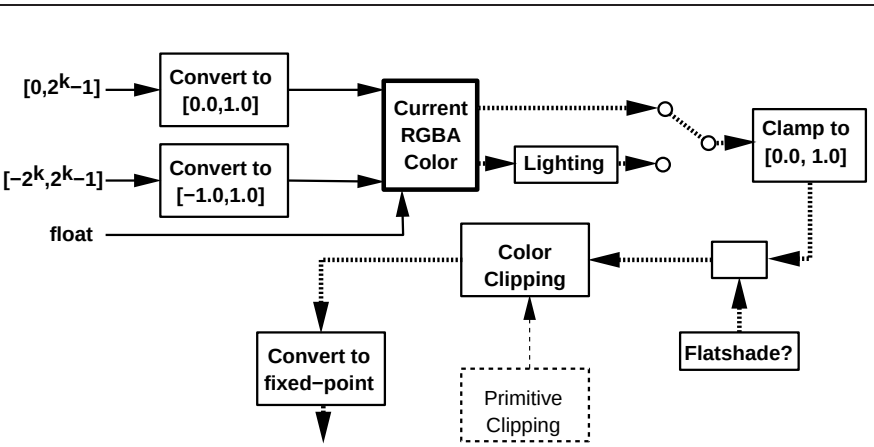


Figure 2.8. Processing of RGBA colors. The heavy dotted lines indicate both primary and secondary vertex colors, which are processed in the same fashion. See Table 2.6 for the interpretation of k.

Microsoft Corp. Exhibit 1009

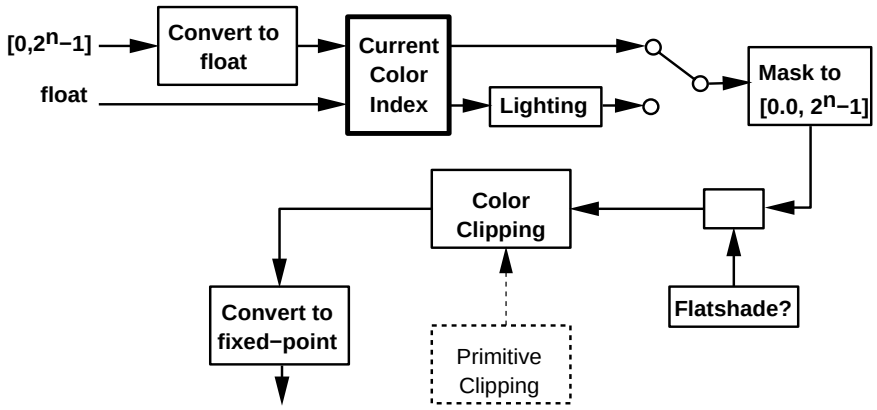


Figure 2.9. Processing of color indices. n is the number of bits in a color index.

2.13 Colors and Coloring

Figures 2.8 and 2.9 diagram the processing of RGBA colors and color indices before rasterization. Incoming colors arrive in one of several formats. Table 2.6 summarizes the conversions that take place on R, G, B, and A components depending on which version of the **Color** command was invoked to specify the components. As a result of limited precision, some converted values will not be represented exactly. In color index mode, a single-valued color index is not mapped.

Next, lighting, if enabled, produces either a color index or primary and secondary colors. If lighting is disabled, the current color index or color is used in further processing (the current color is the primary color, and the secondary color is $(0, 0, 0, 0)$). After lighting, RGBA colors are clamped to the range $[0, 1]$. A color index is converted to fixed-point and then its integer portion is masked (see section 2.13.6). After clamping or masking, a primitive may be *flatshaded*, indicating that all vertices of the primitive are to have the same color. Finally, if a primitive is clipped, then colors (and texture coordinates) must be computed at the vertices introduced or modified by clipping.

Microsoft Corp. Exhibit 1009

APPENDIX T

GL Type	Conversion
ubyte	$c/(2^8 - 1)$
byte	$(2c + 1)/(2^8 - 1)$
ushort	$c/(2^{16} - 1)$
short	$(2c + 1)/(2^{16} - 1)$
uint	$c/(2^{32} - 1)$
int	$(2c + 1)/(2^{32} - 1)$
float	c
double	c

Table 2.6: Component conversions. Color, normal, and depth components, (c), are converted to an internal floating-point representation, (f), using the equations in this table. All arithmetic is done in the internal floating point format. These conversions apply to components specified as parameters to GL commands and to components in pixel data. The equations remain the same even if the implemented ranges of the GL data types are greater than the minimum required ranges. (Refer to table 2.2)

2.13.1 Lighting

GL lighting computes colors for each vertex sent to the GL. This is accomplished by applying an equation defined by a client-specified lighting model to a collection of parameters that can include the vertex coordinates, the coordinates of one or more light sources, the current normal, and parameters defining the characteristics of the light sources and a current material. The following discussion assumes that the GL is in RGBA mode. (Color index lighting is described in section 2.13.5.)

Lighting may be in one of two states:

1. Lighting Off. In this state, the current color is assigned to the vertex primary color. The secondary color is $(0, 0, 0, 0)$.

2. Lighting On. In this state, the vertex primary and secondary colors are computed from the current lighting parameters.

Lighting is turned on or off using the generic **Enable** or **Disable** commands with the symbolic value `LIGHTING`.

Microsoft Corp. Exhibit 1009

Lighting Operation

A lighting parameter is of one of five types: color, position, direction, real, or boolean. A color parameter consists of four floating-point values, one for each of R, G, B, and A, in that order. There are no restrictions on the allowable values for these parameters. A position parameter consists of four floating-point coordinates (x, y, z, and w) that specify a position in object coordinates (w may be zero, indicating a point at infinity in the direction given by x, y, and z). A direction parameter consists of three floating-point coordinates (x, y, and z) that specify a direction in object coordinates. A real parameter is one floating-point value. The various values and their types are summarized in Table 2.7. The result of a lighting computation is undefined if a value for a parameter is specified that is outside the range given for that parameter in the table.

There are n light sources, indexed by $i = 0, \ldots, n-1$. (n is an implementation dependent maximum that must be at least 8.) Note that the default values for $\mathbf{d}_{cli}$ and $\mathbf{s}_{cli}$ differ for $i = 0$ and $i > 0$.

Before specifying the way that lighting computes colors, we introduce operators and notation that simplify the expressions involved. If $\mathbf{c}_1$ and $\mathbf{c}_2$ are colors without alpha where $\mathbf{c}_1 = (r_1, g_1, b_1)$ and $\mathbf{c}_2 = (r_2, g_2, b_2)$, then define $\mathbf{c}_1 * \mathbf{c}_2 = (r_1 r_2, g_1 g_2, b_1 b_2)$. Addition of colors is accomplished by addition of the components. Multiplication of colors by a scalar means multiplying each component by that scalar. If $\mathbf{d}_1$ and $\mathbf{d}_2$ are directions, then define

$$\mathbf{d}_1 \odot \mathbf{d}_2 = \max\{\mathbf{d}_1 \cdot \mathbf{d}_2, 0\}.$$

(Directions are taken to have three coordinates.) If $\mathbf{P}_1$ and $\mathbf{P}_2$ are (homogeneous, with four coordinates) points then let $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ be the unit vector that points from $\mathbf{P}_1$ to $\mathbf{P}_2$. Note that if $\mathbf{P}_2$ has a zero w coordinate and $\mathbf{P}_1$ has non-zero w coordinate, then $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ is the unit vector corresponding to the direction specified by the x, y, and z coordinates of $\mathbf{P}_2$; if $\mathbf{P}_1$ has a zero w coordinate and $\mathbf{P}_2$ has a non-zero w coordinate then $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ is the unit vector that is the negative of that corresponding to the direction specified by $\mathbf{P}_1$. If both $\mathbf{P}_1$ and $\mathbf{P}_2$ have zero w coordinates, then $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ is the unit vector obtained by normalizing the direction corresponding to $\mathbf{P}_2 - \mathbf{P}_1$.

If $\mathbf{d}$ is an arbitrary direction, then let $\hat{\mathbf{d}}$ be the unit vector in $\mathbf{d}$'s direction. Let $\|\mathbf{P}_1\mathbf{P}_2\|$ be the distance between $\mathbf{P}_1$ and $\mathbf{P}_2$. Finally, let $\mathbf{V}$ be the point corresponding to the vertex being lit, and $\mathbf{n}$ be the corresponding normal. Let $\mathbf{P}_e$ be the eyepoint ($(0, 0, 0, 1)$ in eye coordinates).

Lighting produces two colors at a vertex: a primary color $\mathbf{c}_{pri}$ and a secondary color $\mathbf{c}_{sec}$. The values of $\mathbf{c}_{pri}$ and $\mathbf{c}_{sec}$ depend on the light model

Microsoft Corp. Exhibit 1009

Parameter	Type	Default Value	Description
Material Parameters			
$\mathbf{a}_{cm}$	color	$(0.2, 0.2, 0.2, 1.0)$	ambient color of material
$\mathbf{d}_{cm}$	color	$(0.8, 0.8, 0.8, 1.0)$	diffuse color of material
$\mathbf{s}_{cm}$	color	$(0.0, 0.0, 0.0, 1.0)$	specular color of material
$\mathbf{e}_{cm}$	color	$(0.0, 0.0, 0.0, 1.0)$	emissive color of material
s_{rm}	real	0.0	specular exponent (range: $[0.0, 128.0]$)
a_m	real	0.0	ambient color index
d_m	real	1.0	diffuse color index
s_m	real	1.0	specular color index
Light Source Parameters			
$\mathbf{a}_{cli}$	color	$(0.0, 0.0, 0.0, 1.0)$	ambient intensity of light i
$\mathbf{d}_{cli}(i = 0)$	color	$(1.0, 1.0, 1.0, 1.0)$	diffuse intensity of light 0
$\mathbf{d}_{cli}(i > 0)$	color	$(0.0, 0.0, 0.0, 1.0)$	diffuse intensity of light i
$\mathbf{s}_{cli}(i = 0)$	color	$(1.0, 1.0, 1.0, 1.0)$	specular intensity of light 0
$\mathbf{s}_{cli}(i > 0)$	color	$(0.0, 0.0, 0.0, 1.0)$	specular intensity of light i
$\mathbf{P}_{pli}$	position	$(0.0, 0.0, 1.0, 0.0)$	position of light i
$\mathbf{s}_{dli}$	direction	$(0.0, 0.0, -1.0)$	direction of spotlight for light i
s_{rli}	real	0.0	spotlight exponent for light i (range: $[0.0, 128.0]$)
c_{rli}	real	180.0	spotlight cutoff angle for light i (range: $[0.0, 90.0]$, 180.0)
k_{0i}	real	1.0	constant attenuation factor for light i (range: $[0.0, \infty)$)
k_{1i}	real	0.0	linear attenuation factor for light i (range: $[0.0, \infty)$)
k_{2i}	real	0.0	quadratic attenuation factor for light i (range: $[0.0, \infty)$)
Lighting Model Parameters			
$\mathbf{a}_{cs}$	color	$(0.2, 0.2, 0.2, 1.0)$	ambient color of scene
v_{bs}	boolean	`FALSE`	viewer assumed to be at $(0, 0, 0)$ in eye coordinates (`TRUE`) or $(0, 0, \infty)$ (`FALSE`)
c_{es}	enum	`SINGLE_COLOR`	controls computation of colors
t_{bs}	boolean	`FALSE`	use two-sided lighting mode

Table 2.7: Summary of lighting parameters. The range of individual color components is $(-\infty, +\infty)$.

Microsoft Corp. Exhibit 1009

color control, c_{es}. If $c_{es} =$ `SINGLE_COLOR`, then the equations to compute $\mathbf{c}_{pri}$ and $\mathbf{c}_{sec}$ are

$$\begin{aligned}
\mathbf{c}_{pri} &= \mathbf{e}_{cm} \\
&+ \mathbf{a}_{cm} * \mathbf{a}_{cs} \\
&+ \sum_{i=0}^{n-1} (att_i)(spot_i) \left[\mathbf{a}_{cm} * \mathbf{a}_{cli} \right. \\
&\qquad + (\mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli}) \mathbf{d}_{cm} * \mathbf{d}_{cli} \\
&\qquad \left. + (f_i)(\mathbf{n} \odot \hat{\mathbf{h}}_i)^{s_{rm}} \mathbf{s}_{cm} * \mathbf{s}_{cli} \right] \\
\mathbf{c}_{sec} &= (0, 0, 0, 0)
\end{aligned}$$

If $c_{es} =$ `SEPARATE_SPECULAR_COLOR`, then

$$\begin{aligned}
\mathbf{c}_{pri} &= \mathbf{e}_{cm} \\
&+ \mathbf{a}_{cm} * \mathbf{a}_{cs} \\
&+ \sum_{i=0}^{n-1} (att_i)(spot_i) \left[\mathbf{a}_{cm} * \mathbf{a}_{cli} \right. \\
&\qquad \left. + (\mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli}) \mathbf{d}_{cm} * \mathbf{d}_{cli} \right] \\
\mathbf{c}_{sec} &= \sum_{i=0}^{n-1} (att_i)(spot_i)(f_i)(\mathbf{n} \odot \hat{\mathbf{h}}_i)^{s_{rm}} \mathbf{s}_{cm} * \mathbf{s}_{cli}
\end{aligned}$$

where

$$f_i = \begin{cases} 1, & \mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli} \neq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{2.2}$$

$$\mathbf{h}_i = \begin{cases} \overrightarrow{\mathbf{VP}}_{pli} + \overrightarrow{\mathbf{VP}}_e, & v_{bs} = \texttt{TRUE}, \\ \overrightarrow{\mathbf{VP}}_{pli} + (\,0 \quad 0 \quad 1\,)^T, & v_{bs} = \texttt{FALSE}, \end{cases} \tag{2.3}$$

$$att_i = \begin{cases} \dfrac{1}{k_{0i} + k_{1i}\|\mathbf{VP}_{pli}\| + k_{2i}\|\mathbf{VP}_{pli}\|^2}, & \text{if } \mathbf{P}_{pli}\text{'s } w \neq 0, \\ 1.0, & \text{otherwise.} \end{cases} \tag{2.4}$$

Microsoft Corp. Exhibit 1009

$$spot_i = \begin{cases} (\overrightarrow{\mathbf{P}_{pli}\mathbf{V}} \odot \hat{\mathbf{s}}_{dli})^{s_{rli}}, & c_{rli} \neq 180.0, \overrightarrow{\mathbf{P}_{pli}\mathbf{V}} \odot \hat{\mathbf{s}}_{dli} \geq \cos(c_{rli}), \\ 0.0, & c_{rli} \neq 180.0, \overrightarrow{\mathbf{P}_{pli}\mathbf{V}} \odot \hat{\mathbf{s}}_{dli} < \cos(c_{rli}), \\ 1.0, & c_{rli} = 180.0. \end{cases} \quad (2.5)$$

(2.6)

All computations are carried out in eye coordinates.

The value of A produced by lighting is the alpha value associated with $\mathbf{d}_{cm}$. A is always associated with the primary color $\mathbf{c}_{pri}$; the alpha component of $\mathbf{c}_{sec}$ is 0. Results of lighting are undefined if the w_e coordinate (w in eye coordinates) of $\mathbf{V}$ is zero.

Lighting may operate in *two-sided* mode (t_{bs} = `TRUE`), in which a *front* color is computed with one set of material parameters (the *front material*) and a *back* color is computed with a second set of material parameters (the *back material*). This second computation replaces $\mathbf{n}$ with $-\mathbf{n}$. If t_{bs} = `FALSE`, then the back color and front color are both assigned the color computed using the front material with $\mathbf{n}$.

The selection between back color and front color depends on the primitive of which the vertex being lit is a part. If the primitive is a point or a line segment, the front color is always selected. If it is a polygon, then the selection is based on the sign of the (clipped or unclipped) polygon's signed area computed in window coordinates. One way to compute this area is

$$a = \frac{1}{2} \sum_{i=0}^{n-1} x_w^i y_w^{i \oplus 1} - x_w^{i \oplus 1} y_w^i \quad (2.7)$$

where x_w^i and y_w^i are the x and y window coordinates of the ith vertex of the n-vertex polygon (vertices are numbered starting at zero for purposes of this computation) and $i \oplus 1$ is $(i+1) \bmod n$. The interpretation of the sign of this value is controlled with

`void` **FrontFace**(`enum` *dir*);

Setting *dir* to `CCW` (corresponding to counter-clockwise orientation of the projected polygon in window coordinates) indicates that if $a \leq 0$, then the color of each vertex of the polygon becomes the back color computed for that vertex while if $a > 0$, then the front color is selected. If *dir* is `CW`, then a is replaced by $-a$ in the above inequalities. This requires one bit of state; initially, it indicates `CCW`.

Microsoft Corp. Exhibit 1009

2.13.2 Lighting Parameter Specification

Lighting parameters are divided into three categories: material parameters, light source parameters, and lighting model parameters (see Table 2.7). Sets of lighting parameters are specified with

```
void Material{if}( enum face, enum pname, T param );
void Material{if}v( enum face, enum pname, T params );
void Light{if}( enum light, enum pname, T param );
void Light{if}v( enum light, enum pname, T params );
void LightModel{if}( enum pname, T param );
void LightModel{if}v( enum pname, T params );
```

pname is a symbolic constant indicating which parameter is to be set (see Table 2.8). In the vector versions of the commands, *params* is a pointer to a group of values to which to set the indicated parameter. The number of values pointed to depends on the parameter being set. In the non-vector versions, *param* is a value to which to set a single-valued parameter. (If *param* corresponds to a multi-valued parameter, the error `INVALID_ENUM` results.) For the **Material** command, *face* must be one of `FRONT`, `BACK`, or `FRONT_AND_BACK`, indicating that the property *name* of the front or back material, or both, respectively, should be set. In the case of **Light**, *light* is a symbolic constant of the form `LIGHT`i, indicating that light i is to have the specified parameter set. The constants obey `LIGHT`i = `LIGHT0` + i.

Table 2.8 gives, for each of the three parameter groups, the correspondence between the pre-defined constant names and their names in the lighting equations, along with the number of values that must be specified with each. Color parameters specified with **Material** and **Light** are converted to floating-point values (if specified as integers) as indicated in Table 2.6 for signed integers. The error `INVALID_VALUE` occurs if a specified lighting parameter lies outside the allowable range given in Table 2.7. (The symbol "∞" indicates the maximum representable magnitude for the indicated type.)

The current model-view matrix is applied to the position parameter indicated with **Light** for a particular light source when that position is specified. These transformed values are the values used in the lighting equation.

The spotlight direction is transformed when it is specified using only the upper leftmost 3x3 portion of the model-view matrix. That is, if $\mathbf{M}_u$ is the upper left 3x3 matrix taken from the current model-view matrix M, then

Microsoft Corp. Exhibit 1009

APPENDIX T

Parameter	Name	Number of values
Material Parameters (**Material**)		
$\mathbf{a}_{cm}$	AMBIENT	4
$\mathbf{d}_{cm}$	DIFFUSE	4
$\mathbf{a}_{cm}, \mathbf{d}_{cm}$	AMBIENT_AND_DIFFUSE	4
$\mathbf{s}_{cm}$	SPECULAR	4
$\mathbf{e}_{cm}$	EMISSION	4
s_{rm}	SHININESS	1
a_m, d_m, s_m	COLOR_INDEXES	3
Light Source Parameters (**Light**)		
$\mathbf{a}_{cli}$	AMBIENT	4
$\mathbf{d}_{cli}$	DIFFUSE	4
$\mathbf{s}_{cli}$	SPECULAR	4
$\mathbf{P}_{pli}$	POSITION	4
$\mathbf{s}_{dli}$	SPOT_DIRECTION	3
s_{rli}	SPOT_EXPONENT	1
c_{rli}	SPOT_CUTOFF	1
k_0	CONSTANT_ATTENUATION	1
k_1	LINEAR_ATTENUATION	1
k_2	QUADRATIC_ATTENUATION	1
Lighting Model Parameters (**LightModel**)		
$\mathbf{a}_{cs}$	LIGHT_MODEL_AMBIENT	4
v_{bs}	LIGHT_MODEL_LOCAL_VIEWER	1
t_{bs}	LIGHT_MODEL_TWO_SIDE	1
c_{es}	LIGHT_MODEL_COLOR_CONTROL	1

Table 2.8: Correspondence of lighting parameter symbols to names. AMBIENT_AND_DIFFUSE is used to set $\mathbf{a}_{cm}$ and $\mathbf{d}_{cm}$ to the same value.

Microsoft Corp. Exhibit 1009

the spotlight direction

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$$

is transformed to

$$\begin{pmatrix} d'_x \\ d'_y \\ d'_z \end{pmatrix} = M_u \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}.$$

An individual light is enabled or disabled by calling **Enable** or **Disable** with the symbolic value `LIGHT`i (i is in the range 0 to $n-1$, where n is the implementation-dependent number of lights). If light i is disabled, the ith term in the lighting equation is effectively removed from the summation.

2.13.3 ColorMaterial

It is possible to attach one or more material properties to the current color, so that they continuously track its component values. This behavior is enabled and disabled by calling **Enable** or **Disable** with the symbolic value `COLOR_MATERIAL`.

The command that controls which of these modes is selected is

`void` **ColorMaterial**(`enum` *face*, `enum` *mode*);

face is one of `FRONT`, `BACK`, or `FRONT_AND_BACK`, indicating whether the front material, back material, or both are affected by the current color. *mode* is one of `EMISSION`, `AMBIENT`, `DIFFUSE`, `SPECULAR`, or `AMBIENT_AND_DIFFUSE` and specifies which material property or properties track the current color. If *mode* is `EMISSION`, `AMBIENT`, `DIFFUSE`, or `SPECULAR`, then the value of $\mathbf{e}_{cm}$, $\mathbf{a}_{cm}$, $\mathbf{d}_{cm}$ or $\mathbf{s}_{cm}$, respectively, will track the current color. If *mode* is `AMBIENT_AND_DIFFUSE`, both $\mathbf{a}_{cm}$ and $\mathbf{d}_{cm}$ track the current color. The replacements made to material properties are permanent; the replaced values remain until changed by either sending a new color or by setting a new material value when **ColorMaterial** is not currently enabled to override that particular value. When `COLOR_MATERIAL` is enabled, the indicated parameter or parameters always track the current color. For instance, calling

ColorMaterial(`FRONT`, `AMBIENT`)

while `COLOR_MATERIAL` is enabled sets the front material $\mathbf{a}_{cm}$ to the value of the current color.

Microsoft Corp. Exhibit 1009

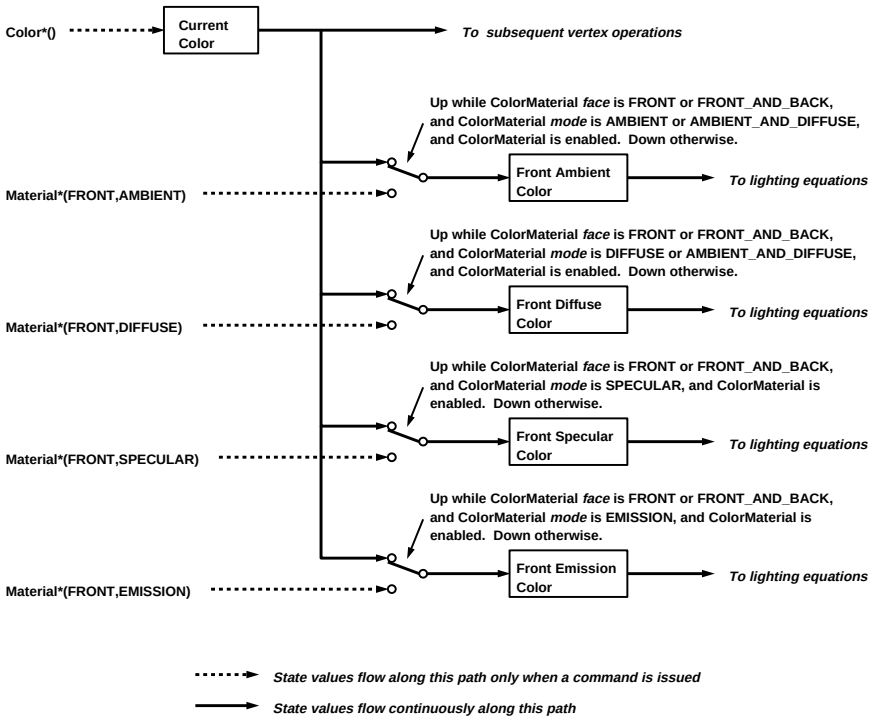


Figure 2.10. **ColorMaterial** operation. Material properties are continuously updated from the current color while **ColorMaterial** is enabled and has the appropriate mode. Only the front material properties are included in this figure. The back material properties are treated identically, except that *face* must be `BACK` or `FRONT_AND_BACK`.

Microsoft Corp. Exhibit 1009

APPENDIX T

2.13.4 Lighting State

The state required for lighting consists of all of the lighting parameters (front and back material parameters, lighting model parameters, and at least 8 sets of light parameters), a bit indicating whether a back color distinct from the front color should be computed, at least 8 bits to indicate which lights are enabled, a five-valued variable indicating the current **ColorMaterial** mode, a bit indicating whether or not `COLOR_MATERIAL` is enabled, and a single bit to indicate whether lighting is enabled or disabled. In the initial state, all lighting parameters have their default values. Back color evaluation does not take place, **ColorMaterial** is `FRONT_AND_BACK` and `AMBIENT_AND_DIFFUSE`, and both lighting and `COLOR_MATERIAL` are disabled.

2.13.5 Color Index Lighting

A simplified lighting computation applies in color index mode that uses many of the parameters controlling RGBA lighting, but none of the RGBA material parameters. First, the RGBA diffuse and specular intensities of light i ($\mathbf{d}_{cli}$ and $\mathbf{s}_{cli}$, respectively) determine color index diffuse and specular light intensities, d_{li} and s_{li} from

$$d_{li} = (.30)R(\mathbf{d}_{cli}) + (.59)G(\mathbf{d}_{cli}) + (.11)B(\mathbf{d}_{cli})$$

and

$$s_{li} = (.30)R(\mathbf{s}_{cli}) + (.59)G(\mathbf{s}_{cli}) + (.11)B(\mathbf{s}_{cli}).$$

$R(\mathbf{x})$ indicates the R component of the color $\mathbf{x}$ and similarly for $G(\mathbf{x})$ and $B(\mathbf{x})$.

Next, let

$$s = \sum_{i=0}^{n} (att_i)(spot_i)(s_{li})(f_i)(\mathbf{n} \odot \hat{\mathbf{h}}_i)^{s_{rm}}$$

where att_i and $spot_i$ are given by equations 2.4 and 2.5, respectively, and f_i and $\hat{\mathbf{h}}_i$ are given by equations 2.2 and 2.3, respectively. Let $s' = \min\{s, 1\}$. Finally, let

$$d = \sum_{i=0}^{n} (att_i)(spot_i)(d_{li})(\mathbf{n} \odot \overrightarrow{\mathbf{VP}}_{pli}).$$

Then color index lighting produces a value c, given by

$$c = a_m + d(1 - s')(d_m - a_m) + s'(s_m - a_m).$$

The final color index is

$$c' = \min\{c, s_m\}.$$

Microsoft Corp. Exhibit 1009

The values a_m, d_m and s_m are material properties described in Tables 2.7 and 2.8. Any ambient light intensities are incorporated into a_m. As with RGBA lighting, disabled lights cause the corresponding terms from the summations to be omitted. The interpretation of t_{bs} and the calculation of front and back colors is carried out as has already been described for RGBA lighting.

The values a_m, d_m, and s_m are set with **Material** using a *pname* of `COLOR_INDEXES`. Their initial values are 0, 1, and 1, respectively. The additional state consists of three floating-point values. These values have no effect on RGBA lighting.

2.13.6 Clamping or Masking

After lighting (whether enabled or not), all components of both primary and secondary colors are clamped to the range $[0, 1]$.

For a color index, the index is first converted to fixed-point with an unspecified number of bits to the right of the binary point; the nearest fixed-point value is selected. Then, the bits to the right of the binary point are left alone while the integer portion is masked (bitwise ANDed) with $2^n - 1$, where n is the number of bits in a color in the color index buffer (buffers are discussed in chapter 4).

2.13.7 Flatshading

A primitive may be *flatshaded*, meaning that all vertices of the primitive are assigned the same color index or the same primary and secondary colors. These colors are the colors of the vertex that spawned the primitive. For a point, these are the colors associated with the point. For a line segment, they are the colors of the second (final) vertex of the segment. For a polygon, they come from a selected vertex depending on how the polygon was generated. Table 2.9 summarizes the possibilities.

Flatshading is controlled by

```
void ShadeModel( enum mode );
```

mode value must be either of the symbolic constants `SMOOTH` or `FLAT`. If *mode* is `SMOOTH` (the initial state), vertex colors are treated individually. If *mode* is `FLAT`, flatshading is turned on. **ShadeModel** thus requires one bit of state.

Microsoft Corp. Exhibit 1009

Primitive type of polygon i	Vertex
single polygon ($i \equiv 1$)	1
triangle strip	$i+2$
triangle fan	$i+2$
independent triangle	$3i$
quad strip	$2i+2$
independent quad	$4i$

Table 2.9: Polygon flatshading color selection. The colors used for flatshading the ith polygon generated by the indicated **Begin**/**End** type are derived from the current color (if lighting is disabled) in effect when the indicated vertex is specified. If lighting is enabled, the colors are produced by lighting the indicated vertex. Vertices are numbered 1 through n, where n is the number of vertices between the **Begin**/**End** pair.

2.13.8 Color and Texture Coordinate Clipping

After lighting, clamping or masking and possible flatshading, colors are clipped. Those colors associated with a vertex that lies within the clip volume are unaffected by clipping. If a primitive is clipped, however, the colors assigned to vertices produced by clipping are clipped colors.

Let the colors assigned to the two vertices $\mathbf{P}_1$ and $\mathbf{P}_2$ of an unclipped edge be $\mathbf{c}_1$ and $\mathbf{c}_2$. The value of t (section 2.11) for a clipped point $\mathbf{P}$ is used to obtain the color associated with $\mathbf{P}$ as

$$\mathbf{c} = t\mathbf{c}_1 + (1-t)\mathbf{c}_2.$$

(For a color index color, multiplying a color by a scalar means multiplying the index by the scalar. For an RGBA color, it means multiplying each of R, G, B, and A by the scalar. Both primary and secondary colors are treated in the same fashion.) Polygon clipping may create a clipped vertex along an edge of the clip volume's boundary. This situation is handled by noting that polygon clipping proceeds by clipping against one plane of the clip volume's boundary at a time. Color clipping is done in the same way, so that clipped points always occur at the intersection of polygon edges (possibly already clipped) with the clip volume's boundary.

Texture coordinates must also be clipped when a primitive is clipped. The method is exactly analogous to that used for color clipping.

Microsoft Corp. Exhibit 1009

APPENDIX T

2.13.9 Final Color Processing

For an RGBA color, each color component (which lies in $[0, 1]$) is converted (by rounding to nearest) to a fixed-point value with m bits. We assume that the fixed-point representation used represents each value $k/(2^m - 1)$, where $k \in \{0, 1, \ldots, 2^m - 1\}$, as k (e.g. 1.0 is represented in binary as a string of all ones). m must be at least as large as the number of bits in the corresponding component of the framebuffer. m must be at least 2 for A if the framebuffer does not contain an A component, or if there is only 1 bit of A in the framebuffer. A color index is converted (by rounding to nearest) to a fixed-point value with at least as many bits as there are in the color index portion of the framebuffer.

Because a number of the form $k/(2^m - 1)$ may not be represented exactly as a limited-precision floating-point quantity, we place a further requirement on the fixed-point conversion of RGBA components. Suppose that lighting is disabled, the color associated with a vertex has not been clipped, and one of **Colorub**, **Colorus**, or **Colorui** was used to specify that color. When these conditions are satisfied, an RGBA component must convert to a value that matches the component as specified in the **Color** command: if m is less than the number of bits b with which the component was specified, then the converted value must equal the most significant m bits of the specified value; otherwise, the most significant b bits of the converted value must equal the specified value.

Microsoft Corp. Exhibit 1009

Chapter 3

Rasterization

Rasterization is the process by which a primitive is converted to a two-dimensional image. Each point of this image contains such information as color and depth. Thus, rasterizing a primitive consists of two parts. The first is to determine which squares of an integer grid in window coordinates are occupied by the primitive. The second is assigning a color and a depth value to each such square. The results of this process are passed on to the next stage of the GL (per-fragment operations), which uses the information to update the appropriate locations in the framebuffer. Figure 3.1 diagrams the rasterization process.

A grid square along with its parameters of assigned color, z (depth), and texture coordinates is called a *fragment*; the parameters are collectively dubbed the fragment's *associated data*. A fragment is located by its lower-left corner, which lies on integer grid coordinates. Rasterization operations also refer to a fragment's *center*, which is offset by $(1/2, 1/2)$ from its lower-left corner (and so lies on half-integer coordinates).

Grid squares need not actually be square in the GL. Rasterization rules are not affected by the actual aspect ratio of the grid squares. Display of non-square grids, however, will cause rasterized points and line segments to appear fatter in one direction than the other. We assume that fragments are square, since it simplifies antialiasing and texturing.

Several factors affect rasterization. Lines and polygons may be stippled. Points may be given differing diameters and line segments differing widths. A point, line segment, or polygon may be antialiased.

Microsoft Corp. Exhibit 1009

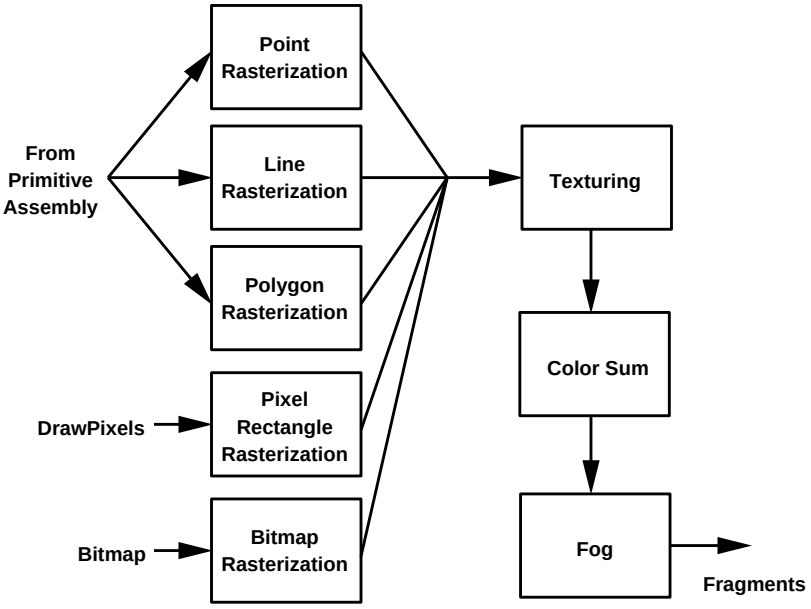


Figure 3.1. Rasterization.

Microsoft Corp. Exhibit 1009

3.1 Invariance

Consider a primitive p' obtained by translating a primitive p through an offset (x, y) in window coordinates, where x and y are integers. As long as neither p' nor p is clipped, it must be the case that each fragment f' produced from p' is identical to a corresponding fragment f from p except that the center of f' is offset by (x, y) from the center of f.

3.2 Antialiasing

Antialiasing of a point, line, or polygon is effected in one of two ways depending on whether the GL is in RGBA or color index mode.

In RGBA mode, the R, G, and B values of the rasterized fragment are left unaffected, but the A value is multiplied by a floating-point value in the range $[0, 1]$ that describes a fragment's screen pixel coverage. The per-fragment stage of the GL can be set up to use the A value to blend the incoming fragment with the corresponding pixel already present in the framebuffer.

In color index mode, the least significant b bits (to the left of the binary point) of the color index are used for antialiasing; $b = \min\{4, m\}$, where m is the number of bits in the color index portion of the framebuffer. The antialiasing process sets these b bits based on the fragment's coverage value: the bits are set to zero for no coverage and to all ones for complete coverage.

The details of how antialiased fragment coverage values are computed are difficult to specify in general. The reason is that high-quality antialiasing may take into account perceptual issues as well as characteristics of the monitor on which the contents of the framebuffer are displayed. Such details cannot be addressed within the scope of this document. Further, the coverage value computed for a fragment of some primitive may depend on the primitive's relationship to a number of grid squares neighboring the one corresponding to the fragment, and not just on the fragment's grid square. Another consideration is that accurate calculation of coverage values may be computationally expensive; consequently we allow a given GL implementation to approximate true coverage values by using a fast but not entirely accurate coverage computation.

In light of these considerations, we chose to specify the behavior of exact antialiasing in the prototypical case that each displayed pixel is a perfect square of uniform intensity. The square is called a *fragment square* and has lower left corner (x, y) and upper right corner $(x + 1, y + 1)$. We recognize

Microsoft Corp. Exhibit 1009

that this simple box filter may not produce the most favorable antialiasing results, but it provides a simple, well-defined model.

A GL implementation may use other methods to perform antialiasing, subject to the following conditions:

1. If f_1 and f_2 are two fragments, and the portion of f_1 covered by some primitive is a subset of the corresponding portion of f_2 covered by the primitive, then the coverage computed for f_1 must be less than or equal to that computed for f_2.

2. The coverage computation for a fragment f must be local: it may depend only on f's relationship to the boundary of the primitive being rasterized. It may not depend on f's x and y coordinates.

Another property that is desirable, but not required, is:

3. The sum of the coverage values for all fragments produced by rasterizing a particular primitive must be constant, independent of any rigid motions in window coordinates, as long as none of those fragments lies along window edges.

In some implementations, varying degrees of antialiasing quality may be obtained by providing GL hints (section 5.6), allowing a user to make an image quality versus speed tradeoff.

3.3 Points

The rasterization of points is controlled with

`void` **PointSize**(`float` *size*);

size specifies the width or diameter of a point. The default value is 1.0. A value less than or equal to zero results in the error `INVALID_VALUE`.

Point antialiasing is enabled or disabled by calling **Enable** or **Disable** with the symbolic constant `POINT_SMOOTH`. The default state is for point antialiasing to be disabled.

In the default state, a point is rasterized by truncating its x_w and y_w coordinates (recall that the subscripts indicate that these are x and y window coordinates) to integers. This (x, y) address, along with data derived from the data associated with the vertex corresponding to the point, is sent as a single fragment to the per-fragment stage of the GL.

Microsoft Corp. Exhibit 1009

The effect of a point width other than 1.0 depends on the state of point antialiasing. If antialiasing is disabled, the actual width is determined by rounding the supplied width to the nearest integer, then clamping it to the implementation-dependent maximum non-antialiased point width. This implementation-dependent value must be no less than the implementation-dependent maximum antialiased point width, rounded to the nearest integer value, and in any event no less than 1. If rounding the specified width results in the value 0, then it is as if the value were 1. If the resulting width is odd, then the point

$$(x, y) = (\lfloor x_w \rfloor + \frac{1}{2}, \lfloor y_w \rfloor + \frac{1}{2})$$

is computed from the vertex's x_w and y_w, and a square grid of the odd width centered at (x, y) defines the centers of the rasterized fragments (recall that fragment centers lie at half-integer window coordinate values). If the width is even, then the center point is

$$(x, y) = (\lfloor x_w + \frac{1}{2} \rfloor, \lfloor y_w + \frac{1}{2} \rfloor);$$

the rasterized fragment centers are the half-integer window coordinate values within the square of the even width centered on (x, y). See figure 3.2.

All fragments produced in rasterizing a non-antialiased point are assigned the same associated data, which are those of the vertex corresponding to the point, with texture coordinates s, t, and r replaced with s/q, t/q, and r/q, respectively. If q is less than or equal to zero, the results are undefined.

If antialiasing is enabled, then point rasterization produces a fragment for each fragment square that intersects the region lying within the circle having diameter equal to the current point width and centered at the point's (x_w, y_w) (figure 3.3). The coverage value for each fragment is the window coordinate area of the intersection of the circular region with the corresponding fragment square (but see section 3.2). This value is saved and used in the final step of rasterization (section 3.11). The data associated with each fragment are otherwise the data associated with the point being rasterized, with texture coordinates s, t, and r replaced with s/q, t/q, and r/q, respectively. If q is less than or equal to zero, the results are undefined.

Not all widths need be supported when point antialiasing is on, but the width 1.0 must be provided. If an unsupported width is requested, the nearest supported width is used instead. The range of supported widths and the width of evenly-spaced gradations within that range are implementation dependent. The range and gradations may be obtained using the query

Microsoft Corp. Exhibit 1009

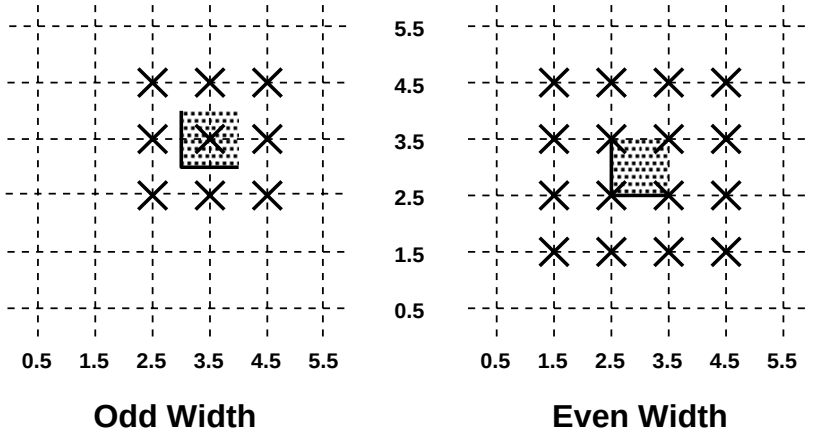


Figure 3.2. Rasterization of non-antialiased wide points. The crosses show fragment centers produced by rasterization for any point that lies within the shaded region. The dotted grid lines lie on half-integer coordinates.

mechanism described in Chapter 6. If, for instance, the width range is from 0.1 to 2.0 and the gradation width is 0.1, then the widths $0.1, 0.2, \ldots, 1.9, 2.0$ are supported.

3.3.1 Point Rasterization State

The state required to control point rasterization consists of the floating-point point width and a bit indicating whether or not antialiasing is enabled.

3.4 Line Segments

A line segment results from a line strip **Begin**/**End** object, a line loop, or a series of separate line segments. Line segment rasterization is controlled by several variables. Line width, which may be set by calling

`void` **LineWidth**(`float` *width*);

with an appropriate positive floating-point width, controls the width of rasterized line segments. The default width is 1.0. Values less than or equal

Microsoft Corp. Exhibit 1009

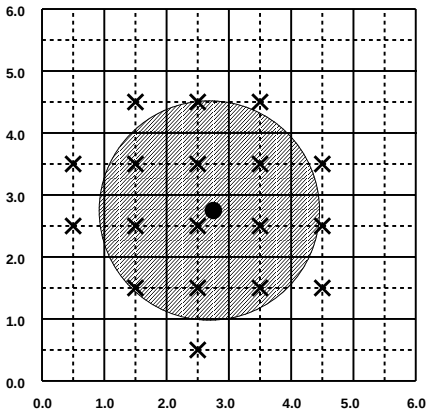


Figure 3.3. Rasterization of antialiased wide points. The black dot indicates the point to be rasterized. The shaded region has the specified width. The X marks indicate those fragment centers produced by rasterization. A fragment's computed coverage value is based on the portion of the shaded region that covers the corresponding fragment square. Solid lines lie on integer coordinates.

Microsoft Corp. Exhibit 1009

to 0.0 generate the error INVALID_VALUE. Antialiasing is controlled with **Enable** and **Disable** using the symbolic constant LINE_SMOOTH. Finally, line segments may be stippled. Stippling is controlled by a GL command that sets a *stipple pattern* (see below).

3.4.1 Basic Line Segment Rasterization

Line segment rasterization begins by characterizing the segment as either *x-major* or *y-major*. x-major line segments have slope in the closed interval $[-1, 1]$; all other line segments are y-major (slope is determined by the segment's endpoints). We shall specify rasterization only for x-major segments except in cases where the modifications for y-major segments are not self-evident.

Ideally, the GL uses a "diamond-exit" rule to determine those fragments that are produced by rasterizing a line segment. For each fragment f with center at window coordinates x_f and y_f, define a diamond-shaped region that is the intersection of four half planes:

$$R_f = \{\, (x, y) \mid |x - x_f| + |y - y_f| < 1/2. \}$$

Essentially, a line segment starting at $\mathbf{p}_a$ and ending at $\mathbf{p}_b$ produces those fragments f for which the segment intersects R_f, except if $\mathbf{p}_b$ is contained in R_f. See figure 3.4.

To avoid difficulties when an endpoint lies on a boundary of R_f we (in principle) perturb the supplied endpoints by a tiny amount. Let $\mathbf{p}_a$ and $\mathbf{p}_b$ have window coordinates (x_a, y_a) and (x_b, y_b), respectively. Obtain the perturbed endpoints $\mathbf{p}'_a$ given by $(x_a, y_a) - (\epsilon, \epsilon^2)$ and $\mathbf{p}'_b$ given by $(x_b, y_b) - (\epsilon, \epsilon^2)$. Rasterizing the line segment starting at $\mathbf{p}_a$ and ending at $\mathbf{p}_b$ produces those fragments f for which the segment starting at $\mathbf{p}'_a$ and ending on $\mathbf{p}'_b$ intersects R_f, except if $\mathbf{p}'_b$ is contained in R_f. ϵ is chosen to be so small that rasterizing the line segment produces the same fragments when δ is substituted for ϵ for any $0 < \delta \leq \epsilon$.

When $\mathbf{p}_a$ and $\mathbf{p}_b$ lie on fragment centers, this characterization of fragments reduces to Bresenham's algorithm with one modification: lines produced in this description are "half-open," meaning that the final fragment (corresponding to $\mathbf{p}_b$) is not drawn. This means that when rasterizing a series of connected line segments, shared endpoints will be produced only once rather than twice (as would occur with Bresenham's algorithm).

Because the initial and final conditions of the diamond-exit rule may be difficult to implement, other line segment rasterization algorithms are allowed, subject to the following rules:

Microsoft Corp. Exhibit 1009

APPENDIX T

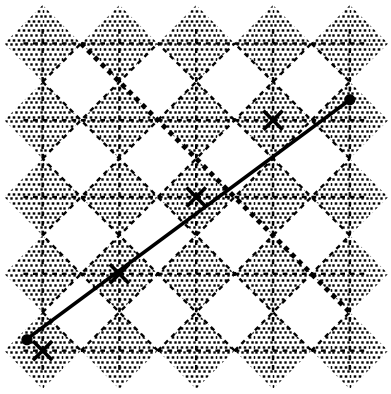


Figure 3.4. Visualization of Bresenham's algorithm. A portion of a line segment is shown. A diamond shaped region of height 1 is placed around each fragment center; those regions that the line segment exits cause rasterization to produce corresponding fragments.

1. The coordinates of a fragment produced by the algorithm may not deviate by more than one unit in either x or y window coordinates from a corresponding fragment produced by the diamond-exit rule.

2. The total number of fragments produced by the algorithm may differ from that produced by the diamond-exit rule by no more than one.

3. For an x-major line, no two fragments may be produced that lie in the same window-coordinate column (for a y-major line, no two fragments may appear in the same row).

4. If two line segments share a common endpoint, and both segments are either x-major (both left-to-right or both right-to-left) or y-major (both bottom-to-top or both top-to-bottom), then rasterizing both segments may not produce duplicate fragments, nor may any fragments be omitted so as to interrupt continuity of the connected segments.

Next we must specify how the data associated with each rasterized fragment are obtained. Let the window coordinates of a produced fragment center be given by $\mathbf{p}_r = (x_d, y_d)$ and let $\mathbf{p}_a = (x_a, y_a)$ and $\mathbf{p}_b = (x_b, y_b)$. Set

Microsoft Corp. Exhibit 1009

$$t = \frac{(\mathbf{p}_r - \mathbf{p}_a) \cdot (\mathbf{p}_b - \mathbf{p}_a)}{\|\mathbf{p}_b - \mathbf{p}_a\|^2}. \tag{3.1}$$

(Note that $t = 0$ at $\mathbf{p}_a$ and $t = 1$ at $\mathbf{p}_b$.) The value of an associated datum f for the fragment, whether it be R, G, B, or A (in RGBA mode) or a color index (in color index mode), or the s, t, or r texture coordinate (the depth value, window z, must be found using equation 3.3, below), is found as

$$f = \frac{(1-t)f_a/w_a + tf_b/w_b}{(1-t)\alpha_a/w_a + t\alpha_b/w_b} \tag{3.2}$$

where f_a and f_b are the data associated with the starting and ending endpoints of the segment, respectively; w_a and w_b are the clip w coordinates of the starting and ending endpoints of the segments, respectively. $\alpha_a = \alpha_b = 1$ for all data except texture coordinates, in which case $\alpha_a = q_a$ and $\alpha_b = q_b$ (q_a and q_b are the homogeneous texture coordinates at the starting and ending endpoints of the segment; results are undefined if either of these is less than or equal to 0). Note that linear interpolation would use

$$f = (1-t)f_a/\alpha_a + tf_b/\alpha_b. \tag{3.3}$$

The reason that this formula is incorrect (except for the depth value) is that it interpolates a datum in window space, which may be distorted by perspective. What is actually desired is to find the corresponding value when interpolated in clip space, which equation 3.2 does. A GL implementation may choose to approximate equation 3.2 with 3.3, but this will normally lead to unacceptable distortion effects when interpolating texture coordinates.

3.4.2 Other Line Segment Features

We have just described the rasterization of non-antialiased line segments of width one using the default line stipple of $FFFF_{16}$. We now describe the rasterization of line segments for general values of the line segment rasterization parameters.

Line Stipple

The command

```
void LineStipple( int factor, ushort pattern );
```


Microsoft Corp. Exhibit 1009

APPENDIX T

defines a *line stipple*. *pattern* is an unsigned short integer. The *line stipple* is taken from the lowest order 16 bits of *pattern*. It determines those fragments that are to be drawn when the line is rasterized. *factor* is a count that is used to modify the effective line stipple by causing each bit in *line stipple* to be used *factor* times. $factor$ is clamped to the range $[1, 256]$. Line stippling may be enabled or disabled using **Enable** or **Disable** with the constant `LINE_STIPPLE`. When disabled, it is as if the line stipple has its default value.

Line stippling masks certain fragments that are produced by rasterization so that they are not sent to the per-fragment stage of the GL. The masking is achieved using three parameters: the 16-bit line stipple p, the line repeat count r, and an integer stipple counter s. Let

$$b = \lfloor s/r \rfloor \bmod 16,$$

Then a fragment is produced if the bth bit of p is 1, and not produced otherwise. The bits of p are numbered with 0 being the least significant and 15 being the most significant. The initial value of s is zero; s is incremented after production of each fragment of a line segment (fragments are produced in order, beginning at the starting point and working towards the ending point). s is reset to 0 whenever a **Begin** occurs, and before every line segment in a group of independent segments (as specified when **Begin** is invoked with `LINES`).

If the line segment has been clipped, then the value of s at the beginning of the line segment is indeterminate.

Wide Lines

The actual width of non-antialiased lines is determined by rounding the supplied width to the nearest integer, then clamping it to the implementation-dependent maximum non-antialiased line width. This implementation-dependent value must be no less than the implementation-dependent maximum antialiased line width, rounded to the nearest integer value, and in any event no less than 1. If rounding the specified width results in the value 0, then it is as if the value were 1.

Non-antialiased line segments of width other than one are rasterized by offsetting them in the minor direction (for an x-major line, the minor direction is y, and for a y-major line, the minor direction is x) and replicating fragments in the minor direction (see figure 3.5). Let w be the width rounded to the nearest integer (if $w = 0$, then it is as if $w = 1$). If the line segment has endpoints given by (x_0, y_0) and (x_1, y_1) in window coordinates, the segment with endpoints $(x_0, y_0 - (w-1)/2)$ and $(x_1, y_1 - (w-1)/2)$ is rasterized, but

Microsoft Corp. Exhibit 1009

APPENDIX T

Figure 3.5. Rasterization of non-antialiased wide lines. x-major line segments are shown. The heavy line segment is the one specified to be rasterized; the light segment is the offset segment used for rasterization. x marks indicate the fragment centers produced by rasterization.

instead of a single fragment, a column of fragments of height w (a row of fragments of length w for a y-major segment) is produced at each x (y for y-major) location. The lowest fragment of this column is the fragment that would be produced by rasterizing the segment of width 1 with the modified coordinates. The whole column is not produced if the stipple bit for the column's x location is zero; otherwise, the whole column is produced.

Antialiasing

Rasterized antialiased line segments produce fragments whose fragment squares intersect a rectangle centered on the line segment. Two of the edges are parallel to the specified line segment; each is at a distance of one-half the current width from that segment: one above the segment and one below it. The other two edges pass through the line endpoints and are perpendicular to the direction of the specified line segment. Coverage values are computed for each fragment by computing the area of the intersection of the rectangle with the fragment square (see figure 3.6; see also section 3.2). Equation 3.2 is used to compute associated data values just as with non-antialiased lines; equation 3.1 is used to find the value of t for each fragment whose square is intersected by the line segment's rectangle. Not all widths need be sup-

Microsoft Corp. Exhibit 1009

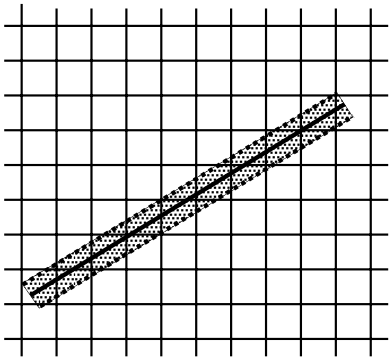


Figure 3.6. The region used in rasterizing and finding corresponding coverage values for an antialiased line segment (an x-major line segment is shown).

ported for line segment antialiasing, but width 1.0 antialiased segments must be provided. As with the point width, a GL implementation may be queried for the range and number of gradations of available antialiased line widths.

For purposes of antialiasing, a stippled line is considered to be a sequence of contiguous rectangles centered on the line segment. Each rectangle has width equal to the current line width and length equal to 1 pixel (except the last, which may be shorter). These rectangles are numbered from 0 to n, starting with the rectangle incident on the starting endpoint of the segment. Each of these rectangles is either eliminated or produced according to the procedure given under **Line Stipple**, above, where "fragment" is replaced with "rectangle." Each rectangle so produced is rasterized as if it were an antialiased polygon, described below (but culling, non-default settings of **PolygonMode**, and polygon stippling are not applied).

3.4.3 Line Rasterization State

The state required for line rasterization consists of the floating-point line width, a 16-bit line stipple, the line stipple repeat count, a bit indicating whether stippling is enabled or disabled, and a bit indicating whether line antialiasing is on or off. In addition, during rasterization, an integer stipple counter must be maintained to implement line stippling. The initial value of the line width is 1.0. The initial value of the line stipple is $FFFF_{16}$ (a stipple of all ones). The initial value of the line stipple repeat count is one.

Microsoft Corp. Exhibit 1009

APPENDIX T

The initial state of line stippling is disabled. The initial state of line segment antialiasing is disabled.

3.5 Polygons

A polygon results from a polygon **Begin**/**End** object, a triangle resulting from a triangle strip, triangle fan, or series of separate triangles, or a quadrilateral arising from a quadrilateral strip, series of separate quadrilaterals, or a **Rect** command. Like points and line segments, polygon rasterization is controlled by several variables. Polygon antialiasing is controlled with **Enable** and **Disable** with the symbolic constant `POLYGON_SMOOTH`. The analog to line segment stippling for polygons is polygon stippling, described below.

3.5.1 Basic Polygon Rasterization

The first step of polygon rasterization is to determine if the polygon is *back facing* or *front facing*. This determination is made by examining the sign of the area computed by equation 2.7 of section 2.13.1 (including the possible reversal of this sign as indicated by the last call to **FrontFace**). If this sign is positive, the polygon is frontfacing; otherwise, it is back facing. This determination is used in conjunction with the **CullFace** enable bit and mode value to decide whether or not a particular polygon is rasterized. The **CullFace** mode is set by calling

`void` **CullFace**(`enum` *mode*);

mode is a symbolic constant: one of `FRONT`, `BACK` or `FRONT_AND_BACK`. Culling is enabled or disabled with **Enable** or **Disable** using the symbolic constant `CULL_FACE`. Front facing polygons are rasterized if either culling is disabled or the **CullFace** mode is `BACK` while back facing polygons are rasterized only if either culling is disabled or the **CullFace** mode is `FRONT`. The initial setting of the **CullFace** mode is `BACK`. Initially, culling is disabled.

The rule for determining which fragments are produced by polygon rasterization is called *point sampling*. The two-dimensional projection obtained by taking the x and y window coordinates of the polygon's vertices is formed. Fragment centers that lie inside of this polygon are produced by rasterization. Special treatment is given to a fragment whose center lies on a polygon boundary edge. In such a case we require that if two polygons lie on either side of a common edge (with identical endpoints) on which a fragment center lies, then exactly one of the polygons results in the production of the fragment during rasterization.

Microsoft Corp. Exhibit 1009

As for the data associated with each fragment produced by rasterizing a polygon, we begin by specifying how these values are produced for fragments in a triangle. Define *barycentric coordinates* for a triangle. Barycentric coordinates are a set of three numbers, a, b, and c, each in the range $[0, 1]$, with $a + b + c = 1$. These coordinates uniquely specify any point p within the triangle or on the triangle's boundary as

$$p = ap_a + bp_b + cp_c,$$

where p_a, p_b, and p_c are the vertices of the triangle. a, b, and c can be found as

$$a = \frac{\mathrm{A}(pp_bp_c)}{\mathrm{A}(p_ap_bp_c)}, \quad b = \frac{\mathrm{A}(pp_ap_c)}{\mathrm{A}(p_ap_bp_c)}, \quad c = \frac{\mathrm{A}(pp_ap_b)}{\mathrm{A}(p_ap_bp_c)},$$

where $\mathrm{A}(lmn)$ denotes the area in window coordinates of the triangle with vertices l, m, and n.

Denote a datum at p_a, p_b, or p_c as f_a, f_b, or f_c, respectively. Then the value f of a datum at a fragment produced by rasterizing a triangle is given by

$$f = \frac{af_a/w_a + bf_b/w_b + cf_c/w_c}{a\alpha_a/w_a + b\alpha_b/w_b + c\alpha_c/w_c} \tag{3.4}$$

where w_a, w_b and w_c are the clip w coordinates of p_a, p_b, and p_c, respectively. a, b, and c are the barycentric coordinates of the fragment for which the data are produced. $\alpha_a = \alpha_b = \alpha_c = 1$ except for texture s, t, and r coordinates, for which $\alpha_a = q_a$, $\alpha_b = q_b$, and $\alpha_c = q_c$ (if any of q_a, q_b, or q_c are less than or equal to zero, results are undefined). a, b, and c must correspond precisely to the exact coordinates of the center of the fragment. Another way of saying this is that the data associated with a fragment must be sampled at the fragment's center.

Just as with line segment rasterization, equation 3.4 may be approximated by

$$f = af_a/\alpha_a + bf_b/\alpha_b + cf_c/\alpha_c;$$

this may yield acceptable results for color values (it *must* be used for depth values), but will normally lead to unacceptable distortion effects if used for texture coordinates.

For a polygon with more than three edges, we require only that a convex combination of the values of the datum at the polygon's vertices can be used to obtain the value assigned to each fragment produced by the rasterization

Microsoft Corp. Exhibit 1009

APPENDIX T

algorithm. That is, it must be the case that at every fragment

$$f = \sum_{i=1}^{n} a_i f_i$$

where n is the number of vertices in the polygon, f_i is the value of the f at vertex i; for each i $0 \leq a_i \leq 1$ and $\sum_{i=1}^{n} a_i = 1$. The values of the a_i may differ from fragment to fragment, but at vertex i, $a_j = 0, j \neq i$ and $a_i = 1$.

One algorithm that achieves the required behavior is to triangulate a polygon (without adding any vertices) and then treat each triangle individually as already discussed. A scan-line rasterizer that linearly interpolates data along each edge and then linearly interpolates data across each horizontal span from edge to edge also satisfies the restrictions (in this case, the numerator and denominator of equation 3.4 should be iterated independently and a division performed for each fragment).

3.5.2 Stippling

Polygon stippling works much the same way as line stippling, masking out certain fragments produced by rasterization so that they are not sent to the next stage of the GL. This is the case regardless of the state of polygon antialiasing. Stippling is controlled with

`void` **PolygonStipple**(`ubyte` **pattern*);

pattern is a pointer to memory into which a 32×32 pattern is packed. The pattern is unpacked from memory according to the procedure given in section 3.6.4 for **DrawPixels**; it is as if the *height* and *width* passed to that command were both equal to 32, the *type* were `BITMAP`, and the *format* were `COLOR_INDEX`. The unpacked values (before any conversion or arithmetic would have been performed) form a stipple pattern of zeros and ones.

If x_w and y_w are the window coordinates of a rasterized polygon fragment, then that fragment is sent to the next stage of the GL if and only if the bit of the pattern $(x_w \bmod 32, y_w \bmod 32)$ is 1.

Polygon stippling may be enabled or disabled with **Enable** or **Disable** using the constant `POLYGON_STIPPLE`. When disabled, it is as if the stipple pattern were all ones.

3.5.3 Antialiasing

Polygon antialiasing rasterizes a polygon by producing a fragment wherever the interior of the polygon intersects that fragment's square. A coverage

Microsoft Corp. Exhibit 1009

value is computed at each such fragment, and this value is saved to be applied as described in section 3.11. An associated datum is assigned to a fragment by integrating the datum's value over the region of the intersection of the fragment square with the polygon's interior and dividing this integrated value by the area of the intersection. For a fragment square lying entirely within the polygon, the value of a datum at the fragment's center may be used instead of integrating the value across the fragment.

Polygon stippling operates in the same way whether polygon antialiasing is enabled or not. The polygon point sampling rule defined in section 3.5.1, however, is not enforced for antialiased polygons.

3.5.4 Options Controlling Polygon Rasterization

The interpretation of polygons for rasterization is controlled using

`void` **PolygonMode**(`enum` *face*, `enum` *mode*);

face is one of `FRONT`, `BACK`, or `FRONT_AND_BACK`, indicating that the rasterizing method described by *mode* replaces the rasterizing method for front facing polygons, back facing polygons, or both front and back facing polygons, respectively. *mode* is one of the symbolic constants `POINT`, `LINE`, or `FILL`. Calling **PolygonMode** with `POINT` causes certain vertices of a polygon to be treated, for rasterization purposes, just as if they were enclosed within a **Begin**(`POINT`) and **End** pair. The vertices selected for this treatment are those that have been tagged as having a polygon boundary edge beginning on them (see section 2.6.2). `LINE` causes edges that are tagged as boundary to be rasterized as line segments. (The line stipple counter is reset at the beginning of the first rasterized edge of the polygon, but not for subsequent edges.) `FILL` is the default mode of polygon rasterization, corresponding to the description in sections 3.5.1, 3.5.2, and 3.5.3. Note that these modes affect only the final rasterization of polygons: in particular, a polygon's vertices are lit, and the polygon is clipped and possibly culled before these modes are applied.

Polygon antialiasing applies only to the `FILL` state of **PolygonMode**. For `POINT` or `LINE`, point antialiasing or line segment antialiasing, respectively, apply.

3.5.5 Depth Offset

The depth values of all fragments generated by the rasterization of a polygon may be offset by a single value that is computed for that polygon. The

Microsoft Corp. Exhibit 1009

function that determines this value is specified by calling

`void` **PolygonOffset**(`float` *factor*, `float` *units*);

factor scales the maximum depth slope of the polygon, and *units* scales an implementation dependent constant that relates to the usable resolution of the depth buffer. The resulting values are summed to produce the polygon offset value. Both *factor* and *units* may be either positive or negative.

The maximum depth slope m of a triangle is

$$m = \sqrt{\left(\frac{\partial z_w}{\partial x_w}\right)^2 + \left(\frac{\partial z_w}{\partial y_w}\right)^2} \tag{3.5}$$

where (x_w, y_w, z_w) is a point on the triangle. m may be approximated as

$$m = \max\left\{\left|\frac{\partial z_w}{\partial x_w}\right|, \left|\frac{\partial z_w}{\partial y_w}\right|\right\}. \tag{3.6}$$

If the polygon has more than three vertices, one or more values of m may be used during rasterization. Each may take any value in the range [min,max], where min and max are the smallest and largest values obtained by evaluating Equation 3.5 or Equation 3.6 for the triangles formed by all three-vertex combinations.

The minimum resolvable difference r is an implementation constant. It is the smallest difference in window coordinate z values that is guaranteed to remain distinct throughout polygon rasterization and in the depth buffer. All pairs of fragments generated by the rasterization of two polygons with otherwise identical vertices, but z_w values that differ by r, will have distinct depth values.

The offset value o for a polygon is

$$o = m * factor + r * units. \tag{3.7}$$

m is computed as described above, as a function of depth values in the range [0,1], and o is applied to depth values in the same range.

Boolean state values `POLYGON_OFFSET_POINT`, `POLYGON_OFFSET_LINE`, and `POLYGON_OFFSET_FILL` determine whether o is applied during the rasterization of polygons in `POINT`, `LINE`, and `FILL` modes. These boolean state values are enabled and disabled as argument values to the commands **Enable** and **Disable**. If `POLYGON_OFFSET_POINT` is enabled, o is added to the depth value of each fragment produced by the rasterization of a polygon in `POINT` mode. Likewise, if `POLYGON_OFFSET_LINE` or `POLYGON_OFFSET_FILL` is enabled, o

Microsoft Corp. Exhibit 1009

is added to the depth value of each fragment produced by the rasterization of a polygon in `LINE` or `FILL` modes, respectively.

Fragment depth values are always limited to the range [0,1], either by clamping after offset addition is performed (preferred), or by clamping the vertex values used in the rasterization of the polygon.

3.5.6 Polygon Rasterization State

The state required for polygon rasterization consists of a polygon stipple pattern, whether stippling is enabled or disabled, the current state of polygon antialiasing (enabled or disabled), the current values of the **PolygonMode** setting for each of front and back facing polygons, whether point, line, and fill mode polygon offsets are enabled or disabled, and the factor and bias values of the polygon offset equation. The initial stipple pattern is all ones; initially stippling is disabled. The initial setting of polygon antialiasing is disabled. The initial state for **PolygonMode** is `FILL` for both front and back facing polygons. The initial polygon offset factor and bias values are both 0; initially polygon offset is disabled for all modes.

3.6 Pixel Rectangles

Rectangles of color, depth, and certain other values may be converted to fragments using the **DrawPixels** command (described in section 3.6.4). Some of the parameters and operations governing the operation of **DrawPixels** are shared by **ReadPixels** (used to obtain pixel values from the framebuffer) and **CopyPixels** (used to copy pixels from one framebuffer location to another); the discussion of **ReadPixels** and **CopyPixels**, however, is deferred until Chapter 4 after the framebuffer has been discussed in detail. Nevertheless, we note in this section when parameters and state pertaining to **DrawPixels** also pertain to **ReadPixels** or **CopyPixels**.

A number of parameters control the encoding of pixels in client memory (for reading and writing) and how pixels are processed before being placed in or after being read from the framebuffer (for reading, writing, and copying). These parameters are set with three commands: **PixelStore**, **PixelTransfer**, and **PixelMap**.

3.6.1 Pixel Storage Modes

Pixel storage modes affect the operation of **DrawPixels** and **ReadPixels** (as well as other commands; see sections 3.5.2, 3.7, and 3.8) when one of

Microsoft Corp. Exhibit 1009

Parameter Name	Type	Initial Value	Valid Range
UNPACK_SWAP_BYTES	boolean	FALSE	TRUE/FALSE
UNPACK_LSB_FIRST	boolean	FALSE	TRUE/FALSE
UNPACK_ROW_LENGTH	integer	0	$[0, \infty)$
UNPACK_SKIP_ROWS	integer	0	$[0, \infty)$
UNPACK_SKIP_PIXELS	integer	0	$[0, \infty)$
UNPACK_ALIGNMENT	integer	4	1,2,4,8
UNPACK_IMAGE_HEIGHT	integer	0	$[0, \infty)$
UNPACK_SKIP_IMAGES	integer	0	$[0, \infty)$

Table 3.1: **PixelStore** parameters pertaining to one or more of **DrawPixels**, **TexImage1D**, **TexImage2D**, and **TexImage3D**.

these commands is issued. This may differ from the time that the command is executed if the command is placed in a display list (see section 5.4). Pixel storage modes are set with

`void` **PixelStore**{**if**}(`enum` *pname*, `T` *param*);

pname is a symbolic constant indicating a parameter to be set, and *param* is the value to set it to. Table 3.1 summarizes the pixel storage parameters, their types, their initial values, and their allowable ranges. Setting a parameter to a value outside the given range results in the error `INVALID_VALUE`.

The version of **PixelStore** that takes a floating-point value may be used to set any type of parameter; if the parameter is boolean, then it is set to `FALSE` if the passed value is 0.0 and `TRUE` otherwise, while if the parameter is an integer, then the passed value is rounded to the nearest integer. The integer version of the command may also be used to set any type of parameter; if the parameter is boolean, then it is set to `FALSE` if the passed value is 0 and `TRUE` otherwise, while if the parameter is a floating-point value, then the passed value is converted to floating-point.

3.6.2 The Imaging Subset

Some pixel transfer and per-fragment operations are only made available in GL implementations which incorporate the optional *imaging subset*. The imaging subset includes both new commands, and new enumerants allowed as parameters to existing commands. If the subset is supported, *all* of these

Microsoft Corp. Exhibit 1009

calls and enumerants must be implemented as described later in the GL specification. If the subset is not supported, calling any of the new commands generates the error INVALID_OPERATION, and using any of the new enumerants generates the error INVALID_ENUM.

The individual operations available only in the imaging subset are described in section 3.6.3, except for blending features, which are described in chapter 4. Imaging subset operations include:

1. Color tables, including all commands and enumerants described in subsections **Color Table Specification**, **Alternate Color Table Specification Commands**, **Color Table State and Proxy State**, **Color Table Lookup**, **Post Convolution Color Table Lookup**, and **Post Color Matrix Color Table Lookup**, as well as the query commands described in section 6.1.7.

2. Convolution, including all commands and enumerants described in subsections **Convolution Filter Specification**, **Alternate Convolution Filter Specification Commands**, and **Convolution**, as well as the query commands described in section 6.1.8.

3. Color matrix, including all commands and enumerants described in subsections **Color Matrix Specification** and **Color Matrix Transformation**, as well as the simple query commands described in section 6.1.6.

4. Histogram and minmax, including all commands and enumerants described in subsections **Histogram Table Specification**, **Histogram State and Proxy State**, **Histogram**, **Minmax Table Specification**, and **Minmax**, as well as the query commands described in section 6.1.9 and section 6.1.10.

5. The subset of blending features described by **BlendEquation**, **BlendColor**, and the **BlendFunc** *modes* CONSTANT_COLOR, ONE_MINUS_CONSTANT_COLOR, CONSTANT_ALPHA, and ONE_MINUS_CONSTANT_ALPHA. These are described separately in section 4.1.6.

The imaging subset is supported only if the EXTENSIONS string includes the substring "ARB_imaging". Querying EXTENSIONS is described in section 6.1.11.

If the imaging subset is not supported, the related pixel transfer operations are not performed; pixels are passed unchanged to the next operation.

Microsoft Corp. Exhibit 1009

Parameter Name	Type	Initial Value	Valid Range
MAP_COLOR	boolean	FALSE	TRUE/FALSE
MAP_STENCIL	boolean	FALSE	TRUE/FALSE
INDEX_SHIFT	integer	0	$(-\infty, \infty)$
INDEX_OFFSET	integer	0	$(-\infty, \infty)$
x_SCALE	float	1.0	$(-\infty, \infty)$
DEPTH_SCALE	float	1.0	$(-\infty, \infty)$
x_BIAS	float	0.0	$(-\infty, \infty)$
DEPTH_BIAS	float	0.0	$(-\infty, \infty)$
POST_CONVOLUTION_x_SCALE	float	1.0	$(-\infty, \infty)$
POST_CONVOLUTION_x_BIAS	float	0.0	$(-\infty, \infty)$
POST_COLOR_MATRIX_x_SCALE	float	1.0	$(-\infty, \infty)$
POST_COLOR_MATRIX_x_BIAS	float	0.0	$(-\infty, \infty)$

Table 3.2: **PixelTransfer** parameters. x is RED, GREEN, BLUE, or ALPHA.

3.6.3 Pixel Transfer Modes

Pixel transfer modes affect the operation of **DrawPixels** (section 3.6.4), **ReadPixels** (section 4.3.2), and **CopyPixels** (section 4.3.3) at the time when one of these commands is executed (which may differ from the time the command is issued). Some pixel transfer modes are set with

`void` **PixelTransfer{if}**(`enum` *param*, `T` *value*);

param is a symbolic constant indicating a parameter to be set, and *value* is the value to set it to. Table 3.2 summarizes the pixel transfer parameters that are set with **PixelTransfer**, their types, their initial values, and their allowable ranges. Setting a parameter to a value outside the given range results in the error INVALID_VALUE. The same versions of the command exist as for **PixelStore**, and the same rules apply to accepting and converting passed values to set parameters.

The pixel map lookup tables are set with

`void` **PixelMap{ui us f}v**(`enum` *map*, `sizei` *size*, `T` *values*);

map is a symbolic map name, indicating the map to set, *size* indicates the size of the map, and *values* is a pointer to an array of *size* map values.

Microsoft Corp. Exhibit 1009

Map Name	Address	Value	Init. Size	Init. Value
PIXEL_MAP_I_TO_I	color idx	color idx	1	0.0
PIXEL_MAP_S_TO_S	stencil idx	stencil idx	1	0
PIXEL_MAP_I_TO_R	color idx	R	1	0.0
PIXEL_MAP_I_TO_G	color idx	G	1	0.0
PIXEL_MAP_I_TO_B	color idx	B	1	0.0
PIXEL_MAP_I_TO_A	color idx	A	1	0.0
PIXEL_MAP_R_TO_R	R	R	1	0.0
PIXEL_MAP_G_TO_G	G	G	1	0.0
PIXEL_MAP_B_TO_B	B	B	1	0.0
PIXEL_MAP_A_TO_A	A	A	1	0.0

Table 3.3: **PixelMap** parameters.

The entries of a table may be specified using one of three types: single-precision floating-point, unsigned short integer, or unsigned integer, depending on which of the three versions of **PixelMap** is called. A table entry is converted to the appropriate type when it is specified. An entry giving a color component value is converted according to table 2.6. An entry giving a color index value is converted from an unsigned short integer or unsigned integer to floating-point. An entry giving a stencil index is converted from single-precision floating-point to an integer by rounding to nearest. The various tables and their initial sizes and entries are summarized in table 3.3. A table that takes an index as an address must have $size = 2^n$ or the error INVALID_VALUE results. The maximum allowable *size* of each table is specified by the implementation dependent value MAX_PIXEL_MAP_TABLE, but must be at least 32 (a single maximum applies to all tables). The error INVALID_VALUE is generated if a *size* larger than the implemented maximum, or less than one, is given to **PixelMap**.

Color Table Specification

Color lookup tables are specified with

```
void ColorTable( enum target, enum internalformat,
    sizei width, enum format, enum type, void *data );
```

target must be one of the *regular* color table names listed in table 3.4 to define the table. A *proxy* table name is a special case discussed later in

Microsoft Corp. Exhibit 1009

Table Name	Type
COLOR_TABLE POST_CONVOLUTION_COLOR_TABLE POST_COLOR_MATRIX_COLOR_TABLE	regular
PROXY_COLOR_TABLE PROXY_POST_CONVOLUTION_COLOR_TABLE PROXY_POST_COLOR_MATRIX_COLOR_TABLE	proxy

Table 3.4: Color table names. Regular tables have associated image data. Proxy tables have no image data, and are used only to determine if an image can be loaded into the corresponding regular table.

this section. *width*, *format*, *type*, and *data* specify an image in memory with the same meaning and allowed values as the corresponding arguments to **DrawPixels** (see section 3.6.4), with *height* taken to be 1. The maximum allowable *width* of a table is implementation-dependent, but must be at least 32. The *format*s COLOR_INDEX, DEPTH_COMPONENT, and STENCIL_INDEX and the *type* BITMAP are not allowed.

The specified image is taken from memory and processed just as if **DrawPixels** were called, stopping after the final expansion to RGBA. The R, G, B, and A components of each pixel are then scaled by the four COLOR_TABLE_SCALE parameters, biased by the four COLOR_TABLE_BIAS parameters, and clamped to $[0, 1]$. These parameters are set by calling **ColorTableParameterfv** as described below.

Components are then selected from the resulting R, G, B, and A values to obtain a table with the *base internal format* specified by (or derived from) *internalformat*, in the same manner as for textures (section 3.8.1). *internalformat* must be one of the formats in table 3.15 or table 3.16.

The color lookup table is redefined to have *width* entries, each with the specified internal format. The table is formed with indices 0 through $width - 1$. Table location i is specified by the ith image pixel, counting from zero.

The error INVALID_VALUE is generated if *width* is not zero or a non-negative power of two. The error TABLE_TOO_LARGE is generated if the specified color lookup table is too large for the implementation.

The scale and bias parameters for a table are specified by calling

```
void ColorTableParameter{if}v( enum target,
    enum pname, T params );
```


Microsoft Corp. Exhibit 1009

target must be a regular color table name. *pname* is one of `COLOR_TABLE_SCALE` or `COLOR_TABLE_BIAS`. *params* points to an array of four values: red, green, blue, and alpha, in that order.

A GL implementation may vary its allocation of internal component resolution based on any **ColorTable** parameter, but the allocation must not be a function of any other factor, and cannot be changed once it is established. Allocations must be invariant; the same allocation must be made each time a color table is specified with the same parameter values. These allocation rules also apply to proxy color tables, which are described later in this section.

Alternate Color Table Specification Commands

Color tables may also be specified using image data taken directly from the framebuffer, and portions of existing tables may be respecified.

The command

```
void CopyColorTable( enum target, enum internalformat,
    int x, int y, sizei width );
```

defines a color table in exactly the manner of **ColorTable**, except that table data are taken from the framebuffer, rather than from client memory. *target* must be a regular color table name. x, y, and *width* correspond precisely to the corresponding arguments of **CopyPixels** (refer to section 4.3.3); they specify the image's *width* and the lower left (x, y) coordinates of the framebuffer region to be copied. The image is taken from the framebuffer exactly as if these arguments were passed to **CopyPixels** with argument *type* set to `COLOR` and *height* set to 1, stopping after the final expansion to RGBA.

Subsequent processing is identical to that described for **ColorTable**, beginning with scaling by `COLOR_TABLE_SCALE`. Parameters *target*, *internalformat* and *width* are specified using the same values, with the same meanings, as the equivalent arguments of **ColorTable**. *format* is taken to be `RGBA`.

Two additional commands,

```
void ColorSubTable( enum target, sizei start,
    sizei count, enum format, enum type, void *data );
void CopyColorSubTable( enum target, sizei start,
    int x, int y, sizei count );
```

respecify only a portion of an existing color table. No change is made to the *internalformat* or *width* parameters of the specified color table, nor is any

Microsoft Corp. Exhibit 1009

change made to table entries outside the specified portion. *target* must be a regular color table name.

ColorSubTable arguments *format*, *type*, and *data* match the corresponding arguments to **ColorTable**, meaning that they are specified using the same values, and have the same meanings. Likewise, **CopyColorSubTable** arguments x, y, and *count* match the x, y, and *width* arguments of **CopyColorTable**. Both of the **ColorSubTable** commands interpret and process pixel groups in exactly the manner of their **ColorTable** counterparts, except that the assignment of R, G, B, and A pixel group values to the color table components is controlled by the *internalformat* of the table, not by an argument to the command.

Arguments *start* and *count* of **ColorSubTable** and **CopyColorSubTable** specify a subregion of the color table starting at index *start* and ending at index $start + count - 1$. Counting from zero, the nth pixel group is assigned to the table entry with index $count + n$. The error `INVALID_VALUE` is generated if $start + count > width$.

Color Table State and Proxy State

The state necessary for color tables can be divided into two categories. For each of the three tables, there is an array of values. Each array has associated with it a width, an integer describing the internal format of the table, six integer values describing the resolutions of each of the red, green, blue, alpha, luminance, and intensity components of the table, and two groups of four floating-point numbers to store the table scale and bias. Each initial array is null (zero width, internal format `RGBA`, with zero-sized components). The initial value of the scale parameters is (1,1,1,1) and the initial value of the bias parameters is (0,0,0,0).

In addition to the color lookup tables, partially instantiated proxy color lookup tables are maintained. Each proxy table includes width and internal format state values, as well as state for the red, green, blue, alpha, luminance, and intensity component resolutions. Proxy tables do not include image data, nor do they include scale and bias parameters. When **ColorTable** is executed with *target* specified as one of the proxy color table names listed in table 3.4, the proxy state values of the table are recomputed and updated. If the table is too large, no error is generated, but the proxy format, width and component resolutions are set to zero. If the color table would be accommodated by **ColorTable** called with *target* set to the corresponding regular table name (`COLOR_TABLE` is the regular name corresponding to `PROXY_COLOR_TABLE`, for example), the proxy state values are set exactly as

Microsoft Corp. Exhibit 1009

though the regular table were being specified. Calling **ColorTable** with a proxy *target* has no effect on the image or state of any actual color table.

There is no image associated with any of the proxy targets. They cannot be used as color tables, and they must never be queried using **GetColorTable**. The error INVALID_ENUM is generated if this is attempted.

Convolution Filter Specification

A two-dimensional convolution filter image is specified by calling

```
void ConvolutionFilter2D( enum target,
    enum internalformat, sizei width, sizei height,
    enum format, enum type, void *data );
```

target must be CONVOLUTION_2D. *width*, *height*, *format*, *type*, and *data* specify an image in memory with the same meaning and allowed values as the corresponding parameters to **DrawPixels**. The *format*s COLOR_INDEX, DEPTH_COMPONENT, and STENCIL_INDEX and the *type* BITMAP are not allowed.

The specified image is extracted from memory and processed just as if **DrawPixels** were called, stopping after the final expansion to RGBA. The R, G, B, and A components of each pixel are then scaled by the four two-dimensional CONVOLUTION_FILTER_SCALE parameters and biased by the four two-dimensional CONVOLUTION_FILTER_BIAS parameters. These parameters are set by calling **ConvolutionParameterfv** as described below. No clamping takes place at any time during this process.

Components are then selected from the resulting R, G, B, and A values to obtain a table with the *base internal format* specified by (or derived from) *internalformat*, in the same manner as for textures (section 3.8.1). *internalformat* must be one of the formats in table 3.15 or table 3.16.

The red, green, blue, alpha, luminance, and/or intensity components of the pixels are stored in floating point, rather than integer format. They form a two-dimensional image indexed with coordinates i, j such that i increases from left to right, starting at zero, and j increases from bottom to top, also starting at zero. Image location i, j is specified by the Nth pixel, counting from zero, where

$$N = i + j * width$$

The error INVALID_VALUE is generated if *width* or *height* is greater than the maximum supported value. These values are queried with **GetConvolutionParameteriv**, setting *target* to CONVOLUTION_2D and *pname* to MAX_CONVOLUTION_WIDTH or MAX_CONVOLUTION_HEIGHT, respectively.

Microsoft Corp. Exhibit 1009

The scale and bias parameters for a two-dimensional filter are specified by calling

```
void ConvolutionParameter{if}v( enum target,
    enum pname, T params );
```

with *target* CONVOLUTION_2D. *pname* is one of CONVOLUTION_FILTER_SCALE or CONVOLUTION_FILTER_BIAS. *params* points to an array of four values: red, green, blue, and alpha, in that order.

A one-dimensional convolution filter is defined using

```
void ConvolutionFilter1D( enum target,
    enum internalformat, sizei width, enum format,
    enum type, void *data );
```

target must be CONVOLUTION_1D. *internalformat*, *width*, *format*, and *type* have identical semantics and accept the same values as do their two-dimensional counterparts. *data* must point to a one-dimensional image, however.

The image is extracted from memory and processed as if **ConvolutionFilter2D** were called with a *height* of 1, except that it is scaled and biased by the one-dimensional CONVOLUTION_FILTER_SCALE and CONVOLUTION_FILTER_BIAS parameters. These parameters are specified exactly as the two-dimensional parameters, except that **ConvolutionParameterfv** is called with *target* CONVOLUTION_1D.

The image is formed with coordinates i such that i increases from left to right, starting at zero. Image location i is specified by the ith pixel, counting from zero.

The error INVALID_VALUE is generated if *width* is greater than the maximum supported value. This value is queried using **GetConvolutionParameteriv**, setting *target* to CONVOLUTION_1D and *pname* to MAX_CONVOLUTION_WIDTH.

Special facilities are provided for the definition of two-dimensional *separable* filters – filters whose image can be represented as the product of two one-dimensional images, rather than as full two-dimensional images. A two-dimensional separable convolution filter is specified with

```
void SeparableFilter2D( enum target, enum internalformat,
    sizei width, sizei height, enum format, enum type,
    void *row, void *column );
```


Microsoft Corp. Exhibit 1009

target must be SEPARABLE_2D. *internalformat* specifies the formats of the table entries of the two one-dimensional images that will be retained. *row* points to a *width* pixel wide image of the specified *format* and *type*. *column* points to a *height* pixel high image, also of the specified *format* and *type*.

The two images are extracted from memory and processed as if **ConvolutionFilter1D** were called separately for each, except that each image is scaled and biased by the two-dimensional separable CONVOLUTION_FILTER_SCALE and CONVOLUTION_FILTER_BIAS parameters. These parameters are specified exactly as the one-dimensional and two-dimensional parameters, except that **ConvolutionParameteriv** is called with *target* SEPARABLE_2D.

Alternate Convolution Filter Specification Commands

One and two-dimensional filters may also be specified using image data taken directly from the framebuffer.

The command

```
void CopyConvolutionFilter2D( enum target,
    enum internalformat, int x, int y, sizei width,
    sizei height );
```

defines a two-dimensional filter in exactly the manner of **ConvolutionFilter2D**, except that image data are taken from the framebuffer, rather than from client memory. *target* must be CONVOLUTION_2D. x, y, *width*, and *height* correspond precisely to the corresponding arguments of **CopyPixels** (refer to section 4.3.3); they specify the image's *width* and *height*, and the lower left (x, y) coordinates of the framebuffer region to be copied. The image is taken from the framebuffer exactly as if these arguments were passed to **CopyPixels** with argument *type* set to COLOR, stopping after the final expansion to RGBA.

Subsequent processing is identical to that described for **ConvolutionFilter2D**, beginning with scaling by CONVOLUTION_FILTER_SCALE. Parameters *target*, *internalformat*, *width*, and *height* are specified using the same values, with the same meanings, as the equivalent arguments of **ConvolutionFilter2D**. *format* is taken to be RGBA.

The command

```
void CopyConvolutionFilter1D( enum target,
    enum internalformat, int x, int y, sizei width );
```


Microsoft Corp. Exhibit 1009

defines a one-dimensional filter in exactly the manner of **ConvolutionFilter1D**, except that image data are taken from the framebuffer, rather than from client memory. *target* must be `CONVOLUTION_1D`. x, y, and *width* correspond precisely to the corresponding arguments of **CopyPixels** (refer to section 4.3.3); they specify the image's *width* and the lower left (x, y) coordinates of the framebuffer region to be copied. The image is taken from the framebuffer exactly as if these arguments were passed to **CopyPixels** with argument *type* set to `COLOR` and *height* set to 1, stopping after the final expansion to RGBA.

Subsequent processing is identical to that described for **ConvolutionFilter1D**, beginning with scaling by `CONVOLUTION_FILTER_SCALE`. Parameters *target*, *internalformat*, and *width* are specified using the same values, with the same meanings, as the equivalent arguments of **ConvolutionFilter2D**. *format* is taken to be `RGBA`.

Convolution Filter State

The required state for convolution filters includes a one-dimensional image array, two one-dimensional image arrays for the separable filter, and a two-dimensional image array. The two-dimensional array has associated with it a height. Each array has associated with it a width, an integer describing the internal format of the table, and six integer values describing the resolutions of each of the red, green, blue, alpha, luminance, and intensity components of the table. Each filter (one-dimensional, two-dimensional, and two-dimensional separable) also has associated with it two groups of four floating-point numbers to store the filter scale and bias.

Each initial convolution filter is null (zero width and height, internal format RGBA, with zero-sized components). The initial value of all scale parameters is (1,1,1,1) and the initial value of all bias parameters is (0,0,0,0).

Color Matrix Specification

Setting the matrix mode to `COLOR` causes the matrix operations described in section 2.10.2 to apply to the top matrix on the color matrix stack. All matrix operations have the same effect on the color matrix as they do on the other matrices.

Histogram Table Specification

The histogram table is specified with

Microsoft Corp. Exhibit 1009

```
void Histogram( enum target, sizei width,
    enum internalformat, boolean sink );
```

target must be HISTOGRAM if a histogram table is to be specified. *target* value PROXY_HISTOGRAM is a special case discussed later in this section. *width* specifies the number of entries in the histogram table, and *internalformat* specifies the format of each table entry. The maximum allowable *width* of the histogram table is implementation-dependent, but must be at least 32. *sink* specifies whether pixel groups will be consumed by the histogram operation (TRUE) or passed on to the minmax operation (FALSE).

If no error results from the execution of **Histogram**, the specified histogram table is redefined to have *width* entries, each with the specified internal format. The entries are indexed 0 through $width - 1$. Each component in each entry is set to zero. The values in the previous histogram table, if any, are lost.

The error INVALID_VALUE is generated if *width* is not zero or a non-negative power of 2. The error TABLE_TOO_LARGE is generated if the specified histogram table is too large for the implementation. The error INVALID_ENUM is generated if *internalformat* is not one of the values accepted by the corresponding parameter of **TexImage2D**, or is 1, 2, 3, 4, INTENSITY, INTENSITY4, INTENSITY8, INTENSITY12, or INTENSITY16.

A GL implementation may vary its allocation of internal component resolution based on any **Histogram** parameter, but the allocation must not be a function of any other factor, and cannot be changed once it is established. In particular, allocations must be invariant; the same allocation must be made each time a histogram is specified with the same parameter values. These allocation rules also apply to the proxy histogram, which is described later in this section.

Histogram State and Proxy State

The state necessary for histogram operation is an array of values, with which is associated a width, an integer describing the internal format of the histogram, five integer values describing the resolutions of each of the red, green, blue, alpha, and luminance components of the table, and a flag indicating whether or not pixel groups are consumed by the operation. The initial array is null (zero width, internal format RGBA, with zero-sized components). The initial value of the flag is false.

In addition to the histogram table, a partially instantiated proxy histogram table is maintained. It includes width, internal format, and red,

Microsoft Corp. Exhibit 1009

green, blue, alpha, and luminance component resolutions. The proxy table does not include image data or the flag. When **Histogram** is executed with *target* set to PROXY_HISTOGRAM, the proxy state values are recomputed and updated. If the histogram array is too large, no error is generated, but the proxy format, width, and component resolutions are set to zero. If the histogram table would be accomodated by **Histogram** called with *target* set to HISTOGRAM, the proxy state values are set exactly as though the actual histogram table were being specified. Calling **Histogram** with *target* PROXY_HISTOGRAM has no effect on the actual histogram table.

There is no image associated with PROXY_HISTOGRAM. It cannot be used as a histogram, and its image must never queried using **GetHistogram**. The error INVALID_ENUM results if this is attempted.

Minmax Table Specification

The minmax table is specified with

```
void Minmax( enum target, enum internalformat,
    boolean sink );
```

target must be MINMAX. *internalformat* specifies the format of the table entries. *sink* specifies whether pixel groups will be consumed by the minmax operation (TRUE) or passed on to final conversion (FALSE).

The error INVALID_ENUM is generated if *internalformat* is not one of the values accepted by the corresponding parameter of **TexImage2D**, or is 1, 2, 3, 4, INTENSITY, INTENSITY4, INTENSITY8, INTENSITY12, or INTENSITY16. The resulting table always has 2 entries, each with values corresponding only to the components of the internal format.

The state necessary for minmax operation is a table containing two elements (the first element stores the minimum values, the second stores the maximum values), an integer describing the internal format of the table, and a flag indicating whether or not pixel groups are consumed by the operation. The initial state is a minimum table entry set to the maximum representable value and a maximum table entry set to the minimum representable value. Internal format is set to RGBA and the initial value of the flag is false.

3.6.4 Rasterization of Pixel Rectangles

The process of drawing pixels encoded in host memory is diagrammed in figure 3.7. We describe the stages of this process in the order in which they occur.

Microsoft Corp. Exhibit 1009

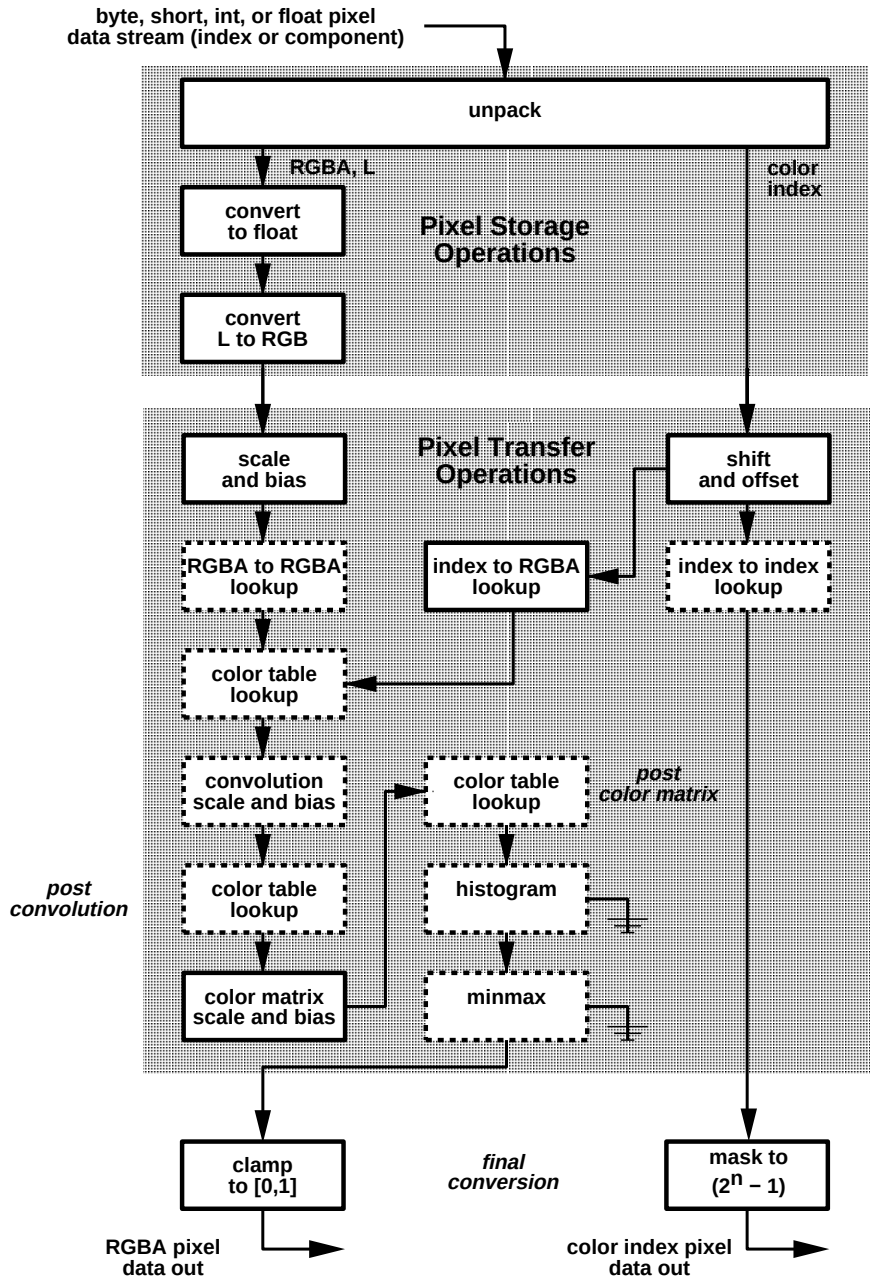


Figure 3.7. Operation of **DrawPixels**. Output is RGBA pixels if the GL is in RGBA mode, color index pixels otherwise. Operations in dashed boxes may be enabled or disabled. RGBA and color index pixel paths are shown; depth and stencil pixel paths are not shown.

Microsoft Corp. Exhibit 1009

Pixels are drawn using

```
void DrawPixels( sizei width, sizei height, enum format,
    enum type, void *data );
```

format is a symbolic constant indicating what the values in memory represent. *width* and *height* are the width and height, respectively, of the pixel rectangle to be drawn. *data* is a pointer to the data to be drawn. These data are represented with one of seven GL data types, specified by *type*. The correspondence between the twenty *type* token values and the GL data types they indicate is given in table 3.5. If the GL is in color index mode and *format* is not one of `COLOR_INDEX`, `STENCIL_INDEX`, or `DEPTH_COMPONENT`, then the error `INVALID_OPERATION` occurs. If *type* is `BITMAP` and *format* is not `COLOR_INDEX` or `STENCIL_INDEX` then the error `INVALID_ENUM` occurs. Some additional constraints on the combinations of *format* and *type* values that are accepted is discussed below.

Unpacking

Data are taken from host memory as a sequence of signed or unsigned bytes (GL data types `byte` and `ubyte`), signed or unsigned short integers (GL data types `short` and `ushort`), signed or unsigned integers (GL data types `int` and `uint`), or floating point values (GL data type `float`). These elements are grouped into sets of one, two, three, or four values, depending on the *format*, to form a group. Table 3.6 summarizes the format of groups obtained from memory; it also indicates those formats that yield indices and those that yield components.

By default the values of each GL data type are interpreted as they would be specified in the language of the client's GL binding. If `UNPACK_SWAP_BYTES` is enabled, however, then the values are interpreted with the bit orderings modified as per table 3.7. The modified bit orderings are defined only if the GL data type `ubyte` has eight bits, and then for each specific GL data type only if that type is represented with 8, 16, or 32 bits.

The groups in memory are treated as being arranged in a rectangle. This rectangle consists of a series of *rows*, with the first element of the first group of the first row pointed to by the pointer passed to **DrawPixels**. If the value of `UNPACK_ROW_LENGTH` is not positive, then the number of groups in a row is *width*; otherwise the number of groups is `UNPACK_ROW_LENGTH`. If p indicates the location in memory of the first element of the first row, then the first element of the Nth row is indicated by

Microsoft Corp. Exhibit 1009

type Parameter Token Name	Corresponding GL Data Type	Special Interpretation
UNSIGNED_BYTE	ubyte	No
BITMAP	ubyte	Yes
BYTE	byte	No
UNSIGNED_SHORT	ushort	No
SHORT	short	No
UNSIGNED_INT	uint	No
INT	int	No
FLOAT	float	No
UNSIGNED_BYTE_3_3_2	ubyte	Yes
UNSIGNED_BYTE_2_3_3_REV	ubyte	Yes
UNSIGNED_SHORT_5_6_5	ushort	Yes
UNSIGNED_SHORT_5_6_5_REV	ushort	Yes
UNSIGNED_SHORT_4_4_4_4	ushort	Yes
UNSIGNED_SHORT_4_4_4_4_REV	ushort	Yes
UNSIGNED_SHORT_5_5_5_1	ushort	Yes
UNSIGNED_SHORT_1_5_5_5_REV	ushort	Yes
UNSIGNED_INT_8_8_8_8	uint	Yes
UNSIGNED_INT_8_8_8_8_REV	uint	Yes
UNSIGNED_INT_10_10_10_2	uint	Yes
UNSIGNED_INT_2_10_10_10_REV	uint	Yes

Table 3.5: **DrawPixels** and **ReadPixels** *type* parameter values and the corresponding GL data types. Refer to table 2.2 for definitions of GL data types. Special interpretations are described near the end of section 3.6.4.

Microsoft Corp. Exhibit 1009

APPENDIX T

Format Name	Element Meaning and Order	Target Buffer
COLOR_INDEX	Color Index	Color
STENCIL_INDEX	Stencil Index	Stencil
DEPTH_COMPONENT	Depth	Depth
RED	R	Color
GREEN	G	Color
BLUE	B	Color
ALPHA	A	Color
RGB	R, G, B	Color
RGBA	R, G, B, A	Color
BGR	B, G, R	Color
BGRA	B, G, R, A	Color
LUMINANCE	Luminance	Color
LUMINANCE_ALPHA	Luminance, A	Color

Table 3.6: **DrawPixels** and **ReadPixels** formats. The second column gives a description of and the number and order of elements in a group. Unless specified as an index, formats yield components.

Element Size	Default Bit Ordering	Modified Bit Ordering
8 bit	[7..0]	[7..0]
16 bit	[15..0]	[7..0][15..8]
32 bit	[31..0]	[7..0][15..8][23..16][31..24]

Table 3.7: Bit ordering modification of elements when UNPACK_SWAP_BYTES is enabled. These reorderings are defined only when GL data type ubyte has 8 bits, and then only for GL data types with 8, 16, or 32 bits. Bit 0 is the least significant.

Microsoft Corp. Exhibit 1009

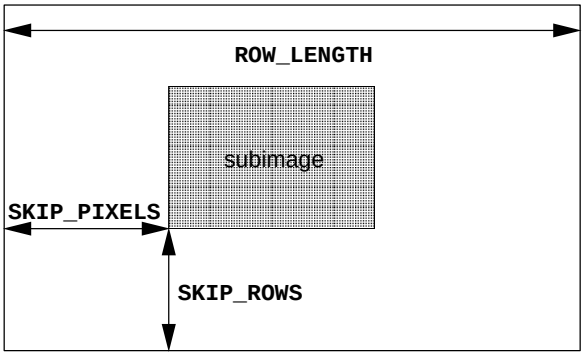


Figure 3.8. Selecting a subimage from an image. The indicated parameter names are prefixed by `UNPACK_` for **DrawPixels** and by `PACK_` for **ReadPixels**.

$$p + Nk \tag{3.8}$$

where N is the row number (counting from zero) and k is defined as

$$k = \begin{cases} nl & s \geq a, \\ a/s \lceil snl/a \rceil & s < a \end{cases} \tag{3.9}$$

where n is the number of elements in a group, l is the number of groups in the row, a is the value of `UNPACK_ALIGNMENT`, and s is the size, in units of GL `ubytes`, of an element. If the number of bits per element is not 1, 2, 4, or 8 times the number of bits in a GL `ubyte`, then $k = nl$ for all values of a.

There is a mechanism for selecting a sub-rectangle of groups from a larger containing rectangle. This mechanism relies on three integer parameters: `UNPACK_ROW_LENGTH`, `UNPACK_SKIP_ROWS`, and `UNPACK_SKIP_PIXELS`. Before obtaining the first group from memory, the pointer supplied to **DrawPixels** is effectively advanced by (`UNPACK_SKIP_PIXELS`)n + (`UNPACK_SKIP_ROWS`)k elements. Then *width* groups are obtained from contiguous elements in memory (without advancing the pointer), after which the pointer is advanced by k elements. *height* sets of *width* groups of values are obtained this way. See figure 3.8.

Calling **DrawPixels** with a *type* of `UNSIGNED_BYTE_3_3_2`, `UNSIGNED_BYTE_2_3_3_REV`, `UNSIGNED_SHORT_5_6_5`, `UNSIGNED_SHORT_5_6_5_REV`,

Microsoft Corp. Exhibit 1009

type Parameter Token Name	GL Data Type	Number of Components	Matching Pixel Formats
UNSIGNED_BYTE_3_3_2	ubyte	3	RGB
UNSIGNED_BYTE_2_3_3_REV	ubyte	3	RGB
UNSIGNED_SHORT_5_6_5	ushort	3	RGB
UNSIGNED_SHORT_5_6_5_REV	ushort	3	RGB
UNSIGNED_SHORT_4_4_4_4	ushort	4	RGBA,BGRA
UNSIGNED_SHORT_4_4_4_4_REV	ushort	4	RGBA,BGRA
UNSIGNED_SHORT_5_5_5_1	ushort	4	RGBA,BGRA
UNSIGNED_SHORT_1_5_5_5_REV	ushort	4	RGBA,BGRA
UNSIGNED_INT_8_8_8_8	uint	4	RGBA,BGRA
UNSIGNED_INT_8_8_8_8_REV	uint	4	RGBA,BGRA
UNSIGNED_INT_10_10_10_2	uint	4	RGBA,BGRA
UNSIGNED_INT_2_10_10_10_REV	uint	4	RGBA,BGRA

Table 3.8: Packed pixel formats.

UNSIGNED_SHORT_4_4_4_4, UNSIGNED_SHORT_4_4_4_4_REV, UNSIGNED_SHORT_5_5_5_1, UNSIGNED_SHORT_1_5_5_5_REV, UNSIGNED_INT_8_8_8_8, UNSIGNED_INT_8_8_8_8_REV, UNSIGNED_INT_10_10_10_2, or UNSIGNED_INT_2_10_10_10_REV is a special case in which all the components of each group are packed into a single unsigned byte, unsigned short, or unsigned int, depending on the type. The number of components per packed pixel is fixed by the type, and must match the number of components per group indicated by the *format* parameter, as listed in table 3.8. The error INVALID_OPERATION is generated if a mismatch occurs. This constraint also holds for all other functions that accept or return pixel data using *type* and *format* parameters to define the type and format of that data.

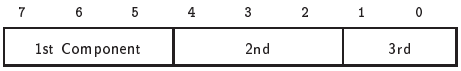
Bitfield locations of the first, second, third, and fourth components of each packed pixel type are illustrated in tables 3.9, 3.10, and 3.11. Each bitfield is interpreted as an unsigned integer value. If the base GL type is supported with more than the minimum precision (e.g. a 9-bit byte) the packed components are right-justified in the pixel.

Components are normally packed with the first component in the most significant bits of the bitfield, and successive component occupying progressively less significant locations. Types whose token names end with _REV reverse the component packing order from least to most significant locations. In all cases, the most significant bit of each component is packed in

Microsoft Corp. Exhibit 1009

the most significant bit location of its location in the bitfield.

`UNSIGNED_BYTE_3_3_2`:



`UNSIGNED_BYTE_2_3_3_REV`:

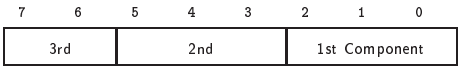
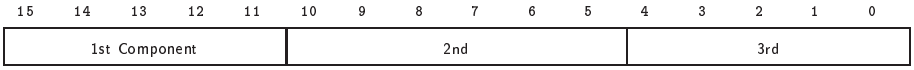


Table 3.9: `UNSIGNED_BYTE` formats. Bit numbers are indicated for each component.

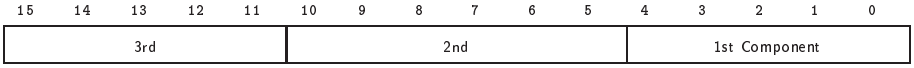
Microsoft Corp. Exhibit 1009

APPENDIX T

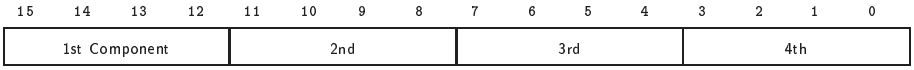
UNSIGNED_SHORT_5_6_5:



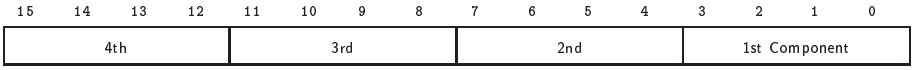
UNSIGNED_SHORT_5_6_5_REV:



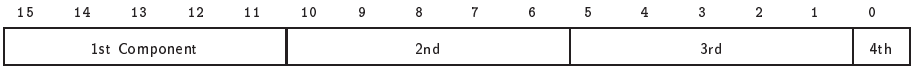
UNSIGNED_SHORT_4_4_4_4:



UNSIGNED_SHORT_4_4_4_4_REV:



UNSIGNED_SHORT_5_5_5_1:



UNSIGNED_SHORT_1_5_5_5_REV:

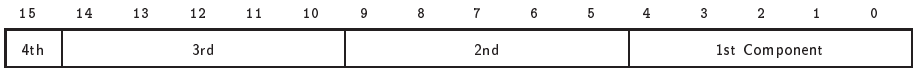
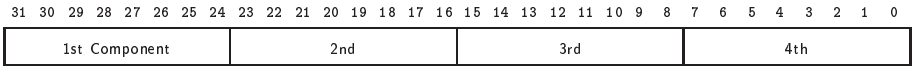


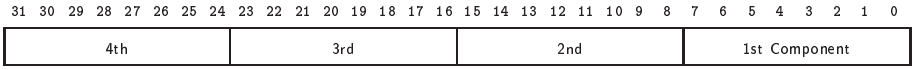
Table 3.10: UNSIGNED_SHORT formats

Microsoft Corp. Exhibit 1009

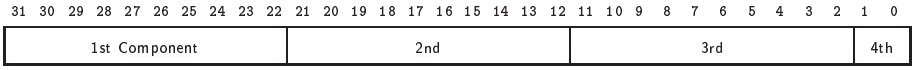
UNSIGNED_INT_8_8_8_8:



UNSIGNED_INT_8_8_8_8_REV:



UNSIGNED_INT_10_10_10_2:



UNSIGNED_INT_2_10_10_10_REV:

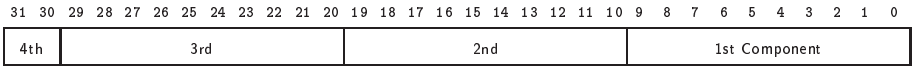


Table 3.11: UNSIGNED_INT formats

Microsoft Corp. Exhibit 1009

APPENDIX T

Format	First Component	Second Component	Third Component	Fourth Component
RGB	red	green	blue	
RGBA	red	green	blue	alpha
BGRA	blue	green	red	alpha

Table 3.12: Packed pixel field assignments

The assignment of component to fields in the packed pixel is as described in table 3.12

Byte swapping, if enabled, is performed before the component are extracted from each pixel. The above discussions of row length and image extraction are valid for packed pixels, if "group" is substituted for "component" and the number of components per group is understood to be one.

Calling **DrawPixels** with a *type* of BITMAP is a special case in which the data are a series of GL ubyte values. Each ubyte value specifies 8 1-bit elements with its 8 least-significant bits. The 8 single-bit elements are ordered from most significant to least significant if the value of UNPACK_LSB_FIRST is FALSE; otherwise, the ordering is from least significant to most significant. The values of bits other than the 8 least significant in each ubyte are not significant.

The first element of the first row is the first bit (as defined above) of the ubyte pointed to by the pointer passed to **DrawPixels**. The first element of the second row is the first bit (again as defined above) of the ubyte at location $p + k$, where k is computed as

$$k = a \left\lceil \frac{l}{8a} \right\rceil \tag{3.10}$$

There is a mechanism for selecting a sub-rectangle of elements from a BITMAP image as well. Before obtaining the first element from memory, the pointer supplied to **DrawPixels** is effectively advanced by UNPACK_SKIP_ROWS $*$ k ubytes. Then UNPACK_SKIP_PIXELS 1-bit elements are ignored, and the subsequent *width* 1-bit elements are obtained, without advancing the ubyte pointer, after which the pointer is advanced by k ubytes. *height* sets of *width* elements are obtained this way.

Microsoft Corp. Exhibit 1009

Conversion to floating-point

This step applies only to groups of components. It is not performed on indices. Each element in a group is converted to a floating-point value according to the appropriate formula in table 2.6 (section 2.13). For packed pixel types, each element in the group is converted by computing c / $(2^N - 1)$, where c is the unsigned integer value of the bitfield containing the element and N is the number of bits in the bitfield.

Conversion to RGB

This step is applied only if the *format* is `LUMINANCE` or `LUMINANCE_ALPHA`. If the *format* is `LUMINANCE`, then each group of one element is converted to a group of R, G, and B (three) elements by copying the original single element into each of the three new elements. If the *format* is `LUMINANCE_ALPHA`, then each group of two elements is converted to a group of R, G, B, and A (four) elements by copying the first original element into each of the first three new elements and copying the second original element to the A (fourth) new element.

Final Expansion to RGBA

This step is performed only for non-depth component groups. Each group is converted to a group of 4 elements as follows: if a group does not contain an A element, then A is added and set to 1.0. If any of R, G, or B is missing from the group, each missing element is added and assigned a value of 0.0.

Pixel Transfer Operations

This step is actually a sequence of steps. Because the pixel transfer operations are performed equivalently during the drawing, copying, and reading of pixels, and during the specification of texture images (either from memory or from the framebuffer), they are described separately in section 3.6.5. After the processing described in that section is completed, groups are processed as described in the following sections.

Final Conversion

For a color index, final conversion consists of masking the bits of the index to the left of the binary point by $2^n - 1$, where n is the number of bits in an index buffer. For RGBA components, each element is clamped to $[0, 1]$. The

Microsoft Corp. Exhibit 1009

resulting values are converted to fixed-point according to the rules given in section 2.13.9 (Final Color Processing).

For a depth component, an element is first clamped to $[0, 1]$ and then converted to fixed-point as if it were a window z value (see section 2.10.1, Controlling the Viewport).

Stencil indices are masked by $2^n - 1$, where n is the number of bits in the stencil buffer.

Conversion to Fragments

The conversion of a group to fragments is controlled with

`void` **PixelZoom**(`float` z_x , `float` z_y);

Let (x_{rp}, y_{rp}) be the current raster position (section 2.12). (If the current raster position is invalid, then **DrawPixels** is ignored; pixel transfer operations do not update the histogram or minmax tables, and no fragments are generated. However, the histogram and minmax tables are updated even if the corresponding fragments are later rejected by the pixel ownership (section 4.1.1) or scissor (section 4.1.2) tests.) If a particular group (index or components) is the nth in a row and belongs to the mth row, consider the region in window coordinates bounded by the rectangle with corners

$$(x_{rp} + z_x n, y_{rp} + z_y m) \qquad \text{and} \qquad (x_{rp} + z_x(n+1), y_{rp} + z_y(m+1))$$

(either z_x or z_y may be negative). Any fragments whose centers lie inside of this rectangle (or on its bottom or left boundaries) are produced in correspondence with this particular group of elements.

A fragment arising from a group consisting of color data takes on the color index or color components of the group; the depth and texture coordinates are taken from the current raster position's associated data. A fragment arising from a depth component takes the component's depth value; the color and texture coordinates are given by those associated with the current raster position. In both cases texture coordinates s, t, and r are replaced with s/q, t/q, and r/q, respectively. If q is less than or equal to zero, the results are undefined. Groups arising from **DrawPixels** with a *format* of `STENCIL_INDEX` are treated specially and are described in section 4.3.1.

3.6.5 Pixel Transfer Operations

The GL defines four kinds of pixel groups:

Microsoft Corp. Exhibit 1009

1. *RGBA component:* Each group comprises four color components: red, green, blue, and alpha.
2. *Depth component:* Each group comprises a single depth component.
3. *Color index:* Each group comprises a single color index.
4. *Stencil index:* Each group comprises a single stencil index.

Each operation described in this section is applied sequentially to each pixel group in an image. Many operations are applied only to pixel groups of certain kinds; if an operation is not applicable to a given group, it is skipped.

Arithmetic on Components

This step applies only to RGBA component and depth component groups. Each component is multiplied by an appropriate signed scale factor: `RED_SCALE` for an R component, `GREEN_SCALE` for a G component, `BLUE_SCALE` for a B component, and `ALPHA_SCALE` for an A component, or `DEPTH_SCALE` for a depth component. Then the result is added to the appropriate signed bias: `RED_BIAS`, `GREEN_BIAS`, `BLUE_BIAS`, `ALPHA_BIAS`, or `DEPTH_BIAS`.

Arithmetic on Indices

This step applies only to color index and stencil index groups. If the index is a floating-point value, it is converted to fixed-point, with an unspecified number of bits to the right of the binary point and at least $\lceil \log_2(\texttt{MAX_PIXEL_MAP_TABLE}) \rceil$ bits to the left of the binary point. Indices that are already integers remain so; any fraction bits in the resulting fixed-point value are zero.

The fixed-point index is then shifted by $|\texttt{INDEX_SHIFT}|$ bits, left if $\texttt{INDEX_SHIFT} > 0$ and right otherwise. In either case the shift is zero-filled. Then, the signed integer offset `INDEX_OFFSET` is added to the index.

RGBA to RGBA Lookup

This step applies only to RGBA component groups, and is skipped if `MAP_COLOR` is `FALSE`. First, each component is clamped to the range $[0, 1]$. There is a table associated with each of the R, G, B, and A component elements: `PIXEL_MAP_R_TO_R` for R, `PIXEL_MAP_G_TO_G` for G, `PIXEL_MAP_B_TO_B` for B, and `PIXEL_MAP_A_TO_A` for A. Each element is multiplied by an integer one less than the size of the corresponding table, and, for each element, an

Microsoft Corp. Exhibit 1009

address is found by rounding this value to the nearest integer. For each element, the addressed value in the corresponding table replaces the element.

Color Index Lookup

This step applies only to color index groups. If the GL command that invokes the pixel transfer operation requires that RGBA component pixel groups be generated, then a conversion is performed at this step. RGBA component pixel groups are required if

1. The groups will be rasterized, and the GL is in RGBA mode, or
2. The groups will be loaded as an image into texture memory, or
3. The groups will be returned to client memory with a format other than `COLOR_INDEX`.

If RGBA component groups are required, then the integer part of the index is used to reference 4 tables of color components: `PIXEL_MAP_I_TO_R`, `PIXEL_MAP_I_TO_G`, `PIXEL_MAP_I_TO_B`, and `PIXEL_MAP_I_TO_A`. Each of these tables must have 2^n entries for some integer value of n (n may be different for each table). For each table, the index is first rounded to the nearest integer; the result is ANDed with $2^n - 1$, and the resulting value used as an address into the table. The indexed value becomes an R, G, B, or A value, as appropriate. The group of four elements so obtained replaces the index, changing the group's type to RGBA component.

If RGBA component groups are not required, and if `MAP_COLOR` is enabled, then the index is looked up in the `PIXEL_MAP_I_TO_I` table (otherwise, the index is not looked up). Again, the table must have 2^n entries for some integer n. The index is first rounded to the nearest integer; the result is ANDed with $2^n - 1$, and the resulting value used as an address into the table. The value in the table replaces the index. The floating-point table value is first rounded to a fixed-point value with unspecified precision. The group's type remains color index.

Stencil Index Lookup

This step applies only to stencil index groups. If `MAP_STENCIL` is enabled, then the index is looked up in the `PIXEL_MAP_S_TO_S` table (otherwise, the index is not looked up). The table must have 2^n entries for some integer n. The integer index is ANDed with $2^n - 1$, and the resulting value used as an address into the table. The integer value in the table replaces the index.

Microsoft Corp. Exhibit 1009

Base Internal Format	R	G	B	A
ALPHA				A_t
LUMINANCE	L_t	L_t	L_t	
LUMINANCE_ALPHA	L_t	L_t	L_t	A_t
INTENSITY	I_t	I_t	I_t	I_t
RGB	R_t	G_t	B_t	
RGBA	R_t	G_t	B_t	A_t

Table 3.13: Color table lookup. R_t, G_t, B_t, A_t, L_t, and I_t are color table values that are assigned to pixel components R, G, B, and A depending on the table format. When there is no assignment, the component value is left unchanged by lookup.

Color Table Lookup

This step applies only to RGBA component groups. Color table lookup is only done if COLOR_TABLE is enabled. If a zero-width table is enabled, no lookup is performed.

The internal format of the table determines which components of the group will be replaced (see table 3.13). The components to be replaced are converted to indices by clamping to $[0, 1]$, multiplying by an integer one less than the width of the table, and rounding to the nearest integer. Components are replaced by the table entry at the index.

The required state is one bit indicating whether color table lookup is enabled or disabled. In the initial state, lookup is disabled.

Convolution

This step applies only to RGBA component groups. If CONVOLUTION_1D is enabled, the one-dimensional convolution filter is applied only to the one-dimensional texture images passed to **TexImage1D**, **TexSubImage1D**, **CopyTexImage1D**, and **CopyTexSubImage1D**, and returned by **GetTexImage** (see section 6.1.4) with target TEXTURE_1D. If CONVOLUTION_2D is enabled, the two-dimensional convolution filter is applied only to the two-dimensional images passed to **DrawPixels**, **CopyPixels**, **ReadPixels**, **TexImage2D**, **TexSubImage2D**, **CopyTexImage2D**, **CopyTexSubImage2D**, and **CopyTexSubImage3D**, and returned by **GetTexImage** with target TEXTURE_2D. If SEPARABLE_2D is enabled, and CONVOLUTION_2D is disabled, the separable two-dimensional convolution filter is instead ap-

Microsoft Corp. Exhibit 1009

Base Filter Format	R	G	B	A
ALPHA	R_s	G_s	B_s	$A_s * A_f$
LUMINANCE	$R_s * L_f$	$G_s * L_f$	$B_s * L_f$	A_s
LUMINANCE_ALPHA	$R_s * L_f$	$G_s * L_f$	$B_s * L_f$	$A_s * A_f$
INTENSITY	$R_s * I_f$	$G_s * I_f$	$B_s * I_f$	$A_s * I_f$
RGB	$R_s * R_f$	$G_s * G_f$	$B_s * B_f$	A_s
RGBA	$R_s * R_f$	$G_s * G_f$	$B_s * B_f$	$A_s * A_f$

Table 3.14: Computation of filtered color components depending on filter image format. $C * F$ indicates the convolution of image component C with filter F.

plied these images.

The convolution operation is a sum of products of source image pixels and convolution filter pixels. Source image pixels always have four components: red, green, blue, and alpha, denoted in the equations below as R_s, G_s, B_s, and A_s. Filter pixels may be stored in one of five formats, with 1, 2, 3, or 4 components. These components are denoted as R_f, G_f, B_f, A_f, L_f, and I_f in the equations below. The result of the convolution operation is the 4-tuple R,G,B,A. Depending on the internal format of the filter, individual color components of each source image pixel are convolved with one filter component, or are passed unmodified. The rules for this are defined in table 3.14.

The convolution operation is defined differently for each of the three convolution filters. The variables W_f and H_f refer to the dimensions of the convolution filter. The variables W_s and H_s refer to the dimensions of the source pixel image.

The convolution equations are defined as follows, where C refers to the filtered result, C_f refers to the one- or two-dimensional convolution filter, and C_{row} and C_{column} refer to the two one-dimensional filters comprising the two-dimensional separable filter. C'_s depends on the source image color C_s and the convolution border mode as described below. C_r, the filtered output image, depends on all of these variables and is described separately for each border mode. The pixel indexing nomenclature is decribed in the **Convolution Filter Specification** subsection of section 3.6.3.

One-dimensional filter:

$$C[i'] = \sum_{n=0}^{W_f-1} C'_s[i'+n] * C_f[n]$$

Microsoft Corp. Exhibit 1009

Two-dimensional filter:

$$C[i', j'] = \sum_{n=0}^{W_f - 1} \sum_{m=0}^{H_f - 1} C'_s[i' + n, j' + m] * C_f[n, m]$$

Two-dimensional separable filter:

$$C[i', j'] = \sum_{n=0}^{W_f - 1} \sum_{m=0}^{H_f - 1} C'_s[i' + n, j' + m] * C_{row}[n] * C_{column}[m]$$

If W_f of a one-dimensional filter is zero, then $C[i]$ is always set to zero. Likewise, if either W_f or H_f of a two-dimensional filter is zero, then $C[i, j]$ is always set to zero.

The convolution border mode for a specific convolution filter is specified by calling

`void` **ConvolutionParameter{if}**(`enum` *target*,
`enum` *pname*, `T` *param*);

where *target* is the name of the filter, *pname* is `CONVOLUTION_BORDER_MODE`, and *param* is one of `REDUCE`, `CONSTANT_BORDER` or `REPLICATE_BORDER`.

Border Mode `REDUCE`

The width and height of source images convolved with border mode `REDUCE` are reduced by $W_f - 1$ and $H_f - 1$, respectively. If this reduction would generate a resulting image with zero or negative width and/or height, the output is simply null, with no error generated. The coordinates of the image that results from a convolution with border mode `REDUCE` are zero through $W_s - W_f$ in width, and zero through $H_s - H_f$ in height. In cases where errors can result from the specification of invalid image dimensions, it is these resulting dimensions that are tested, not the dimensions of the source image. (A specific example is **TexImage1D** and **TexImage2D**, which specify constraints for image dimensions. Even if **TexImage1D** or **TexImage2D** is called with a null pixel pointer, the dimensions of the resulting texture image are those that would result from the convolution of the specified image).

When the border mode is `REDUCE`, C'_s equals the source image color C_s and C_r equals the filtered result C.

For the remaining border modes, define $C_w = \lfloor W_f/2 \rfloor$ and $C_h = \lfloor H_f/2 \rfloor$. The coordinates (C_w, C_h) define the center of the convolution filter.

Microsoft Corp. Exhibit 1009

Border Mode CONSTANT_BORDER

If the convolution border mode is CONSTANT_BORDER, the output image has the same dimensions as the source image. The result of the convolution is the same as if the source image were surrounded by pixels with the same color as the current convolution border color. Whenever the convolution filter extends beyond one of the edges of the source image, the constant-color border pixels are used as input to the filter. The current convolution border color is set by calling **ConvolutionParameterfv** or **ConvolutionParameteriv** with *pname* set to CONVOLUTION_BORDER_COLOR and *params* containing four values that comprise the RGBA color to be used as the image border. Integer color components are interpreted linearly such that the most positive integer maps to 1.0, and the most negative integer maps to -1.0. Floating point color components are not clamped when they are specified.

For a one-dimensional filter, the result color is defined by

$$C_r[i] = C[i - C_w]$$

where $C[i']$ is computed using the following equation for $C'_s[i']$:

$$C'_s[i'] = \begin{cases} C_s[i'], & 0 \le i' < W_s \\ C_c, & otherwise \end{cases}$$

and C_c is the convolution border color.

For a two-dimensional or two-dimensional separable filter, the result color is defined by

$$C_r[i, j] = C[i - C_w, j - C_h]$$

where $C[i', j']$ is computed using the following equation for $C'_s[i', j']$:

$$C'_s[i', j'] = \begin{cases} C_s[i', j'], & 0 \le i' < W_s, 0 \le j' < H_s \\ C_c, & otherwise \end{cases}$$

Border Mode REPLICATE_BORDER

The convolution border mode REPLICATE_BORDER also produces an output image with the same dimensions as the source image. The behavior of this mode is identical to that of the CONSTANT_BORDER mode except for the treatment of pixel locations where the convolution filter extends beyond the edge of the source image. For these locations, it is as if the outermost one-pixel border of the source image was replicated. Conceptually, each pixel in

Microsoft Corp. Exhibit 1009

the leftmost one-pixel column of the source image is replicated C_w times to provide additional image data along the left edge, each pixel in the rightmost one-pixel column is replicated C_w times to provide additional image data along the right edge, and each pixel value in the top and bottom one-pixel rows is replicated to create C_h rows of image data along the top and bottom edges. The pixel value at each corner is also replicated in order to provide data for the convolution operation at each corner of the source image.

For a one-dimensional filter, the result color is defined by

$$C_r[i] = C[i - C_w]$$

where $C[i']$ is computed using the following equation for $C'_s[i']$:

$$C'_s[i'] = C_s[\text{clamp}(i', W_s)]$$

and the clamping function $\text{clamp}(val, max)$ is defined as

$$\text{clamp}(val, max) = \begin{cases} 0, & val < 0 \\ val, & 0 \leq val < max \\ max - 1, & val >= max \end{cases}$$

For a two-dimensional or two-dimensional separable filter, the result color is defined by

$$C_r[i, j] = C[i - C_w, j - C_h]$$

where $C[i', j']$ is computed using the following equation for $C'_s[i', j']$:

$$C'_s[i', j'] = C_s[\text{clamp}(i', W_s), \text{clamp}(j', H_s)]$$

After convolution, each component of the resulting image is scaled by the corresponding **PixelTransfer** parameters: `POST_CONVOLUTION_RED_SCALE` for an R component, `POST_CONVOLUTION_GREEN_SCALE` for a G component, `POST_CONVOLUTION_BLUE_SCALE` for a B component, and `POST_CONVOLUTION_ALPHA_SCALE` for an A component. The result is added to the corresponding bias: `POST_CONVOLUTION_RED_BIAS`, `POST_CONVOLUTION_GREEN_BIAS`, `POST_CONVOLUTION_BLUE_BIAS`, or `POST_CONVOLUTION_ALPHA_BIAS`.

The required state is three bits indicating whether each of one-dimensional, two-dimensional, or separable two-dimensional convolution is enabled or disabled, an integer describing the current convolution border mode, and four floating-point values specifying the convolution border color. In the initial state, all convolution operations are disabled, the border mode is `REDUCE`, and the border color is $(0, 0, 0, 0)$.

Microsoft Corp. Exhibit 1009

Post Convolution Color Table Lookup

This step applies only to RGBA component groups. Post convolution color table lookup is enabled or disabled by calling **Enable** or **Disable** with the symbolic constant `POST_CONVOLUTION_COLOR_TABLE`. The post convolution table is defined by calling **ColorTable** with a *target* argument of `POST_CONVOLUTION_COLOR_TABLE`. In all other respects, operation is identical to color table lookup, as defined earlier in section 3.6.5.

The required state is one bit indicating whether post convolution table lookup is enabled or disabled. In the initial state, lookup is disabled.

Color Matrix Transformation

This step applies only to RGBA component groups. The components are transformed by the color matrix. Each transformed component is multiplied by an appropriate signed scale factor: `POST_COLOR_MATRIX_RED_SCALE` for an R component, `POST_COLOR_MATRIX_GREEN_SCALE` for a G component, `POST_COLOR_MATRIX_BLUE_SCALE` for a B component, and `POST_COLOR_MATRIX_ALPHA_SCALE` for an A component. The result is added to a signed bias: `POST_COLOR_MATRIX_RED_BIAS`, `POST_COLOR_MATRIX_GREEN_BIAS`, `POST_COLOR_MATRIX_BLUE_BIAS`, or `POST_COLOR_MATRIX_ALPHA_BIAS`. The resulting components replace each component of the original group.

That is, if M_c is the color matrix, a subscript of s represents the scale term for a component, and a subscript of b represents the bias term, then the components

$$\begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix}$$

are transformed to

$$\begin{pmatrix} R' \\ G' \\ B' \\ A' \end{pmatrix} = \begin{pmatrix} R_s & 0 & 0 & 0 \\ 0 & G_s & 0 & 0 \\ 0 & 0 & B_s & 0 \\ 0 & 0 & 0 & A_s \end{pmatrix} M_c \begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} + \begin{pmatrix} R_b \\ G_b \\ B_b \\ A_b \end{pmatrix}.$$

Post Color Matrix Color Table Lookup

This step applies only to RGBA component groups. Post color matrix color table lookup is enabled or disabled by calling **Enable** or **Disable**

Microsoft Corp. Exhibit 1009

APPENDIX T

with the symbolic constant POST_COLOR_MATRIX_COLOR_TABLE. The post color matrix table is defined by calling **ColorTable** with a *target* argument of POST_COLOR_MATRIX_COLOR_TABLE. In all other respects, operation is identical to color table lookup, as defined in section 3.6.5.

The required state is one bit indicating whether post color matrix lookup is enabled or disabled. In the initial state, lookup is disabled.

Histogram

This step applies only to RGBA component groups. Histogram operation is enabled or disabled by calling **Enable** or **Disable** with the symbolic constant HISTOGRAM.

If the width of the table is non-zero, then indices R_i, G_i, B_i, and A_i are derived from the red, green, blue, and alpha components of each pixel group (without modifying these components) by clamping each component to $[0, 1]$, multiplying by one less than the width of the histogram table, and rounding to the nearest integer. If the format of the HISTOGRAM table includes red or luminance, the red or luminance component of histogram entry R_i is incremented by one. If the format of the HISTOGRAM table includes green, the green component of histogram entry G_i is incremented by one. The blue and alpha components of histogram entries B_i and A_i are incremented in the same way. If a histogram entry component is incremented beyond its maximum value, its value becomes undefined; this is not an error.

If the **Histogram** *sink* parameter is FALSE, histogram operation has no effect on the stream of pixel groups being processed. Otherwise, all RGBA pixel groups are discarded immediately after the histogram operation is completed. Because histogram precedes minmax, no minmax operation is performed. No pixel fragments are generated, no change is made to texture memory contents, and no pixel values are returned. However, texture object state is modified whether or not pixel groups are discarded.

Minmax

This step applies only to RGBA component groups. Minmax operation is enabled or disabled by calling **Enable** or **Disable** with the symbolic constant MINMAX.

If the format of the minmax table includes red or luminance, the red component value replaces the red or luminance value in the minimum table element if and only if it is less than that component. Likewise, if the format includes red or luminance and the red component of the group is greater

Microsoft Corp. Exhibit 1009

than the red or luminance value in the maximum element, the red group component replaces the red or luminance maximum component. If the format of the table includes green, the green group component conditionally replaces the green minimum and/or maximum if it is smaller or larger, respectively. The blue and alpha group components are similarly tested and replaced, if the table format includes blue and/or alpha. The internal type of the minimum and maximum component values is floating point, with at least the same representable range as a floating point number used to represent colors (section 2.1.1). There are no semantics defined for the treatment of group component values that are outside the representable range.

If the **Minmax** *sink* parameter is `FALSE`, minmax operation has no effect on the stream of pixel groups being processed. Otherwise, all RGBA pixel groups are discarded immediately after the minmax operation is completed. No pixel fragments are generated, no change is made to texture memory contents, and no pixel values are returned. However, texture object state is modified whether or not pixel groups are discarded.

3.7 Bitmaps

Bitmaps are rectangles of zeros and ones specifying a particular pattern of fragments to be produced. Each of these fragments has the same associated data. These data are those associated with the *current raster position.*

Bitmaps are sent using

```
void Bitmap( sizei w, sizei h, float x_bo, float y_bo,
   float x_bi, float y_bi, ubyte *data );
```

w and h comprise the integer width and height of the rectangular bitmap, respectively. (x_{bo}, y_{bo}) gives the floating-point x and y values of the bitmap's origin. (x_{bi}, y_{bi}) gives the floating-point x and y increments that are added to the raster position after the bitmap is rasterized. *data* is a pointer to a bitmap.

Like a polygon pattern, a bitmap is unpacked from memory according to the procedure given in section 3.6.4 for **DrawPixels**; it is as if the *width* and *height* passed to that command were equal to w and h, respectively, the *type* were `BITMAP`, and the *format* were `COLOR_INDEX`. The unpacked values (before any conversion or arithmetic would have been performed) form a stipple pattern of zeros and ones. See figure 3.9.

A bitmap sent using **Bitmap** is rasterized as follows. First, if the current raster position is invalid (the valid bit is reset), the bitmap is ignored.

Microsoft Corp. Exhibit 1009

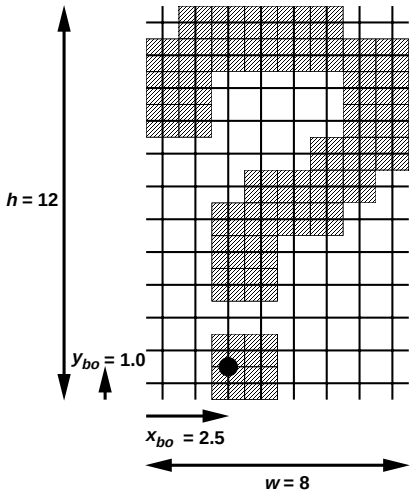


Figure 3.9. A bitmap and its associated parameters. x_{bi} and y_{bi} are not shown.

Otherwise, a rectangular array of fragments is constructed, with lower left corner at

$$(x_{ll}, y_{ll}) = (\lfloor x_{rp} - x_{bo} \rfloor, \lfloor y_{rp} - y_{bo} \rfloor)$$

and upper right corner at $(x_{ll} + w, y_{ll} + h)$ where w and h are the width and height of the bitmap, respectively. Fragments in the array are produced if the corresponding bit in the bitmap is 1 and not produced otherwise. The associated data for each fragment are those associated with the current raster position, with texture coordinates s, t, and r replaced with s/q, t/q, and r/q, respectively. If q is less than or equal to zero, the results are undefined. Once the fragments have been produced, the current raster position is updated:

$$(x_{rp}, y_{rp}) \leftarrow (x_{rp} + x_{bi}, y_{rp} + y_{bi}).$$

The z and w values of the current raster position remain unchanged.

3.8 Texturing

Texturing maps a portion of a specified image onto each primitive for which texturing is enabled. This mapping is accomplished by using the color of

Microsoft Corp. Exhibit 1009

an image at the location indicated by a fragment's (s, t, r) coordinates to modify the fragment's primary RGBA color. Texturing does not affect the secondary color.

Texturing is specified only for RGBA mode; its use in color index mode is undefined.

The GL provides a means to specify the details of how texturing of a primitive is effected. These details include specification of the image to be texture mapped, the means by which the image is filtered when applied to the primitive, and the function that determines what RGBA value is produced given a fragment color and an image value.

3.8.1 Texture Image Specification

The command

```
void TexImage3D( enum target, int level,
    int internalformat, sizei width, sizei height,
    sizei depth, int border, enum format, enum type,
    void *data );
```

is used to specify a three-dimensional texture image. *target* must be either TEXTURE_3D, or PROXY_TEXTURE_3D in the special case discussed in section 3.8.7. *format*, *type*, and *data* match the corresponding arguments to **DrawPixels** (refer to section 3.6.4); they specify the format of the image data, the type of those data, and a pointer to the image data in host memory. The *format*s STENCIL_INDEX and DEPTH_COMPONENT are not allowed.

The groups in memory are treated as being arranged in a sequence of adjacent rectangles. Each rectangle is a two-dimensional image, whose size and organization are specified by the *width* and *height* parameters to **TexImage3D**. The values of UNPACK_ROW_LENGTH and UNPACK_ALIGNMENT control the row-to-row spacing in these images in the same manner as **DrawPixels**. If the value of the integer parameter UNPACK_IMAGE_HEIGHT is not positive, then the number of rows in each two-dimensional image is *height*; otherwise the number of rows is UNPACK_IMAGE_HEIGHT. Each two-dimensional image comprises an integral number of rows, and is exactly adjacent to its neighbor images.

The mechanism for selecting a sub-volume of a three-dimensional image relies on the integer parameter UNPACK_SKIP_IMAGES. If UNPACK_SKIP_IMAGES is positive, the pointer is advanced by UNPACK_SKIP_IMAGES times the number of elements in one two-dimensional image before obtaining the first group from

Microsoft Corp. Exhibit 1009

memory. Then *depth* two-dimensional images are processed, each having a subimage extracted in the same manner as **DrawPixels**.

The selected groups are processed exactly as for **DrawPixels**, stopping just before final conversion. Each R, G, B, and A value so generated is clamped to $[0, 1]$.

Components are then selected from the resulting R, G, B, and A values to obtain a texture with the *base internal format* specified by (or derived from) *internalformat*. Table 3.15 summarizes the mapping of R, G, B, and A values to texture components, as a function of the base internal format of the texture image. *internalformat* may be specified as one of the six base internal format symbolic constants listed in table 3.15, or as one of the *sized internal format* symbolic constants listed in table 3.16. *internalformat* may (for backwards compatibility with the 1.0 version of the GL) also take on the integer values `1`, `2`, `3`, and `4`, which are equivalent to symbolic constants `LUMINANCE`, `LUMINANCE_ALPHA`, `RGB`, and `RGBA` respectively. Specifying a value for *internalformat* that is not one of the above values generates the error `INVALID_VALUE`.

The *internal component resolution* is the number of bits allocated to each value in a texture image. If *internalformat* is specified as a base internal format, the GL stores the resulting texture with internal component resolutions of its own choosing. If a sized internal format is specified, the mapping of the R, G, B, and A values to texture components is equivalent to the mapping of the corresponding base internal format's components, as specified in table 3.15, and the memory allocation per texture component is assigned by the GL to match the allocations listed in table 3.16 as closely as possible. (The definition of closely is left up to the implementation. Implementations are not required to support more than one resolution for each base internal format.)

A GL implementation may vary its allocation of internal component resolution based on any **TexImage3D**, **TexImage2D** (see below), or **TexImage1D** (see below) parameter (except *target*), but the allocation must not be a function of any other state, and cannot be changed once it is established. Allocations must be invariant; the same allocation must be made each time a texture image is specified with the same parameter values. These allocation rules also apply to proxy textures, which are described in section 3.8.7.

The image itself (pointed to by *data*) is a sequence of groups of values. The first group is the lower left back corner of the texture image. Subsequent groups fill out rows of width *width* from left to right; *height* rows are stacked from bottom to top forming a single two-dimensional image slice; and *depth* slices are stacked from back to front. When the final R, G, B,

Microsoft Corp. Exhibit 1009

Base Internal Format	RGBA Values	Internal Components
ALPHA	A	A
LUMINANCE	R	L
LUMINANCE_ALPHA	R,A	L,A
INTENSITY	R	I
RGB	R,G,B	R,G,B
RGBA	R,G,B,A	R,G,B,A

Table 3.15: Conversion from RGBA pixel components to internal texture, table, or filter components. See section 3.8.9 for a description of the texture components R, G, B, A, L, and I.

and A components have been computed for a group, they are assigned to components of a *texel* as described by table 3.15. Counting from zero, each resulting Nth texel is assigned internal integer coordinates (i, j, k), where

$$i = (N \bmod width) - b_s$$

$$j = (\lfloor \frac{N}{width} \rfloor \bmod height) - b_s$$

$$k = (\lfloor \frac{N}{width \times height} \rfloor \bmod depth) - b_s$$

and b_s is the specified *border* width. Thus the last two-dimensional image slice of the three-dimensional image is indexed with the highest value of k.

Each color component is converted (by rounding to nearest) to a fixed-point value with n bits, where n is the number of bits of storage allocated to that component in the image array. We assume that the fixed-point representation used represents each value $k/(2^n - 1)$, where $k \in \{0, 1, \ldots, 2^n - 1\}$, as k (e.g. 1.0 is represented in binary as a string of all ones).

The *level* argument to **TexImage3D** is an integer *level-of-detail* number. Levels of detail are discussed below, under **Mipmapping**. The main texture image has a level of detail number of 0. If a level-of-detail less than zero is specified, the error INVALID_VALUE is generated.

The *border* argument to **TexImage3D** is a border width. The significance of borders is described below. The border width affects the required dimensions of the texture image: it must be the case that

$$w_s = 2^n + 2b_s \tag{3.11}$$

Microsoft Corp. Exhibit 1009

Sized Internal Format	Base Internal Format	R bits	G bits	B bits	A bits	L bits	I bits
ALPHA4	ALPHA				4		
ALPHA8	ALPHA				8		
ALPHA12	ALPHA				12		
ALPHA16	ALPHA				16		
LUMINANCE4	LUMINANCE					4	
LUMINANCE8	LUMINANCE					8	
LUMINANCE12	LUMINANCE					12	
LUMINANCE16	LUMINANCE					16	
LUMINANCE4_ALPHA4	LUMINANCE_ALPHA				4	4	
LUMINANCE6_ALPHA2	LUMINANCE_ALPHA				2	6	
LUMINANCE8_ALPHA8	LUMINANCE_ALPHA				8	8	
LUMINANCE12_ALPHA4	LUMINANCE_ALPHA				4	12	
LUMINANCE12_ALPHA12	LUMINANCE_ALPHA				12	12	
LUMINANCE16_ALPHA16	LUMINANCE_ALPHA				16	16	
INTENSITY4	INTENSITY						4
INTENSITY8	INTENSITY						8
INTENSITY12	INTENSITY						12
INTENSITY16	INTENSITY						16
R3_G3_B2	RGB	3	3	2			
RGB4	RGB	4	4	4			
RGB5	RGB	5	5	5			
RGB8	RGB	8	8	8			
RGB10	RGB	10	10	10			
RGB12	RGB	12	12	12			
RGB16	RGB	16	16	16			
RGBA2	RGBA	2	2	2	2		
RGBA4	RGBA	4	4	4	4		
RGB5_A1	RGBA	5	5	5	1		
RGBA8	RGBA	8	8	8	8		
RGB10_A2	RGBA	10	10	10	2		
RGBA12	RGBA	12	12	12	12		
RGBA16	RGBA	16	16	16	16		

Table 3.16: Correspondence of sized internal formats to base internal formats, and *desired* component resolutions for each sized internal format.

Microsoft Corp. Exhibit 1009

$$h_s = 2^m + 2b_s \tag{3.12}$$

$$d_s = 2^l + 2b_s \tag{3.13}$$

for some integers n, m, and l, where w_s, h_s, and d_s are the specified image *width*, *height*, and *depth*. If any one of these relationships cannot be satisfied, then the error `INVALID_VALUE` is generated.

Currently, the maximum border width b_t is 1. If b_s is less than zero, or greater than b_t, then the error `INVALID_VALUE` is generated.

The maximum allowable width, height, or depth of a three-dimensional texture image is an implementation dependent function of the level-of-detail and internal format of the resulting image array. It must be at least $2^{k-lod} + 2b_t$ for image arrays of level-of-detail 0 through k, where k is the log base 2 of `MAX_3D_TEXTURE_SIZE`, *lod* is the level-of-detail of the image array, and b_t is the maximum border width. It may be zero for image arrays of any level-of-detail greater than k. The error `INVALID_VALUE` is generated if the specified image is too large to be stored under any conditions.

In a similar fashion, the maximum allowable width of a one- or two-dimensional texture image, and the maximum allowable height of a two-dimensional texture image, must be at least $2^{k-lod} + 2b_t$ for image arrays of level 0 through k, where k is the log base 2 of `MAX_TEXTURE_SIZE`.

Furthermore, an implementation may allow a one-, two-, or three-dimensional image array of level 1 or greater to be created only if a complete[1] set of image arrays consistent with the requested array can be supported. Likewise, an implementation may allow an image array of level 0 to be created only if that single image array can be supported.

The command

```
void TexImage2D( enum target, int level,
    int internalformat, sizei width, sizei height,
    int border, enum format, enum type, void *data );
```

is used to specify a two-dimensional texture image. *target* must be either `TEXTURE_2D`, or `PROXY_TEXTURE_2D` in the special case discussed in section 3.8.7. The other parameters match the corresponding parameters of **TexImage3D**.

[1] For this purpose the definition of "complete", as provided under **Mipmapping**, is augmented as follows: 1) it is as though `TEXTURE_BASE_LEVEL` is 0 and `TEXTURE_MAX_LEVEL` is 1000. 2) Excluding borders, the dimensions of the next lower numbered array are all understood to be twice the corresponding dimensions of the specified array.

Microsoft Corp. Exhibit 1009

For the purposes of decoding the texture image, **TexImage2D** is equivalent to calling **TexImage3D** with corresponding arguments and *depth* of 1, except that

- The *depth* of the image is always 1 regardless of the value of *border*.
- Convolution will be performed on the image (possibly changing its *width* and *height*) if `SEPARABLE_2D` or `CONVOLUTION_2D` is enabled.
- `UNPACK_SKIP_IMAGES` is ignored.

Finally, the command

```
void TexImage1D( enum target, int level,
    int internalformat, sizei width, int border,
    enum format, enum type, void *data );
```

is used to specify a one-dimensional texture image. *target* must be either `TEXTURE_1D`, or `PROXY_TEXTURE_1D` in the special case discussed in section 3.8.7.)

For the purposes of decoding the texture image, **TexImage1D** is equivalent to calling **TexImage2D** with corresponding arguments and *height* of 1, except that

- The *height* of the image is always 1 regardless of the value of *border*.
- Convolution will be performed on the image (possibly changing its *width*) only if `CONVOLUTION_1D` is enabled.

An image with zero width, height (**TexImage2D** and **TexImage3D** only), or depth (**TexImage3D** only) indicates the null texture. If the null texture is specified for the level-of-detail specified by `TEXTURE_BASE_LEVEL`, it is as if texturing were disabled.

The image indicated to the GL by the image pointer is decoded and copied into the GL's internal memory. This copying effectively places the decoded image inside a border of the maximum allowable width b_t whether or not a border has been specified (see figure 3.10) [2]. If no border or a border smaller than the maximum allowable width has been specified, then the image is still stored as if it were surrounded by a border of the maximum possible width. Any excess border (which surrounds the specified image,

[2] Figure 3.10 needs to show a three-dimensional texture image.

Microsoft Corp. Exhibit 1009

including any border) is assigned unspecified values. A two-dimensional texture has a border only at its left, right, top, and bottom ends, and a one-dimensional texture has a border only at its left and right ends.

We shall refer to the (possibly border augmented) decoded image as the *texture array*. A three-dimensional texture array has width, height, and depth

$$w_t = 2^n + 2b_t$$

$$h_t = 2^m + 2b_t$$

$$d_t = 2^l + 2b_t$$

where b_t is the maximum allowable border width and n, m, and l are defined in equations 3.11, 3.12, and 3.13. A two-dimensional texture array has depth $d_t = 1$, with height h_t and width w_t as above, and a one-dimensional texture array has depth $d_t = 1$, height $h_t = 1$, and width w_t as above.

An element (i, j, k) of the texture array is called a *texel* (for a two-dimensional texture, k is irrelevant; for a one-dimensional texture, j and k are both irrelevant). The *texture value* used in texturing a fragment is determined by that fragment's associated (s, t, r) coordinates, but may not correspond to any actual texel. See figure 3.10.

If the *data* argument of **TexImage1D**, **TexImage2D**, or **TexImage3D** is a null pointer (a zero-valued pointer in the C implementation), a one-, two-, or three-dimensional texture array is created with the specified *target*, *level*, *internalformat*, *width*, *height*, and *depth*, but with unspecified image contents. In this case no pixel values are accessed in client memory, and no pixel processing is performed. Errors are generated, however, exactly as though the *data* pointer were valid.

3.8.2 Alternate Texture Image Specification Commands

Two-dimensional and one-dimensional texture images may also be specified using image data taken directly from the framebuffer, and rectangular subregions of existing texture images may be respecified.

The command

```
void CopyTexImage2D( enum target, int level,
    enum internalformat, int x, int y, sizei width,
    sizei height, int border );
```


Microsoft Corp. Exhibit 1009

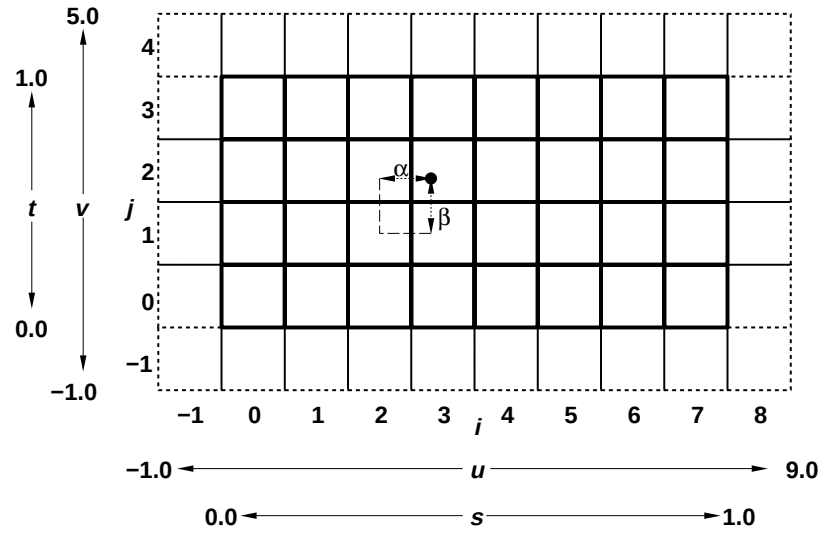


Figure 3.10. A texture image and the coordinates used to access it. This is a two-dimensional texture with $n = 3$ and $m = 2$. A one-dimensional texture would consist of a single horizontal strip. α and β, values used in blending adjacent texels to obtain a texture value, are also shown.

Microsoft Corp. Exhibit 1009

defines a two-dimensional texture array in exactly the manner of **TexImage2D**, except that the image data are taken from the framebuffer rather than from client memory. Currently, *target* must be `TEXTURE_2D`. *x*, *y*, *width*, and *height* correspond precisely to the corresponding arguments to **CopyPixels** (refer to section 4.3.3); they specify the image's *width* and *height*, and the lower left (x, y) coordinates of the framebuffer region to be copied. The image is taken from the framebuffer exactly as if these arguments were passed to **CopyPixels**, with argument *type* set to `COLOR`, stopping after pixel transfer processing is complete. Subsequent processing is identical to that described for **TexImage2D**, beginning with clamping of the R, G, B, and A values from the resulting pixel groups. Parameters *level*, *internalformat*, and *border* are specified using the same values, with the same meanings, as the equivalent arguments of **TexImage2D**, except that *internalformat* may not be specified as `1`, `2`, `3`, or `4`. An invalid value specified for *internalformat* generates the error `INVALID_ENUM`. The constraints on *width*, *height*, and *border* are exactly those for the equivalent arguments of **TexImage2D**.

The command

```
void CopyTexImage1D( enum target, int level,
    enum internalformat, int x, int y, sizei width,
    int border );
```

defines a one-dimensional texture array in exactly the manner of **TexImage1D**, except that the image data are taken from the framebuffer, rather than from client memory. Currently, *target* must be `TEXTURE_1D`. For the purposes of decoding the texture image, **CopyTexImage1D** is equivalent to calling **CopyTexImage2D** with corresponding arguments and *height* of 1, except that the *height* of the image is always 1, regardless of the value of *border*. *level*, *internalformat*, and *border* are specified using the same values, with the same meanings, as the equivalent arguments of **TexImage1D**, except that *internalformat* may not be specified as `1`, `2`, `3`, or `4`. The constraints on *width* and *border* are exactly those of the equivalent arguments of **TexImage1D**.

Six additional commands,

```
void TexSubImage3D( enum target, int level, int xoffset,
    int yoffset, int zoffset, sizei width, sizei height,
    sizei depth, enum format, enum type, void *data );
void TexSubImage2D( enum target, int level, int xoffset,
    int yoffset, sizei width, sizei height, enum format,
    enum type, void *data );
```


Microsoft Corp. Exhibit 1009

APPENDIX T


```
void TexSubImage1D( enum target, int level, int xoffset,
    sizei width, enum format, enum type, void *data );
void CopyTexSubImage3D( enum target, int level,
    int xoffset, int yoffset, int zoffset, int x, int y,
    sizei width, sizei height );
void CopyTexSubImage2D( enum target, int level,
    int xoffset, int yoffset, int x, int y, sizei width,
    sizei height );
void CopyTexSubImage1D( enum target, int level,
    int xoffset, int x, int y, sizei width );
```

respecify only a rectangular subregion of an existing texture array. No change is made to the *internalformat*, *width*, *height*, *depth*, or *border* parameters of the specified texture array, nor is any change made to texel values outside the specified subregion. Currently the *target* arguments of **TexSubImage1D** and **CopyTexSubImage1D** must be TEXTURE_1D, the *target* arguments of **TexSubImage2D** and **CopyTexSubImage2D** must be TEXTURE_2D, and the *target* arguments of **TexSubImage3D** and **CopyTexSubImage3D** must be TEXTURE_3D. The *level* parameter of each command specifies the level of the texture array that is modified. If *level* is less than zero or greater than the base 2 logarithm of the maximum texture width or height, the error INVALID_VALUE is generated.

TexSubImage3D arguments *width*, *height*, *depth*, *format*, *type*, and *data* match the corresponding arguments to **TexImage3D**, meaning that they are specified using the same values, and have the same meanings. Likewise, **TexSubImage2D** arguments *width*, *height*, *format*, *type*, and *data* match the corresponding arguments to **TexImage2D**, and **TexSubImage1D** arguments *width*, *format*, *type*, and *data* match the corresponding arguments to **TexImage1D**.

CopyTexSubImage3D and **CopyTexSubImage2D** arguments x, y, *width*, and *height* match the corresponding arguments to **CopyTexImage2D**[3]. **CopyTexSubImage1D** arguments x, y, and *width* match the corresponding arguments to **CopyTexImage1D**. Each of the **TexSubImage** commands interprets and processes pixel groups in exactly the manner of its **TexImage** counterpart, except that the assignment of R, G, B, and A pixel group values to the texture components is controlled by the *internalformat* of the texture array, not by an argument to the command.

[3]Because the framebuffer is inherently two-dimensional, there is no **CopyTexImage3D** command.

Microsoft Corp. Exhibit 1009

Arguments *xoffset*, *yoffset*, and *zoffset* of **TexSubImage3D** and **CopyTexSubImage3D** specify the lower left texel coordinates of a *width*-wide by *height*-high by *depth*-deep rectangular subregion of the texture array. The *depth* argument associated with **CopyTexSubImage3D** is always 1, because framebuffer memory is two-dimensional - only a portion of a single s, t slice of a three-dimensional texture is replaced by **CopyTexSubImage3D**.

Negative values of *xoffset*, *yoffset*, and *zoffset* correspond to the coordinates of border texels, addressed as in figure 3.10. Taking w_s, h_s, d_s, and b_s to be the specified width, height, depth, and border width of the texture array, (not the actual array dimensions w_t, h_t, d_t, and b_t), and taking x, y, z, w, h, and d to be the *xoffset*, *yoffset*, *zoffset*, *width*, *height*, and *depth* argument values, any of the following relationships generates the error `INVALID_VALUE`:

$$x < -b_s$$

$$x + w > w_s - b_s$$

$$y < -b_s$$

$$y + h > h_s - b_s$$

$$z < -b_s$$

$$z + d > d_s - b_s$$

(Recall that d_s, w_s, and h_s include twice the specified border width b_s.) Counting from zero, the nth pixel group is assigned to the texel with internal integer coordinates $[i, j, k]$, where

$$i = x + (n \bmod w)$$

$$j = y + (\lfloor \frac{n}{w} \rfloor \bmod h)$$

$$k = z + (\lfloor \frac{n}{width * height} \rfloor \bmod d$$

Arguments *xoffset* and *yoffset* of **TexSubImage2D** and **CopyTexSubImage2D** specify the lower left texel coordinates of a *width*-wide by *height*-high rectangular subregion of the texture array. Negative values of *xoffset* and *yoffset* correspond to the coordinates of border texels, addressed as in figure 3.10. Taking w_s, h_s, and b_s to be the specified width, height, and border width of the texture array, (not the actual array dimensions w_t, h_t, and b_t), and taking x, y, w, and h to be the *xoffset*, *yoffset*, *width*, and

Microsoft Corp. Exhibit 1009

height argument values, any of the following relationships generates the error `INVALID_VALUE`:

$$x < -b_s$$
$$x + w > w_s - b_s$$
$$y < -b_s$$
$$y + h > h_s - b_s$$

(Recall that w_s and h_s include twice the specified border width b_s.) Counting from zero, the nth pixel group is assigned to the texel with internal integer coordinates $[i, j]$, where

$$i = x + (n \bmod w)$$
$$j = y + (\lfloor \frac{n}{w} \rfloor \bmod h)$$

The *xoffset* argument of **TexSubImage1D** and **CopyTexSubImage1D** specifies the left texel coordinate of a *width*-wide subregion of the texture array. Negative values of *xoffset* correspond to the coordinates of border texels. Taking w_s and b_s to be the specified width and border width of the texture array, and x and w to be the *xoffset* and *width* argument values, either of the following relationships generates the error `INVALID_VALUE`:

$$x < -b_s$$
$$x + w > w_s - b_s$$

Counting from zero, the nth pixel group is assigned to the texel with internal integer coordinates $[i]$, where

$$i = x + (n \bmod w)$$

3.8.3 Texture Parameters

Various parameters control how the texture array is treated when applied to a fragment. Each parameter is set by calling

```
void TexParameter{if}( enum target, enum pname,
    T param );
void TexParameter{if}v( enum target, enum pname,
    T params );
```


Microsoft Corp. Exhibit 1009

Name	Type	Legal Values
TEXTURE_WRAP_S	integer	CLAMP, CLAMP_TO_EDGE, REPEAT
TEXTURE_WRAP_T	integer	CLAMP, CLAMP_TO_EDGE, REPEAT
TEXTURE_WRAP_R	integer	CLAMP, CLAMP_TO_EDGE, REPEAT
TEXTURE_MIN_FILTER	integer	NEAREST, LINEAR, NEAREST_MIPMAP_NEAREST, NEAREST_MIPMAP_LINEAR, LINEAR_MIPMAP_NEAREST, LINEAR_MIPMAP_LINEAR,
TEXTURE_MAG_FILTER	integer	NEAREST, LINEAR
TEXTURE_BORDER_COLOR	4 floats	any 4 values in $[0, 1]$
TEXTURE_PRIORITY	float	any value in $[0, 1]$
TEXTURE_MIN_LOD	float	any value
TEXTURE_MAX_LOD	float	any value
TEXTURE_BASE_LEVEL	integer	any non-negative integer
TEXTURE_MAX_LEVEL	integer	any non-negative integer

Table 3.17: Texture parameters and their values.

target is the target, either TEXTURE_1D, TEXTURE_2D, or TEXTURE_3D. *pname* is a symbolic constant indicating the parameter to be set; the possible constants and corresponding parameters are summarized in table 3.17. In the first form of the command, *param* is a value to which to set a single-valued parameter; in the second form of the command, *params* is an array of parameters whose type depends on the parameter being set. If the values for TEXTURE_BORDER_COLOR are specified as integers, the conversion for signed integers from table 2.6 is applied to convert the values to floating-point. Each of the four values set by TEXTURE_BORDER_COLOR is clamped to lie in $[0, 1]$.

3.8.4 Texture Wrap Modes

If TEXTURE_WRAP_S, TEXTURE_WRAP_T, or TEXTURE_WRAP_R is set to REPEAT, then the GL ignores the integer part of s, t, or r coordinates, respectively, using only the fractional part. (For a number f, the fractional part is $f - \lfloor f \rfloor$, regardless of the sign of f; recall that the *floor* function truncates towards $-\infty$.) CLAMP causes s, t, or r coordinates to be clamped to the range $[0, 1]$.

Microsoft Corp. Exhibit 1009

The initial state is for all of s, t, and r behavior to be that given by `REPEAT`.

`CLAMP_TO_EDGE` clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel. The color returned when clamping is derived only from texels at the edge of the texture image.

Texture coordinates are clamped to the range $[min, max]$. The minimum value is defined as

$$min = \frac{1}{2N}$$

where N is the size of the one-, two-, or three-dimensional texture image in the direction of clamping. The maximum value is defined as

$$max = 1 - min$$

so that clamping is always symmetric about the $[0, 1]$ mapped range of a texture coordinate.

3.8.5 Texture Minification

Applying a texture to a primitive implies a mapping from texture image space to framebuffer image space. In general, this mapping involves a reconstruction of the sampled texture image, followed by a homogeneous warping implied by the mapping to framebuffer space, then a filtering, followed finally by a resampling of the filtered, warped, reconstructed image before applying it to a fragment. In the GL this mapping is approximated by one of two simple filtering schemes. One of these schemes is selected based on whether the mapping from texture space to framebuffer space is deemed to *magnify* or *minify* the texture image.

Scale Factor and Level of Detail

The choice is governed by a scale factor $\rho(x, y)$ and the *level of detail* parameter $\lambda(x, y)$, defined as

$$\lambda'(x, y) = \log_2[\rho(x, y)]$$

$$\lambda = \begin{cases} \texttt{TEXTURE_MAX_LOD}, & \lambda' > \texttt{TEXTURE_MAX_LOD} \\ \lambda', & \texttt{TEXTURE_MIN_LOD} \leq \lambda' \leq \texttt{TEXTURE_MAX_LOD} \\ \texttt{TEXTURE_MIN_LOD}, & \lambda' < \texttt{TEXTURE_MIN_LOD} \\ undefined, & \texttt{TEXTURE_MIN_LOD} > \texttt{TEXTURE_MAX_LOD} \end{cases} \tag{3.14}$$

Microsoft Corp. Exhibit 1009

If $\lambda(x,y)$ is less than or equal to the constant c (described below in section 3.8.6) the texture is said to be magnified; if it is greater, the texture is minified.

The initial values of `TEXTURE_MIN_LOD` and `TEXTURE_MAX_LOD` are chosen so as to never clamp the normal range of λ. They may be respecified for a specific texture by calling **TexParameter[if]**.

Let $s(x,y)$ be the function that associates an s texture coordinate with each set of window coordinates (x,y) that lie within a primitive; define $t(x,y)$ and $r(x,y)$ analogously. Let $u(x,y) = 2^n s(x,y)$, $v(x,y) = 2^m t(x,y)$, and $w(x,y) = 2^l r(x,y)$, where n, m, and l are as defined by equations 3.11, 3.12, and 3.13 with w_s, h_s, and d_s equal to the width, height, and depth of the image array whose level is `TEXTURE_BASE_LEVEL`. For a one-dimensional texture, define $v(x,y) \equiv 0$ and $w(x,y) \equiv 0$; for a two-dimensional texture, define $w(x,y) \equiv 0$. For a polygon, ρ is given at a fragment with window coordinates (x,y) by

$$\rho = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2}\right\} \tag{3.15}$$

where $\partial u/\partial x$ indicates the derivative of u with respect to window x, and similarly for the other derivatives.

For a line, the formula is

$$\rho = \sqrt{\left(\frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y\right)^2 + \left(\frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y\right)^2 + \left(\frac{\partial w}{\partial x}\Delta x + \frac{\partial w}{\partial y}\Delta y\right)^2} \Big/ l, \tag{3.16}$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$ with (x_1, y_1) and (x_2, y_2) being the segment's window coordinate endpoints and $l = \sqrt{\Delta x^2 + \Delta y^2}$. For a point, pixel rectangle, or bitmap, $\rho \equiv 1$.

While it is generally agreed that equations 3.15 and 3.16 give the best results when texturing, they are often impractical to implement. Therefore, an implementation may approximate the ideal ρ with a function $f(x,y)$ subject to these conditions:

1. $f(x,y)$ is continuous and monotonically increasing in each of $|\partial u/\partial x|$, $|\partial u/\partial y|$, $|\partial v/\partial x|$, $|\partial v/\partial y|$, $|\partial w/\partial x|$, and $|\partial w/\partial y|$

2. Let
$$m_u = \max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|\right\}$$

Microsoft Corp. Exhibit 1009

$$m_v = \max\left\{\left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial v}{\partial y}\right|\right\}$$

$$m_w = \max\left\{\left|\frac{\partial w}{\partial x}\right|, \left|\frac{\partial w}{\partial y}\right|\right\}.$$

Then $\max\{m_u, m_v, m_w\} \leq f(x,y) \leq m_u + m_v + m_w$.

When λ indicates minification, the value assigned to `TEXTURE_MIN_FILTER` is used to determine how the texture value for a fragment is selected. When `TEXTURE_MIN_FILTER` is `NEAREST`, the texel in the image array of level `TEXTURE_BASE_LEVEL` that is nearest (in Manhattan distance) to that specified by (s,t,r) is obtained. This means the texel at location (i,j,k) becomes the texture value, with i given by

$$i = \begin{cases} \lfloor u \rfloor, & s < 1 \\ 2^n - 1, & s = 1 \end{cases} \tag{3.17}$$

(Recall that if `TEXTURE_WRAP_S` is `REPEAT`, then $0 \leq s < 1$.) Similarly, j is found as

$$j = \begin{cases} \lfloor v \rfloor, & t < 1 \\ 2^m - 1, & t = 1 \end{cases} \tag{3.18}$$

and k is found as

$$k = \begin{cases} \lfloor w \rfloor, & r < 1 \\ 2^l - 1, & r = 1 \end{cases} \tag{3.19}$$

For a one-dimensional texture, j and k are irrelevant; the texel at location i becomes the texture value. For a two-dimensional texture, k is irrelevant; the texel at location (i,j) becomes the texture value.

When `TEXTURE_MIN_FILTER` is `LINEAR`, a $2 \times 2 \times 2$ cube of texels in the image array of level `TEXTURE_BASE_LEVEL` is selected. This cube is obtained by first clamping texture coordinates as described above under **Texture Wrap Modes** (if the wrap mode for a coordinate is `CLAMP` or `CLAMP_TO_EDGE`) and computing

$$i_0 = \begin{cases} \lfloor u - 1/2 \rfloor \bmod 2^n, & \texttt{TEXTURE_WRAP_S} \text{ is } \texttt{REPEAT} \\ \lfloor u - 1/2 \rfloor, & otherwise \end{cases}$$

Microsoft Corp. Exhibit 1009

$$j_0 = \begin{cases} \lfloor v - 1/2 \rfloor \bmod 2^m, & \texttt{TEXTURE_WRAP_T} \text{ is } \texttt{REPEAT} \\ \lfloor v - 1/2 \rfloor, & otherwise \end{cases}$$

and

$$k_0 = \begin{cases} \lfloor w - 1/2 \rfloor \bmod 2^l, & \texttt{TEXTURE_WRAP_R} \text{ is } \texttt{REPEAT} \\ \lfloor w - 1/2 \rfloor, & otherwise \end{cases}$$

Then

$$i_1 = \begin{cases} (i_0 + 1) \bmod 2^n, & \texttt{TEXTURE_WRAP_S} \text{ is } \texttt{REPEAT} \\ i_0 + 1, & otherwise \end{cases}$$

$$j_1 = \begin{cases} (j_0 + 1) \bmod 2^m, & \texttt{TEXTURE_WRAP_T} \text{ is } \texttt{REPEAT} \\ j_0 + 1, & otherwise \end{cases}$$

and

$$k_1 = \begin{cases} (k_0 + 1) \bmod 2^l, & \texttt{TEXTURE_WRAP_R} \text{ is } \texttt{REPEAT} \\ k_0 + 1, & otherwise \end{cases}$$

Let

$$\alpha = \text{frac}(u - 1/2)$$

$$\beta = \text{frac}(v - 1/2)$$

$$\gamma = \text{frac}(w - 1/2)$$

where $\text{frac}(x)$ denotes the fractional part of x.

For a three-dimensional texture, the texture value τ is found as

$$\begin{aligned} \tau \quad = \quad & (1-\alpha)(1-\beta)(1-\gamma)\tau_{i_0 j_0 k_0} + \alpha(1-\beta)(1-\gamma)\tau_{i_1 j_0 k_0} \\ & + (1-\alpha)\beta(1-\gamma)\tau_{i_0 j_1 k_0} + \alpha\beta(1-\gamma)\tau_{i_1 j_1 k_0} \\ & + (1-\alpha)(1-\beta)\gamma\tau_{i_0 j_0 k_1} + \alpha(1-\beta)\gamma\tau_{i_1 j_0 k_1} \\ & + (1-\alpha)\beta\gamma\tau_{i_0 j_1 k_1} + \alpha\beta\gamma\tau_{i_1 j_1 k_1} \end{aligned}$$

where τ_{ijk} is the texel at location (i, j, k) in the three-dimensional texture image.

For a two-dimensional texture,

Microsoft Corp. Exhibit 1009

$$\tau = (1-\alpha)(1-\beta)\tau_{i_0 j_0} + \alpha(1-\beta)\tau_{i_1 j_0} + (1-\alpha)\beta\tau_{i_0 j_1} + \alpha\beta\tau_{i_1 j_1} \quad (3.20)$$

where τ_{ij} is the texel at location (i, j) in the two-dimensional texture image.

And for a one-dimensional texture,

$$\tau = (1-\alpha)\tau_{i_0} + \alpha\tau_{i_1}$$

where τ_i is the texel at location i in the one-dimensional texture.

If any of the selected τ_{ijk}, τ_{ij}, or τ_i in the above equations refer to a border texel with $i < -b_s$, $j < -b_s$, $k < -b_s$, $i \geq w_s - b_s$, $j \geq h_s - b_s$, or $j \geq d_s - b_s$, then the border color given by the current setting of `TEXTURE_BORDER_COLOR` is used instead of the unspecified value or values. The RGBA values of the `TEXTURE_BORDER_COLOR` are interpreted to match the texture's internal format in a manner consistent with table 3.15.

Mipmapping

`TEXTURE_MIN_FILTER` values `NEAREST_MIPMAP_NEAREST`, `NEAREST_MIPMAP_LINEAR`, `LINEAR_MIPMAP_NEAREST`, and `LINEAR_MIPMAP_LINEAR` each require the use of a *mipmap*. A mipmap is an ordered set of arrays representing the same image; each array has a resolution lower than the previous one. If the image array of level `TEXTURE_BASE_LEVEL`, excluding its border, has dimensions $2^n \times 2^m \times 2^l$, then there are $\max\{n, m, l\} + 1$ image arrays in the mipmap. Each array subsequent to the array of level `TEXTURE_BASE_LEVEL` has dimensions

$$\sigma(i-1) \times \sigma(j-1) \times \sigma(k-1)$$

where the dimensions of the previous array are

$$\sigma(i) \times \sigma(j) \times \sigma(k)$$

and

$$\sigma(x) = \begin{cases} 2^x & x > 0 \\ 1 & x \leq 0 \end{cases}$$

until the last array is reached with dimension $1 \times 1 \times 1$.

Each array in a mipmap is defined using **TexImage3D**, **TexImage2D**, **CopyTexImage2D**, **TexImage1D**, or **CopyTexImage1D**; the array being set is indicated with the level-of-detail argument *level*. Level-of-detail numbers proceed from `TEXTURE_BASE_LEVEL` for the original texture array

Microsoft Corp. Exhibit 1009

through $p = \max\{n, m, l\}$ + TEXTURE_BASE_LEVEL with each unit increase indicating an array of half the dimensions of the previous one as already described. If texturing is enabled (and TEXTURE_MIN_FILTER is one that requires a mipmap) at the time a primitive is rasterized and if the set of arrays TEXTURE_BASE_LEVEL through $q = \min\{p, \text{TEXTURE_MAX_LEVEL}\}$ is incomplete, then it is as if texture mapping were disabled. The set of arrays TEXTURE_BASE_LEVEL through q is incomplete if the internal formats of all the mipmap arrays were not specified with the same symbolic constant, if the border widths of the mipmap arrays are not the same, if the dimensions of the mipmap arrays do not follow the sequence described above, if TEXTURE_MAX_LEVEL $<$ TEXTURE_BASE_LEVEL, or if TEXTURE_BASE_LEVEL $> p$. Array levels k where $k <$ TEXTURE_BASE_LEVEL or $k > q$ are insignificant.

The values of TEXTURE_BASE_LEVEL and TEXTURE_MAX_LEVEL may be respecified for a specific texture by calling **TexParameter[if]**. The error INVALID_VALUE is generated if either value is negative.

The mipmap is used in conjunction with the level of detail to approximate the application of an appropriately filtered texture to a fragment. Let c be the value of λ at which the transition from minification to magnification occurs (since this discussion pertains to minification, we are concerned only with values of λ where $\lambda > c$). In the following equations, let

$b =$ TEXTURE_BASE_LEVEL

For mipmap filters NEAREST_MIPMAP_NEAREST and LINEAR_MIPMAP_NEAREST, the dth mipmap array is selected, where

$$d = \begin{cases} b, & \lambda \leq \frac{1}{2} \\ \lceil b + \lambda + \frac{1}{2} \rceil - 1, & \lambda > \frac{1}{2}, b + \lambda \leq q + \frac{1}{2} \\ q, & \lambda > \frac{1}{2}, b + \lambda > q + \frac{1}{2} \end{cases} \tag{3.21}$$

The rules for NEAREST or LINEAR filtering are then applied to the selected array.

For mipmap filters NEAREST_MIPMAP_LINEAR and LINEAR_MIPMAP_LINEAR, the level d_1 and d_2 mipmap arrays are selected, where

$$d_1 = \begin{cases} q, & b + \lambda \geq q \\ \lfloor b + \lambda \rfloor, & otherwise \end{cases} \tag{3.22}$$

$$d_2 = \begin{cases} q, & b + \lambda \geq q \\ d_1 + 1, & otherwise \end{cases} \tag{3.23}$$

The rules for NEAREST or LINEAR filtering are then applied to each of the selected arrays, yielding two corresponding texture values τ_1 and τ_2. The final texture value is then found as

Microsoft Corp. Exhibit 1009

$$\tau = [1 - \mathrm{frac}(\lambda)]\tau_1 + \mathrm{frac}(\lambda)\tau_2.$$

3.8.6 Texture Magnification

When λ indicates magnification, the value assigned to TEXTURE_MAG_FILTER determines how the texture value is obtained. There are two possible values for TEXTURE_MAG_FILTER: NEAREST and LINEAR. NEAREST behaves exactly as NEAREST for TEXTURE_MIN_FILTER (equations 3.17, 3.18, and 3.19 are used); LINEAR behaves exactly as LINEAR for TEXTURE_MIN_FILTER (equation 3.20 is used). The level-of-detail TEXTURE_BASE_LEVEL texture array is always used for magnification.

Finally, there is the choice of c, the minification vs. magnification switchover point. If the magnification filter is given by LINEAR and the minification filter is given by NEAREST_MIPMAP_NEAREST or NEAREST_MIPMAP_LINEAR, then $c = 0.5$. This is done to ensure that a minified texture does not appear "sharper" than a magnified texture. Otherwise $c = 0$.

3.8.7 Texture State and Proxy State

The state necessary for texture can be divided into two categories. First, there are the three sets of mipmap arrays (one-, two-, and three-dimensional) and their number. Each array has associated with it a width, height (two- or three-dimensional only), and depth (three-dimensional only), a border width, an integer describing the internal format of the image, and six integer values describing the resolutions of each of the red, green, blue, alpha, luminance, and intensity components of the image. Each initial texture array is null (zero width, height, and depth, zero border width, internal format 1, with zero-sized components). Next, there are the two sets of texture properties; each consists of the selected minification and magnification filters, the wrap modes for s, t (two- and three-dimensional only), and r (three-dimensional only), the TEXTURE_BORDER_COLOR, two integers describing the minimum and maximum level of detail, two integers describing the base and maximum mipmap array, a boolean flag indicating whether the texture is resident and the priority associated with each set of properties. The value of the resident flag is determined by the GL and may change as a result of other GL operations. The flag may only be queried, not set, by applications. See section 3.8.8). In the initial state, the value assigned to TEXTURE_MIN_FILTER is NEAREST_MIPMAP_LINEAR, and the value for TEXTURE_MAG_FILTER is LINEAR. s, t, and r wrap modes are all set to REPEAT.

Microsoft Corp. Exhibit 1009

The values of TEXTURE_MIN_LOD and TEXTURE_MAX_LOD are -1000 and 1000 respectively. The values of TEXTURE_BASE_LEVEL and TEXTURE_MAX_LEVEL are 0 and 1000 respectively. TEXTURE_PRIORITY is 1.0, and TEXTURE_BORDER_COLOR is (0,0,0,0). The initial value of TEXTURE_RESIDENT is determined by the GL.

In addition to the one-, two-, and three-dimensional sets of image arrays, partially instantiated one-, two-, and three-dimensional sets of proxy image arrays are maintained. Each proxy array includes width, height (two- and three-dimensional arrays only), depth (three-dimensional arrays only), border width, and internal format state values, as well as state for the red, green, blue, alpha, luminance, and intensity component resolutions. Proxy arrays do not include image data, nor do they include texture properties. When **TexImage3D** is executed with *target* specified as PROXY_TEXTURE_3D, the three-dimensional proxy state values of the specified level-of-detail are recomputed and updated. If the image array would not be supported by **TexImage3D** called with *target* set to TEXTURE_3D, no error is generated, but the proxy width, height, depth, border width, and component resolutions are set to zero. If the image array would be supported by such a call to **TexImage3D**, the proxy state values are set exactly as though the actual image array were being specified. No pixel data are transferred or processed in either case.

One- and two-dimensional proxy arrays are operated on in the same way when **TexImage1D** is executed with *target* specified as PROXY_TEXTURE_1D, or **TexImage2D** is executed with *target* specified as PROXY_TEXTURE_2D.

There is no image associated with any of the proxy textures. Therefore PROXY_TEXTURE_1D, PROXY_TEXTURE_2D, and PROXY_TEXTURE_3D cannot be used as textures, and their images must never be queried using **GetTexImage**. The error INVALID_ENUM is generated if this is attempted. Likewise, there is no nonlevel-related state associated with a proxy texture, and **GetTexParameteriv** or **GetTexParameterfv** may not be called with a proxy texture *target*. The error INVALID_ENUM is generated if this is attempted.

3.8.8 Texture Objects

In addition to the default textures TEXTURE_1D, TEXTURE_2D, and TEXTURE_3D named one-, two-, and three-dimensional texture objects can be created and operated upon. The name space for texture objects is the unsigned integers, with zero reserved by the GL.

A texture object is created by *binding* an unused name to TEXTURE_1D, TEXTURE_2D, or TEXTURE_3D. The binding is effected by calling

Microsoft Corp. Exhibit 1009

`void` **BindTexture**(`enum` *target*, `uint` *texture*);

with *target* set to the desired texture target and *texture* set to the unused name. The resulting texture object is a new state vector, comprising all the state values listed in section 3.8.7, set to the same initial values. If the new texture object is bound to `TEXTURE_1D`, `TEXTURE_2D`, or `TEXTURE_3D` respectively, it is and remains a one-, two-, or three-dimensional texture until it is deleted.

BindTexture may also be used to bind an existing texture object to either `TEXTURE_1D`, `TEXTURE_2D`, or `TEXTURE_3D`. The error `INVALID_OPERATION` is generated if an attempt is made to bind a texture object of different dimensionality than the specified *target*. If the bind is successful no change is made to the state of the bound texture object, and any previous binding to *target* is broken.

While a texture object is bound, GL operations on the target to which it is bound affect the bound object, and queries of the target to which it is bound return state from the bound object. If texture mapping of the dimensionality of the target to which a texture object is bound is enabled, the state of the bound texture object directs the texturing operation.

In the initial state, `TEXTURE_1D`, `TEXTURE_2D`, and `TEXTURE_3D` have one-, two-, and three-dimensional texture state vectors associated with them. In order that access to these initial textures not be lost, they are treated as texture objects all of whose names are 0. The initial one-, two-, or three-dimensional texture is therefore operated upon, queried, and applied as `TEXTURE_1D`, `TEXTURE_2D`, or `TEXTURE_3D` respectively while 0 is bound to the corresponding targets.

Texture objects are deleted by calling

`void` **DeleteTextures**(`sizei` *n*, `uint` **textures*);

textures contains *n* names of texture objects to be deleted. After a texture object is deleted, it has no contents or dimensionality, and its name is again unused. If a texture that is currently bound to one of the targets `TEXTURE_1D`, `TEXTURE_2D`, or `TEXTURE_3D` is deleted, it is as though **BindTexture** had been executed with the same *target* and *texture* zero. Unused names in *textures* are silently ignored, as is the value zero.

The command

`void` **GenTextures**(`sizei` *n*, `uint` **textures*);

Microsoft Corp. Exhibit 1009

returns n previously unused texture object names in *textures*. These names are marked as used, for the purposes of **GenTextures** only, but they acquire texture state and a dimensionality only when they are first bound, just as if they were unused.

An implementation may choose to establish a working set of texture objects on which binding operations are performed with higher performance. A texture object that is currently part of the working set is said to be *resident*. The command

```
boolean AreTexturesResident( sizei n, uint *textures,
    boolean *residences );
```

returns `TRUE` if all of the n texture objects named in *textures* are resident, or if the implementation does not distinguish a working set. If at least one of the texture objects named in *textures* is not resident, then `FALSE` is returned, and the residence of each texture object is returned in *residences*. Otherwise the contents of *residences* are not changed. If any of the names in *textures* are unused or are zero, `FALSE` is returned, the error `INVALID_VALUE` is generated, and the contents of *residences* are indeterminate. The residence status of a single bound texture object can also be queried by calling **GetTexParameteriv** or **GetTexParameterfv** with *target* set to the target to which the texture object is bound, and *pname* set to `TEXTURE_RESIDENT`.

AreTexturesResident indicates only whether a texture object is currently resident, not whether it could not be made resident. An implementation may choose to make a texture object resident only on first use, for example. The client may guide the GL implementation in determining which texture objects should be resident by specifying a priority for each texture object. The command

```
void PrioritizeTextures( sizei n, uint *textures,
    clampf *priorities );
```

sets the priorities of the n texture objects named in *textures* to the values in *priorities*. Each priority value is clamped to the range [0,1] before it is assigned. Zero indicates the lowest priority, with the least likelihood of being resident. One indicates the highest priority, with the greatest likelihood of being resident. The priority of a single bound texture object may also be changed by calling **TexParameteri**, **TexParameterf**, **TexParameteriv**, or **TexParameterfv** with *target* set to the target to which the texture object is bound, *pname* set to `TEXTURE_PRIORITY`, and *param* or *params*

Microsoft Corp. Exhibit 1009

specifying the new priority value (which is clamped to the range [0,1] before being assigned). **PrioritizeTextures** silently ignores attempts to prioritize unused texture object names or zero (default textures).

3.8.9 Texture Environments and Texture Functions

The command

```
void TexEnv{if}( enum target, enum pname, T param );
void TexEnv{if}v( enum target, enum pname, T params );
```

sets parameters of the *texture environment* that specifies how texture values are interpreted when texturing a fragment. *target* must currently be the symbolic constant TEXTURE_ENV. *pname* is a symbolic constant indicating the parameter to be set. In the first form of the command, *param* is a value to which to set a single-valued parameter; in the second form, *params* is a pointer to an array of parameters: either a single symbolic constant or a value or group of values to which the parameter should be set. The possible environment parameters are TEXTURE_ENV_MODE and TEXTURE_ENV_COLOR. TEXTURE_ENV_MODE may be set to one of REPLACE, MODULATE, DECAL, or BLEND; TEXTURE_ENV_COLOR is set to an RGBA color by providing four single-precision floating-point values in the range $[0, 1]$ (values outside this range are clamped to it). If integers are provided for TEXTURE_ENV_COLOR, then they are converted to floating-point as specified in table 2.6 for signed integers.

The value of TEXTURE_ENV_MODE specifies a *texture function*. The result of this function depends on the fragment and the texture array value. The precise form of the function depends on the base internal formats of the texture arrays that were last specified. In the following two tables, R_f, G_f, B_f, and A_f are the primary color components of the incoming fragment; R_t, G_t, B_t, A_t, L_t, and I_t are the filtered texture values; R_c, G_c, B_c, and A_c are the texture environment color values; and R_v, G_v, B_v, and A_v are the primary color components computed by the texture function. All of these color values are in the range $[0, 1]$. The REPLACE and MODULATE texture functions are specified in table 3.18, and the DECAL and BLEND texture functions are specified in table 3.19.

The state required for the current texture environment consists of the four-valued integer indicating the texture function and four floating-point TEXTURE_ENV_COLOR values. In the initial state, the texture function is given by MODULATE and TEXTURE_ENV_COLOR is $(0, 0, 0, 0)$.

Microsoft Corp. Exhibit 1009

APPENDIX T

Base Internal Format	REPLACE Texture Function	MODULATE Texture Function
ALPHA	$R_v = R_f$ $G_v = G_f$ $B_v = B_f$ $A_v = A_t$	$R_v = R_f$ $G_v = G_f$ $B_v = B_f$ $A_v = A_f A_t$
LUMINANCE (or 1)	$R_v = L_t$ $G_v = L_t$ $B_v = L_t$ $A_v = A_f$	$R_v = R_f L_t$ $G_v = G_f L_t$ $B_v = B_f L_t$ $A_v = A_f$
LUMINANCE_ALPHA (or 2)	$R_v = L_t$ $G_v = L_t$ $B_v = L_t$ $A_v = A_t$	$R_v = R_f L_t$ $G_v = G_f L_t$ $B_v = B_f L_t$ $A_v = A_f A_t$
INTENSITY	$R_v = I_t$ $G_v = I_t$ $B_v = I_t$ $A_v = I_t$	$R_v = R_f I_t$ $G_v = G_f I_t$ $B_v = B_f I_t$ $A_v = A_f I_t$
RGB (or 3)	$R_v = R_t$ $G_v = G_t$ $B_v = B_t$ $A_v = A_f$	$R_v = R_f R_t$ $G_v = G_f G_t$ $B_v = B_f B_t$ $A_v = A_f$
RGBA (or 4)	$R_v = R_t$ $G_v = G_t$ $B_v = B_t$ $A_v = A_t$	$R_v = R_f R_t$ $G_v = G_f G_t$ $B_v = B_f B_t$ $A_v = A_f A_t$

Table 3.18: Replace and modulate texture functions.

Microsoft Corp. Exhibit 1009

Base Internal Format	DECAL Texture Function	BLEND Texture Function
ALPHA	undefined	$R_v = R_f$ $G_v = G_f$ $B_v = B_f$ $A_v = A_f A_t$
LUMINANCE (or 1)	undefined	$R_v = R_f(1 - L_t) + R_c L_t$ $G_v = G_f(1 - L_t) + G_c L_t$ $B_v = B_f(1 - L_t) + B_c L_t$ $A_v = A_f$
LUMINANCE_ALPHA (or 2)	undefined	$R_v = R_f(1 - L_t) + R_c L_t$ $G_v = G_f(1 - L_t) + G_c L_t$ $B_v = B_f(1 - L_t) + B_c L_t$ $A_v = A_f A_t$
INTENSITY	undefined	$R_v = R_f(1 - I_t) + R_c I_t$ $G_v = G_f(1 - I_t) + G_c I_t$ $B_v = B_f(1 - I_t) + B_c I_t$ $A_v = A_f(1 - I_t) + A_c I_t$
RGB (or 3)	$R_v = R_t$ $G_v = G_t$ $B_v = B_t$ $A_v = A_f$	$R_v = R_f(1 - R_t) + R_c R_t$ $G_v = G_f(1 - G_t) + G_c G_t$ $B_v = B_f(1 - B_t) + B_c B_t$ $A_v = A_f$
RGBA (or 4)	$R_v = R_f(1 - A_t) + R_t A_t$ $G_v = G_f(1 - A_t) + G_t A_t$ $B_v = B_f(1 - A_t) + B_t A_t$ $A_v = A_f$	$R_v = R_f(1 - R_t) + R_c R_t$ $G_v = G_f(1 - G_t) + G_c G_t$ $B_v = B_f(1 - B_t) + B_c B_t$ $A_v = A_f A_t$

Table 3.19: Decal and blend texture functions.

Microsoft Corp. Exhibit 1009

3.8.10 Texture Application

Texturing is enabled or disabled using the generic **Enable** and **Disable** commands, respectively, with the symbolic constants `TEXTURE_1D`, `TEXTURE_2D`, or `TEXTURE_3D` to enable the one-, two-, or three-dimensional texture, respectively. If both two- and one-dimensional textures are enabled, the two-dimensional texture is used. If the three-dimensional and either of the two- or one-dimensional textures is enabled, the three-dimensional texture is used. If all texturing is disabled, a rasterized fragment is passed on unaltered to the next stage of the GL (although its texture coordinates may be discarded). Otherwise, a texture value is found according to the parameter values of the currently bound texture image of the appropriate dimensionality using the rules given in sections 3.8.5 and 3.8.6. This texture value is used along with the incoming fragment in computing the texture function indicated by the currently bound texture environment. The result of this function replaces the incoming fragment's primary R, G, B, and A values. These are the color values passed to subsequent operations. Other data associated with the incoming fragment remain unchanged, except that the texture coordinates may be discarded.

The required state is three bits indicating whether each of one-, two-, or three-dimensional texturing is enabled or disabled. In the initial state, all texturing is disabled.

3.9 Color Sum

At the beginning of color sum, a fragment has two RGBA colors: a primary color $\mathbf{c}_{pri}$ (which texturing, if enabled, may have modified) and a secondary color $\mathbf{c}_{sec}$. The components of these two colors are summed to produce a single post-texturing RGBA color $\mathbf{c}$. The components of $\mathbf{c}$ are then clamped to the range $[0, 1]$.

Color sum has no effect in color index mode.

3.10 Fog

If enabled, fog blends a fog color with a rasterized fragment's post-texturing color using a blending factor f. Fog is enabled and disabled with the **Enable** and **Disable** commands using the symbolic constant `FOG`.

This factor f is computed according to one of three equations:

$$f = \exp(-d \cdot z), \tag{3.24}$$

Microsoft Corp. Exhibit 1009

$$f = \exp(-(d \cdot z)^2), \text{ or} \tag{3.25}$$

$$f = \frac{e - z}{e - s} \tag{3.26}$$

(z is the eye-coordinate distance from the eye, $(0, 0, 0, 1)$ in eye coordinates, to the fragment center). The equation, along with either d or e and s, is specified with

`void` **Fog{if}**(`enum` *pname*, `T` *param*);
`void` **Fog{if}v**(`enum` *pname*, `T` *params*);

If *pname* is `FOG_MODE`, then *param* must be, or *params* must point to an integer that is one of the symbolic constants `EXP`, `EXP2`, or `LINEAR`, in which case equation 3.24, 3.25, or 3.26, respectively, is selected for the fog calculation (if, when 3.26 is selected, $e = s$, results are undefined). If *pname* is `FOG_DENSITY`, `FOG_START`, or `FOG_END`, then *param* is or *params* points to a value that is d, s, or e, respectively. If d is specified less than zero, the error `INVALID_VALUE` results.

An implementation may choose to approximate the eye-coordinate distance from the eye to each fragment center by $|z_e|$. Further, f need not be computed at each fragment, but may be computed at each vertex and interpolated as other data are.

No matter which equation and approximation is used to compute f, the result is clamped to $[0, 1]$ to obtain the final f.

f is used differently depending on whether the GL is in RGBA or color index mode. In RGBA mode, if C_r represents a rasterized fragment's R, G, or B value, then the corresponding value produced by fog is

$$C = fC_r + (1 - f)C_f.$$

(The rasterized fragment's A value is not changed by fog blending.) The R, G, B, and A values of C_f are specified by calling **Fog** with *pname* equal to `FOG_COLOR`; in this case *params* points to four values comprising C_f. If these are not floating-point values, then they are converted to floating-point using the conversion given in table 2.6 for signed integers. Each component of C_f is clamped to $[0, 1]$ when specified.

In color index mode, the formula for fog blending is

$$I = i_r + (1 - f)i_f$$

where i_r is the rasterized fragment's color index and i_f is a single-precision floating-point value. $(1 - f)i_f$ is rounded to the nearest fixed-point value

Microsoft Corp. Exhibit 1009

APPENDIX T

with the same number of bits to the right of the binary point as i_r, and the integer portion of I is masked (bitwise ANDed) with $2^n - 1$, where n is the number of bits in a color in the color index buffer (buffers are discussed in chapter 4). The value of i_f is set by calling **Fog** with *pname* set to `FOG_INDEX` and *param* being or *params* pointing to a single value for the fog index. The integer part of i_f is masked with $2^n - 1$.

The state required for fog consists of a three valued integer to select the fog equation, three floating-point values d, e, and s, an RGBA fog color and a fog color index, and a single bit to indicate whether or not fog is enabled. In the initial state, fog is disabled, `FOG_MODE` is `EXP`, $d = 1.0$, $e = 1.0$, and $s = 0.0$; $C_f = (0, 0, 0, 0)$ and $i_f = 0$.

3.11 Antialiasing Application

Finally, if antialiasing is enabled for the primitive from which a rasterized fragment was produced, then the computed coverage value is applied to the fragment. In RGBA mode, the value is multiplied by the fragment's alpha (A) value to yield a final alpha value. In color index mode, the value is used to set the low order bits of the color index value as described in section 3.2.

Microsoft Corp. Exhibit 1009

Chapter 4

Per-Fragment Operations and the Framebuffer

The framebuffer consists of a set of pixels arranged as a two-dimensional array. The height and width of this array may vary from one GL implementation to another. For purposes of this discussion, each pixel in the framebuffer is simply a set of some number of bits. The number of bits per pixel may also vary depending on the particular GL implementation or context.

Corresponding bits from each pixel in the framebuffer are grouped together into a *bitplane*; each bitplane contains a single bit from each pixel. These bitplanes are grouped into several *logical buffers*. These are the *color*, *depth*, *stencil*, and *accumulation* buffers. The color buffer actually consists of a number of buffers: the *front left* buffer, the *front right* buffer, the *back left* buffer, the *back right* buffer, and some number of *auxiliary* buffers. Typically the contents of the front buffers are displayed on a color monitor while the contents of the back buffers are invisible. (Monoscopic contexts display only the front left buffer; stereoscopic contexts display both the front left and the front right buffers.) The contents of the auxiliary buffers are never visible. All color buffers must have the same number of bitplanes, although an implementation or context may choose not to provide right buffers, back buffers, or auxiliary buffers at all. Further, an implementation or context may not provide depth, stencil, or accumulation buffers.

Color buffers consist of either unsigned integer color indices or R, G, B, and, optionally, A unsigned integer values. The number of bitplanes in each of the color buffers, the depth buffer, the stencil buffer, and the accumulation buffer is fixed and window dependent. If an accumulation buffer is provided,

Microsoft Corp. Exhibit 1009

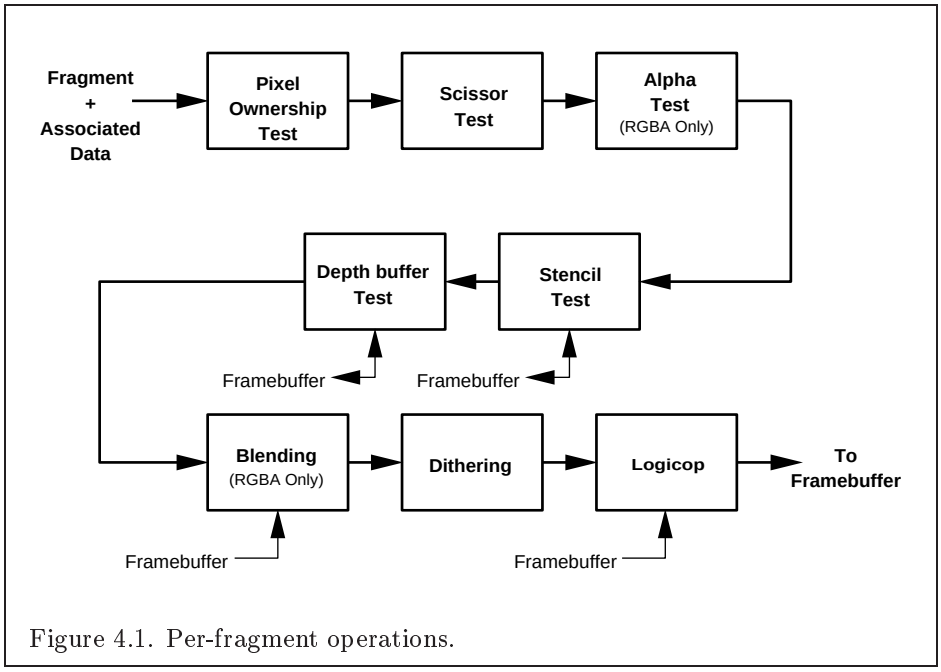


Figure 4.1. Per-fragment operations.

it must have at least as many bitplanes per R, G, and B color component as do the color buffers.

The initial state of all provided bitplanes is undefined.

4.1 Per-Fragment Operations

A fragment produced by rasterization with window coordinates of (x_w, y_w) modifies the pixel in the framebuffer at that location based on a number of parameters and conditions. We describe these modifications and tests, diagrammed in Figure 4.1, in the order in which they are performed. Figure 4.1 diagrams these modifications and tests.

4.1.1 Pixel Ownership Test

The first test is to determine if the pixel at location (x_w, y_w) in the framebuffer is currently owned by the GL (more precisely, by this GL context). If it is not, the window system decides the fate the incoming fragment. Possible results are that the fragment is discarded or that some subset of the subsequent per-fragment operations are applied to the fragment. This test

Microsoft Corp. Exhibit 1009

allows the window system to control the GL's behavior, for instance, when a GL window is obscured.

4.1.2 Scissor test

The scissor test determines if (x_w, y_w) lies within the scissor rectangle defined by four values. These values are set with

```
void Scissor( int left, int bottom, sizei width,
    sizei height );
```

If $left \leq x_w < left + width$ and $bottom \leq y_w < bottom + height$, then the scissor test passes. Otherwise, the test fails and the fragment is discarded. The test is enabled or disabled using **Enable** or **Disable** using the constant `SCISSOR_TEST`. When disabled, it is as if the scissor test always passes. If either *width* or *height* is less than zero, then the error `INVALID_VALUE` is generated. The state required consists of four integer values and a bit indicating whether the test is enabled or disabled. In the initial state $left = bottom = 0$; *width* and *height* are determined by the size of the GL window. Initially, the scissor test is disabled.

4.1.3 Alpha test

This step applies only in RGBA mode. In color index mode, proceed to the next step. The alpha test discards a fragment conditional on the outcome of a comparison between the incoming fragment's alpha value and a constant value. The comparison is enabled or disabled with the generic **Enable** and **Disable** commands using the symbolic constant `ALPHA_TEST`. When disabled, it is as if the comparison always passes. The test is controlled with

```
void AlphaFunc( enum func, clampf ref );
```

func is a symbolic constant indicating the alpha test function; *ref* is a reference value. *ref* is clamped to lie in $[0, 1]$, and then converted to a fixed-point value according to the rules given for an A component in section 2.13.9. For purposes of the alpha test, the fragment's alpha value is also rounded to the nearest integer. The possible constants specifying the test function are `NEVER`, `ALWAYS`, `LESS`, `LEQUAL`, `EQUAL`, `GEQUAL`, `GREATER`, or `NOTEQUAL`, meaning pass the fragment never, always, if the fragment's alpha value is less than, less than or equal to, equal to, greater than or equal to, greater than, or not equal to the reference value, respectively.

Microsoft Corp. Exhibit 1009

The required state consists of the floating-point reference value, an eight-valued integer indicating the comparison function, and a bit indicating if the comparison is enabled or disabled. The initial state is for the reference value to be 0 and the function to be ALWAYS. Initially, the alpha test is disabled.

4.1.4 Stencil test

The stencil test conditionally discards a fragment based on the outcome of a comparison between the value in the stencil buffer at location (x_w, y_w) and a reference value. The test is controlled with

```
void StencilFunc( enum func, int ref, uint mask );
void StencilOp( enum sfail, enum dpfail, enum dppass );
```

The test is enabled or disabled with the **Enable** and **Disable** commands, using the symbolic constant STENCIL_TEST. When disabled, the stencil test and associated modifications are not made, and the fragment is always passed.

ref is an integer reference value that is used in the unsigned stencil comparison. It is clamped to the range $[0, 2^s - 1]$, where s is the number of bits in the stencil buffer. *func* is a symbolic constant that determines the stencil comparison function; the eight symbolic constants are NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GEQUAL, GREATER, or NOTEQUAL. Accordingly, the stencil test passes never, always, if the reference value is less than, less than or equal to, equal to, greater than or equal to, greater than, or not equal to the masked stored value in the stencil buffer. The s least significant bits of *mask* are bitwise ANDed with both the reference and the stored stencil value. The ANDed values are those that participate in the comparison.

StencilOp takes three arguments that indicate what happens to the stored stencil value if this or certain subsequent tests fail or pass. *sfail* indicates what action is taken if the stencil test fails. The symbolic constants are KEEP, ZERO, REPLACE, INCR, DECR, and INVERT. These correspond to keeping the current value, setting it to zero, replacing it with the reference value, incrementing it, decrementing it, or bitwise inverting it. For purposes of increment and decrement, the stencil bits are considered as an unsigned integer; values clamp at 0 and the maximum representable value. The same symbolic values are given to indicate the stencil action if the depth buffer test (below) fails (*dpfail*), or if it passes (*dppass*).

If the stencil test fails, the incoming fragment is discarded. The state required consists of the most recent values passed to **StencilFunc** and **StencilOp**, and a bit indicating whether stencil testing is enabled or disabled.

Microsoft Corp. Exhibit 1009

In the initial state, stenciling is disabled, the stencil reference value is zero, the stencil comparison function is ALWAYS, and the stencil *mask* is all ones. Initially, all three stencil operations are KEEP. If there is no stencil buffer, no stencil modification can occur, and it is as if the stencil tests always pass, regardless of any calls to **StencilOp**.

4.1.5 Depth buffer test

The depth buffer test discards the incoming fragment if a depth comparison fails. The comparison is enabled or disabled with the generic **Enable** and **Disable** commands using the symbolic constant DEPTH_TEST. When disabled, the depth comparison and subsequent possible updates to the depth buffer value are bypassed and the fragment is passed to the next operation. The stencil value, however, is modified as indicated below as if the depth buffer test passed. If enabled, the comparison takes place and the depth buffer and stencil value may subsequently be modified.

The comparison is specified with

```
void DepthFunc( enum func );
```

This command takes a single symbolic constant: one of NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GREATER, GEQUAL, NOTEQUAL. Accordingly, the depth buffer test passes never, always, if the incoming fragment's z_w value is less than, less than or equal to, equal to, greater than, greater than or equal to, or not equal to the depth value stored at the location given by the incoming fragment's (x_w, y_w) coordinates.

If the depth buffer test fails, the incoming fragment is discarded. The stencil value at the fragment's (x_w, y_w) coordinates is updated according to the function currently in effect for depth buffer test failure. Otherwise, the fragment continues to the next operation and the value of the depth buffer at the fragment's (x_w, y_w) location is set to the fragment's z_w value. In this case the stencil value is updated according to the function currently in effect for depth buffer test success.

The necessary state is an eight-valued integer and a single bit indicating whether depth buffering is enabled or disabled. In the initial state the function is LESS and the test is disabled.

If there is no depth buffer, it is as if the depth buffer test always passes.

Microsoft Corp. Exhibit 1009

4.1.6 Blending

Blending combines the incoming fragment's R, G, B, and A values with the R, G, B, and A values stored in the framebuffer at the incoming fragment's (x_w, y_w) location.

This blending is dependent on the incoming fragment's alpha value and that of the corresponding currently stored pixel. Blending applies only in RGBA mode; in color index mode it is bypassed. Blending is enabled or disabled using **Enable** or **Disable** with the symbolic constant `BLEND`. If it is disabled, or if logical operation on color values is enabled (section 4.1.8), proceed to the next stage.

In the following discussion, C_s refers to the source color for an incoming fragment, C_d refers to the destination color at the corresponding framebuffer location, and C_c refers to a constant color in the GL state. Individual RGBA components of these colors are denoted by subscripts of s, d, and c respectively.

Destination (framebuffer) components are taken to be fixed-point values represented according to the scheme given in section 2.13.9 (Final Color Processing), as are source (fragment) components. Constant color components are taken to be floating point values.

Prior to blending, each fixed-point color component undergoes an implied conversion to floating point. This conversion must leave the values 0 and 1 invariant. Blending computations are treated as if carried out in floating point.

The commands that control blending are

```
void BlendColor( clampf red, clampf green, clampf blue,
   clampf alpha );
void BlendEquation( enum mode );

void BlendFunc( enum src, enum dst );
```

Using BlendColor

The constant color C_c to be used in blending is specified with **BlendColor**. The four parameters are clamped to the range $[0, 1]$ before being stored. The constant color can be used in both the source and destination blending factors.

BlendColor is an imaging subset feature (see section 3.6.2), and is only allowed when the imaging subset is supported.

Microsoft Corp. Exhibit 1009

Using BlendEquation

Blending capability is defined by the *blend equation*. **BlendEquation** *mode* `FUNC_ADD` defines the blending equation as

$$C = C_s S + C_d D$$

where C_s and C_d are the source and destination colors, and S and D are quadruplets of weighting factors as specified by **BlendFunc**.

If *mode* is `FUNC_SUBTRACT`, the blending equation is defined as

$$C = C_s S - C_d D$$

If *mode* is `FUNC_REVERSE_SUBTRACT`, the blending equation is defined as

$$C = C_d D - C_s S$$

If *mode* is `MIN`, the blending equation is defined as

$$C = min(C_s, C_d)$$

Finally, if *mode* is `MAX`, the blending equation is defined as

$$C = max(C_s, C_d)$$

The blending equation is evaluated separately for each color component and the corresponding weighting factors.

BlendEquation is an imaging subset feature (see section 3.6.2). If the imaging subset is not available, then blending always uses the blending equation `FUNC_ADD`.

Using BlendFunc

BlendFunc *src* indicates how to compute a source blending factor, while *dst* indicates how to compute a destination factor. The possible arguments and their corresponding computed source and destination factors are summarized in Tables 4.1 and 4.2. Addition or subtraction of quadruplets means adding or subtracting them component-wise.

The computed source and destination blending quadruplets are applied to the source and destination R, G, B, and A values to obtain a new set of values that are sent to the next operation. Let the source and destination blending quadruplets be S and D, respectively. Then a quadruplet of values is computed using the blend equation specified by **BlendEquation**. Each

Microsoft Corp. Exhibit 1009

Value	Blend Factors
ZERO	$(0, 0, 0, 0)$
ONE	$(1, 1, 1, 1)$
DST_COLOR	(R_d, G_d, B_d, A_d)
ONE_MINUS_DST_COLOR	$(1, 1, 1, 1) - (R_d, G_d, B_d, A_d)$
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)$
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)$
CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1, 1) - (R_c, G_c, B_c, A_c)$
CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)
ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1, 1) - (A_c, A_c, A_c, A_c)$
SRC_ALPHA_SATURATE	$(f, f, f, 1)$

Table 4.1: Values controlling the source blending function and the source blending values they compute. $f = \min(A_s, 1 - A_d)$.

Value	Blend factors
ZERO	$(0, 0, 0, 0)$
ONE	$(1, 1, 1, 1)$
SRC_COLOR	(R_s, G_s, B_s, A_s)
ONE_MINUS_SRC_COLOR	$(1, 1, 1, 1) - (R_s, G_s, B_s, A_s)$
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)$
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)$
CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1, 1) - (R_c, G_c, B_c, A_c)$
CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)
ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1, 1) - (A_c, A_c, A_c, A_c)$

Table 4.2: Values controlling the destination blending function and the destination blending values they compute.

Microsoft Corp. Exhibit 1009

floating-point value in this quadruplet is clamped to $[0, 1]$ and converted back to a fixed-point value in the manner described in section 2.13.9. The resulting four values are sent to the next operation.

BlendFunc arguments `CONSTANT_COLOR`, `ONE_MINUS_CONSTANT_COLOR`, `CONSTANT_ALPHA`, and `ONE_MINUS_CONSTANT_ALPHA` are imaging subset features (see section 3.6.2), and are only allowed when the imaging subset is provided.

Blending State

The state required for blending is an integer indicating the blending equation, two integers indicating the source and destination blending functions, four floating-point values to store the RGBA constant blend color, and a bit indicating whether blending is enabled or disabled. The initial blending equation is `FUNC_ADD`. The initial blending functions are `ONE` for the source function and `ZERO` for the destination function. The initial constant blend color is $(\mathrm{R}, \mathrm{G}, \mathrm{B}, \mathrm{A}) = (0, 0, 0, 0)$. Initially, blending is disabled.

Blending occurs once for each color buffer currently enabled for writing (section 4.2.1) using each buffer's color for C_d. If a color buffer has no A value, then A_d is taken to be 1.

4.1.7 Dithering

Dithering selects between two color values or indices. In RGBA mode, consider the value of any of the color components as a fixed-point value with m bits to the left of the binary point, where m is the number of bits allocated to that component in the framebuffer; call each such value c. For each c, dithering selects a value c_1 such that $c_1 \in \{\max\{0, \lceil c \rceil - 1\}, \lceil c \rceil\}$ (after this selection, treat c_1 as a fixed point value in [0,1] with m bits). This selection may depend on the x_w and y_w coordinates of the pixel. In color index mode, the same rule applies with c being a single color index. c must not be larger than the maximum value representable in the framebuffer for either the component or the index, as appropriate.

Many dithering algorithms are possible, but a dithered value produced by any algorithm must depend only the incoming value and the fragment's x and y window coordinates. If dithering is disabled, then each color component is truncated to a fixed-point value with as many bits as there are in the corresponding component in the framebuffer; a color index is rounded to the nearest integer representable in the color index portion of the framebuffer.

Dithering is enabled with **Enable** and disabled with **Disable** using the

Microsoft Corp. Exhibit 1009

symbolic constant DITHER. The state required is thus a single bit. Initially, dithering is enabled.

4.1.8 Logical Operation

Finally, a logical operation is applied between the incoming fragment's color or index values and the color or index values stored at the corresponding location in the framebuffer. The result replaces the values in the framebuffer at the fragment's (x, y) coordinates. The logical operation on color indices is enabled or disabled with **Enable** or **Disable** using the symbolic constant INDEX_LOGIC_OP. (For compatibility with GL version 1.0, the symbolic constant LOGIC_OP may also be used.) The logical operation on color values is enabled or disabled with **Enable** or **Disable** using the symbolic constant COLOR_LOGIC_OP. If the logical operation is enabled for color values, it is as if blending were disabled, regardless of the value of BLEND.

The logical operation is selected by

```
void LogicOp( enum op );
```

op is a symbolic constant; the possible constants and corresponding operations are enumerated in Table 4.3. In this table, s is the value of the incoming fragment and d is the value stored in the framebuffer. The numeric values assigned to the symbolic constants are the same as those assigned to the corresponding symbolic values in the X window system.

Logical operations are performed independently for each color index buffer that is selected for writing, or for each red, green, blue, and alpha value of each color buffer that is selected for writing. The required state is an integer indicating the logical operation, and two bits indicating whether the logical operation is enabled or disabled. The initial state is for the logic operation to be given by COPY, and to be disabled.

4.2 Whole Framebuffer Operations

The preceding sections described the operations that occur as individual fragments are sent to the framebuffer. This section describes operations that control or affect the whole framebuffer.

4.2.1 Selecting a Buffer for Writing

The first such operation is controlling the buffer into which color values are written. This is accomplished with

Microsoft Corp. Exhibit 1009

Argument value	Operation
CLEAR	0
AND	$s \wedge d$
AND_REVERSE	$s \wedge \neg d$
COPY	s
AND_INVERTED	$\neg s \wedge d$
NOOP	d
XOR	s xor d
OR	$s \vee d$
NOR	$\neg(s \vee d)$
EQUIV	$\neg(s$ xor $d)$
INVERT	$\neg d$
OR_REVERSE	$s \vee \neg d$
COPY_INVERTED	$\neg s$
OR_INVERTED	$\neg s \vee d$
NAND	$\neg(s \wedge d)$
SET	all 1's

Table 4.3: Arguments to **LogicOp** and their corresponding operations.

`void` **DrawBuffer**(`enum` *buf*);

buf is a symbolic constant specifying zero, one, two, or four buffers for writing. The constants are NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, and AUX0 through AUXn, where $n+1$ is the number of available auxiliary buffers.

The constants refer to the four potentially visible buffers *front_left*, *front_right*, *back_left*, and *back_right*, and to the *auxiliary* buffers. Arguments other than AUXi that omit reference to LEFT or RIGHT refer to both left and right buffers. Arguments other than AUXi that omit reference to FRONT or BACK refer to both front and back buffers. AUXi enables drawing only to *auxiliary* buffer i. Each AUXi adheres to AUXi = AUX0 + i. The constants and the buffers they indicate are summarized in Table 4.4. If **DrawBuffer** is is supplied with a constant (other than NONE) that does not indicate any of the color buffers allocated to the GL context, the error INVALID_OPERATION results.

Indicating a buffer or buffers using **DrawBuffer** causes subsequent pixel color value writes to affect the indicated buffers. If more than one color buffer is selected for drawing, blending and logical operations are computed

Microsoft Corp. Exhibit 1009

symbolic constant	front left	front right	back left	back right	aux i
NONE					
FRONT_LEFT	•				
FRONT_RIGHT		•			
BACK_LEFT			•		
BACK_RIGHT				•	
FRONT	•	•			
BACK			•	•	
LEFT	•		•		
RIGHT		•		•	
FRONT_AND_BACK	•	•	•	•	
AUXi					•

Table 4.4: Arguments to **DrawBuffer** and the buffers that they indicate.

and applied independently for each buffer. Calling **DrawBuffer** with a value of NONE inhibits the writing of color values to any buffer.

Monoscopic contexts include only left buffers, while stereoscopic contexts include both left and right buffers. Likewise, single buffered contexts include only front buffers, while double buffered contexts include both front and back buffers. The type of context is selected at GL initialization.

The state required to handle buffer selection is a set of up to $4 + n$ bits. 4 bits indicate if the front left buffer, the front right buffer, the back left buffer, or the back right buffer, are enabled for color writing. The other n bits indicate which of the auxiliary buffers is enabled for color writing. In the initial state, the front buffer or buffers are enabled if there are no back buffers; otherwise, only the back buffer or buffers are enabled.

4.2.2 Fine Control of Buffer Updates

Four commands are used to mask the writing of bits to each of the logical framebuffers after all per-fragment operations have been performed. The commands

```
void IndexMask( uint mask );
void ColorMask( boolean r, boolean g, boolean b,
   boolean a );
```


Microsoft Corp. Exhibit 1009

control the color buffer or buffers (depending on which buffers are currently indicated for writing). The least significant n bits of *mask*, where n is the number of bits in a color index buffer, specify a mask. Where a 1 appears in this mask, the corresponding bit in the color index buffer (or buffers) is written; where a 0 appears, the bit is not written. This mask applies only in color index mode. In RGBA mode, **ColorMask** is used to mask the writing of R, G, B and A values to the color buffer or buffers. r, g, b, and a indicate whether R, G, B, or A values, respectively, are written or not (a value of `TRUE` means that the corresponding value is written). In the initial state, all bits (in color index mode) and all color values (in RGBA mode) are enabled for writing.

The depth buffer can be enabled or disabled for writing z_w values using

`void` **DepthMask**(`boolean` *mask*);

If *mask* is non-zero, the depth buffer is enabled for writing; otherwise, it is disabled. In the initial state, the depth buffer is enabled for writing.

The command

`void` **StencilMask**(`uint` *mask*);

controls the writing of particular bits into the stencil planes. The least significant s bits of *mask* comprise an integer mask (s is the number of bits in the stencil buffer), just as for **IndexMask**. The initial state is for the stencil plane mask to be all ones.

The state required for the various masking operations is two integers and a bit: an integer for color indices, an integer for stencil values, and a bit for depth values. A set of four bits is also required indicating which color components of an RGBA value should be written. In the initial state, the integer masks are all ones as are the bits controlling depth value and RGBA component writing.

4.2.3 Clearing the Buffers

The GL provides a means for setting portions of every pixel in a particular buffer to the same value. The argument to

`void` **Clear**(`bitfield` *buf*);

is the bitwise OR of a number of values indicating which buffers are to be cleared. The values are `COLOR_BUFFER_BIT`, `DEPTH_BUFFER_BIT`,

Microsoft Corp. Exhibit 1009

STENCIL_BUFFER_BIT, and ACCUM_BUFFER_BIT, indicating the buffers currently enabled for color writing, the depth buffer, the stencil buffer, and the accumulation buffer (see below), respectively. The value to which each buffer is cleared depends on the setting of the clear value for that buffer. If the mask is not a bitwise OR of the specified values, then the error INVALID_VALUE is generated.

```
void ClearColor( clampf r, clampf g, clampf b,
   clampf a );
```

sets the clear value for the color buffers in RGBA mode. Each of the specified components is clamped to $[0, 1]$ and converted to fixed-point according to the rules of section 2.13.9.

```
void ClearIndex( float index );
```

sets the clear color index. *index* is converted to a fixed-point value with unspecified precision to the left of the binary point; the integer part of this value is then masked with $2^m - 1$, where m is the number of bits in a color index value stored in the framebuffer.

```
void ClearDepth( clampd d );
```

takes a floating-point value that is clamped to the range $[0, 1]$ and converted to fixed-point according to the rules for a window z value given in section 2.10.1. Similarly,

```
void ClearStencil( int s );
```

takes a single integer argument that is the value to which to clear the stencil buffer. s is masked to the number of bitplanes in the stencil buffer.

```
void ClearAccum( float r, float g, float b, float a );
```

takes four floating-point arguments that are the values, in order, to which to set the R, G, B, and A values of the accumulation buffer (see the next section). These values are clamped to the range $[-1, 1]$ when they are specified.

When **Clear** is called, the only per-fragment operations that are applied (if enabled) are the pixel ownership test, the scissor test, and dithering. The masking operations described in the last section (4.2.2) are also effective. If a buffer is not present, then a **Clear** directed at that buffer has no effect.

Microsoft Corp. Exhibit 1009

The state required for clearing is a clear value for each of the color buffer, the depth buffer, the stencil buffer, and the accumulation buffer. Initially, the RGBA color clear value is (0,0,0,0), the clear color index is 0, and the stencil buffer and accumulation buffer clear values are all 0. The depth buffer clear value is initially 1.0.

4.2.4 The Accumulation Buffer

Each portion of a pixel in the accumulation buffer consists of four values: one for each of R, G, B, and A. The accumulation buffer is controlled exclusively through the use of

`void` **Accum**(`enum` *op*, `float` *value*);

(except for clearing it). *op* is a symbolic constant indicating an accumulation buffer operation, and *value* is a floating-point value to be used in that operation. The possible operations are `ACCUM`, `LOAD`, `RETURN`, `MULT`, and `ADD`.

When the scissor test is enabled (section 4.1.2), then only those pixels within the current scissor box are updated by any **Accum** operation; otherwise, all pixels in the window are updated. The accumulation buffer operations apply identically to every affected pixel, so we describe the effect of each operation on an individual pixel. Accumulation buffer values are taken to be signed values in the range $[-1, 1]$. Using `ACCUM` obtains R, G, B, and A components from the buffer currently selected for reading (section 4.3.2). Each component, considered as a fixed-point value in $[0, 1]$. (see section 2.13.9), is converted to floating-point. Each result is then multiplied by *value*. The results of this multiplication are then added to the corresponding color component currently in the accumulation buffer, and the resulting color value replaces the current accumulation buffer color value.

The `LOAD` operation has the same effect as `ACCUM`, but the computed values replace the corresponding accumulation buffer components rather than being added to them.

The `RETURN` operation takes each color value from the accumulation buffer, multiplies each of the R, G, B, and A components by *value*, and clamps the results to the range $[0, 1]$ The resulting color value is placed in the buffers currently enabled for color writing as if it were a fragment produced from rasterization, except that the only per-fragment operations that are applied (if enabled) are the pixel ownership test, the scissor test (section 4.1.2), and dithering (section 4.1.7). Color masking (section 4.2.2) is also applied.

Microsoft Corp. Exhibit 1009

The MULT operation multiplies each R, G, B, and A in the accumulation buffer by *value* and then returns the scaled color components to their corresponding accumulation buffer locations. ADD is the same as MULT except that *value* is added to each of the color components.

The color components operated on by **Accum** must be clamped only if the operation is RETURN. In this case, a value sent to the enabled color buffers is first clamped to $[0, 1]$. Otherwise, results are undefined if the result of an operation on a color component is out of the range $[-1, 1]$. If there is no accumulation buffer, or if the GL is in color index mode, **Accum** generates the error INVALID_OPERATION.

No state (beyond the accumulation buffer itself) is required for accumulation buffering.

4.3 Drawing, Reading, and Copying Pixels

Pixels may be written to and read from the framebuffer using the **DrawPixels** and **ReadPixels** commands. **CopyPixels** can be used to copy a block of pixels from one portion of the framebuffer to another.

4.3.1 Writing to the Stencil Buffer

The operation of **DrawPixels** was described in section 3.6.4, except if the *format* argument was STENCIL_INDEX. In this case, all operations described for **DrawPixels** take place, but window (x, y) coordinates, each with the corresponding stencil index, are produced in lieu of fragments. Each coordinate-stencil index pair is sent directly to the per-fragment operations, bypassing the texture, fog, and antialiasing application stages of rasterization. Each pair is then treated as a fragment for purposes of the pixel ownership and scissor tests; all other per-fragment operations are bypassed. Finally, each stencil index is written to its indicated location in the framebuffer, subject to the current setting of **StencilMask**.

The error INVALID_OPERATION results if there is no stencil buffer.

4.3.2 Reading Pixels

The method for reading pixels from the framebuffer and placing them in client memory is diagrammed in Figure 4.2. We describe the stages of the pixel reading process in the order in which they occur.

Pixels are read using

Microsoft Corp. Exhibit 1009

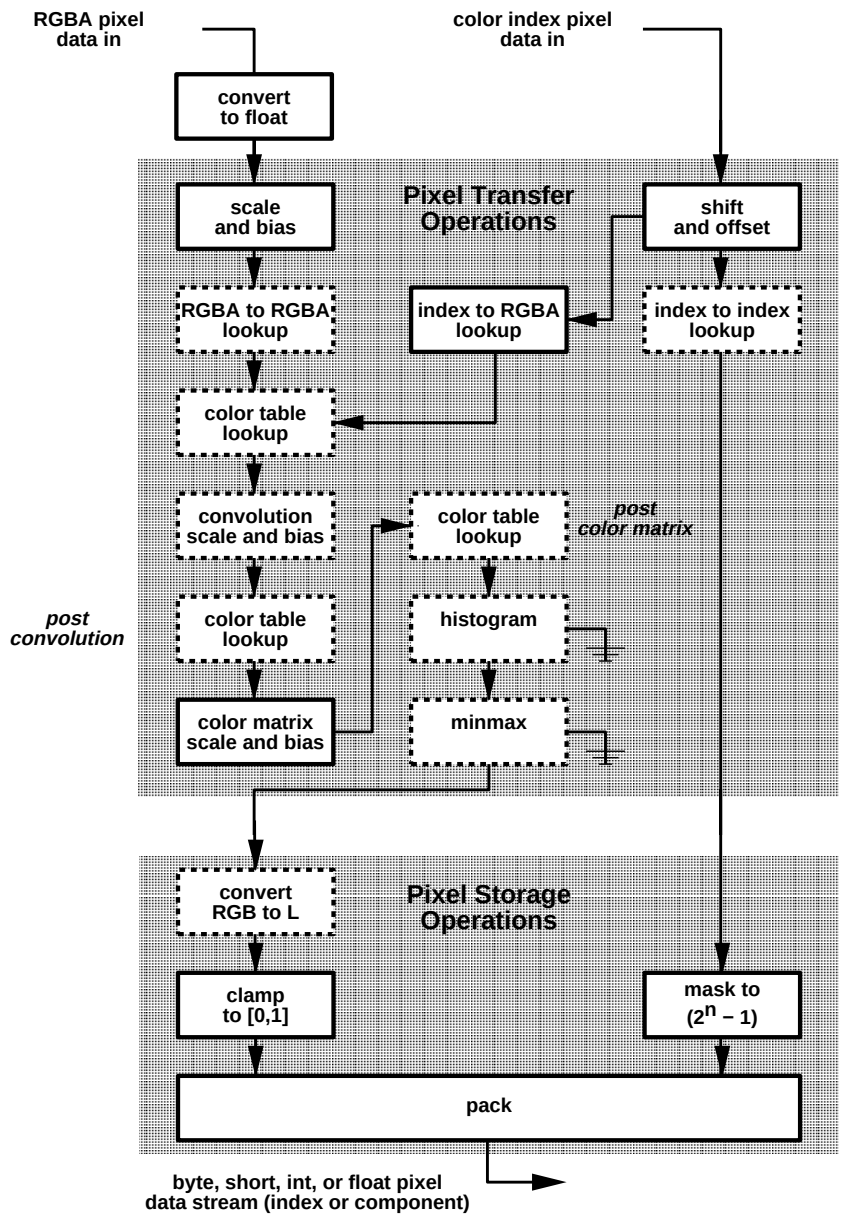


Figure 4.2. Operation of **ReadPixels**. Operations in dashed boxes may be enabled or disabled. RGBA and color index pixel paths are shown; depth and stencil pixel paths are not shown.

Microsoft Corp. Exhibit 1009

Parameter Name	Type	Initial Value	Valid Range
PACK_SWAP_BYTES	boolean	FALSE	TRUE/FALSE
PACK_LSB_FIRST	boolean	FALSE	TRUE/FALSE
PACK_ROW_LENGTH	integer	0	$[0, \infty)$
PACK_SKIP_ROWS	integer	0	$[0, \infty)$
PACK_SKIP_PIXELS	integer	0	$[0, \infty)$
PACK_ALIGNMENT	integer	4	1,2,4,8
PACK_IMAGE_HEIGHT	integer	0	$[0, \infty)$
PACK_SKIP_IMAGES	integer	0	$[0, \infty)$

Table 4.5: **PixelStore** parameters pertaining to **ReadPixels**, **GetTexImage1D**, **GetTexImage2D**, **GetTexImage3D**, **GetColorTable**, **GetConvolutionFilter**, **GetSeparableFilter**, **GetHistogram**, and **GetMinmax**.

```
void ReadPixels( int x, int y, sizei width, sizei height,
    enum format, enum type, void *data );
```

The arguments after x and y to **ReadPixels** correspond to those of **DrawPixels**. The pixel storage modes that apply to **ReadPixels** and other commands that query images (see section 6.1) are summarized in Table 4.5.

Obtaining Pixels from the Framebuffer

If the *format* is DEPTH_COMPONENT, then values are obtained from the depth buffer. If there is no depth buffer, the error INVALID_OPERATION occurs.

If the *format* is STENCIL_INDEX, then values are taken from the stencil buffer; again, if there is no stencil buffer, the error INVALID_OPERATION occurs.

For all other formats, the buffer from which values are obtained is one of the color buffers; the selection of color buffer is controlled with **ReadBuffer**.

The command

```
void ReadBuffer( enum src );
```

takes a symbolic constant as argument. The possible values are FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, and AUX0 through AUXn. FRONT and LEFT refer to the front left buffer, BACK refers to the back left buffer, and RIGHT refers to the front right buffer. The other constants correspond directly to the buffers that they name. If the requested

Microsoft Corp. Exhibit 1009

buffer is missing, then the error INVALID_OPERATION is generated. The initial setting for **ReadBuffer** is FRONT if there is no back buffer and BACK otherwise.

ReadPixels obtains values from the selected buffer from each pixel with lower left hand corner at $(x + i, y + j)$ for $0 \leq i < width$ and $0 \leq j < height$; this pixel is said to be the ith pixel in the jth row. If any of these pixels lies outside of the window allocated to the current GL context, the values obtained for those pixels are undefined. Results are also undefined for individual pixels that are not owned by the current context. Otherwise, **ReadPixels** obtains values from the selected buffer, regardless of how those values were placed there.

If the GL is in RGBA mode, and *format* is one of RED, GREEN, BLUE, ALPHA, RGB, RGBA, BGR, BGRA, LUMINANCE, or LUMINANCE_ALPHA, then red, green, blue, and alpha values are obtained from the selected buffer at each pixel location. If the framebuffer does not support alpha values then the A that is obtained is 1.0. If *format* is COLOR_INDEX and the GL is in RGBA mode then the error INVALID_OPERATION occurs. If the GL is in color index mode, and *format* is not DEPTH_COMPONENT or STENCIL_INDEX, then the color index is obtained at each pixel location.

Conversion of RGBA values

This step applies only if the GL is in RGBA mode, and then only if *format* is neither STENCIL_INDEX nor DEPTH_COMPONENT. The R, G, B, and A values form a group of elements. Each element is taken to be a fixed-point value in $[0, 1]$ with m bits, where m is the number of bits in the corresponding color component of the selected buffer (see section 2.13.9).

Conversion of Depth values

This step applies only if *format* is DEPTH_COMPONENT. An element is taken to be a fixed-point value in [0,1] with m bits, where m is the number of bits in the depth buffer (see section 2.10.1).

Pixel Transfer Operations

This step is actually the sequence of steps that was described separately in section 3.6.5. After the processing described in that section is completed, groups are processed as described in the following sections.

Microsoft Corp. Exhibit 1009

type Parameter	Index Mask
UNSIGNED_BYTE	$2^8 - 1$
BITMAP	1
BYTE	$2^7 - 1$
UNSIGNED_SHORT	$2^{16} - 1$
SHORT	$2^{15} - 1$
UNSIGNED_INT	$2^{32} - 1$
INT	$2^{31} - 1$

Table 4.6: Index masks used by **ReadPixels**. Floating point data are not masked.

Conversion to L

This step applies only to RGBA component groups, and only if the *format* is either LUMINANCE or LUMINANCE_ALPHA. A value L is computed as

$$L = R + G + B$$

where R, G, and B are the values of the R, G, and B components. The single computed L component replaces the R, G, and B components in the group.

Final Conversion

For an index, if the *type* is not FLOAT, final conversion consists of masking the index with the value given in Table 4.6; if the *type* is FLOAT, then the integer index is converted to a GL float data value.

For an RGBA color, each component is first clamped to $[0, 1]$. Then the appropriate conversion formula from table 4.7 is applied to the component.

Placement in Client Memory

Groups of elements are placed in memory just as they are taken from memory for **DrawPixels**. That is, the ith group of the jth row (corresponding to the ith pixel in the jth row) is placed in memory just where the ith group of the jth row would be taken from for **DrawPixels**. See **Unpacking** under section 3.6.4. The only difference is that the storage mode parameters whose names begin with PACK_ are used instead of those whose names begin with UNPACK_. If the *format* is RED, GREEN, BLUE, ALPHA, or LUMINANCE,

Microsoft Corp. Exhibit 1009

type Parameter	GL Data Type	Component Conversion Formula
UNSIGNED_BYTE	ubyte	$c = (2^8 - 1)f$
BYTE	byte	$c = [(2^8 - 1)f - 1]/2$
UNSIGNED_SHORT	ushort	$c = (2^{16} - 1)f$
SHORT	short	$c = [(2^{16} - 1)f - 1]/2$
UNSIGNED_INT	uint	$c = (2^{32} - 1)f$
INT	int	$c = [(2^{32} - 1)f - 1]/2$
FLOAT	float	$c = f$
UNSIGNED_BYTE_3_3_2	ubyte	$c = (2^N - 1)f$
UNSIGNED_BYTE_2_3_3_REV	ubyte	$c = (2^N - 1)f$
UNSIGNED_SHORT_5_6_5	ushort	$c = (2^N - 1)f$
UNSIGNED_SHORT_5_6_5_REV	ushort	$c = (2^N - 1)f$
UNSIGNED_SHORT_4_4_4_4	ushort	$c = (2^N - 1)f$
UNSIGNED_SHORT_4_4_4_4_REV	ushort	$c = (2^N - 1)f$
UNSIGNED_SHORT_5_5_5_1	ushort	$c = (2^N - 1)f$
UNSIGNED_SHORT_1_5_5_5_REV	ushort	$c = (2^N - 1)f$
UNSIGNED_INT_8_8_8_8	uint	$c = (2^N - 1)f$
UNSIGNED_INT_8_8_8_8_REV	uint	$c = (2^N - 1)f$
UNSIGNED_INT_10_10_10_2	uint	$c = (2^N - 1)f$
UNSIGNED_INT_2_10_10_10_REV	uint	$c = (2^N - 1)f$

Table 4.7: Reversed component conversions - used when component data are being returned to client memory. Color, normal, and depth components are converted from the internal floating-point representation (f) to a datum of the specified GL data type (c) using the equations in this table. All arithmetic is done in the internal floating point format. These conversions apply to component data returned by GL query commands and to components of pixel data returned to client memory. The equations remain the same even if the implemented ranges of the GL data types are greater than the minimum required ranges. (See Table 2.2.) Equations with N as the exponent are performed for each bitfield of the packed data type, with N set to the number of bits in the bitfield.

Microsoft Corp. Exhibit 1009

only the corresponding single element is written. Likewise if the *format* is LUMINANCE_ALPHA, RGB, or BGR, only the corresponding two or three elements are written. Otherwise all the elements of each group are written.

4.3.3 Copying Pixels

CopyPixels transfers a rectangle of pixel values from one region of the framebuffer to another. Pixel copying is diagrammed in Figure 4.3.

```
void CopyPixels( int x, int y, sizei width, sizei height,
    enum type );
```

type is a symbolic constant that must be one of COLOR, STENCIL, or DEPTH, indicating that the values to be transferred are colors, stencil values, or depth values, respectively. The first four arguments have the same interpretation as the corresponding arguments to **ReadPixels**.

Values are obtained from the framebuffer, converted (if appropriate), then subjected to the pixel transfer operations described in section 3.6.5, just as if **ReadPixels** were called with the corresponding arguments. If the *type* is STENCIL or DEPTH, then it is as if the *format* for **ReadPixels** were STENCIL_INDEX or DEPTH_COMPONENT, respectively. If the *type* is COLOR, then if the GL is in RGBA mode, it is as if the *format* were RGBA, while if the GL is in color index mode, it is as if the *format* were COLOR_INDEX.

The groups of elements so obtained are then written to the framebuffer just as if **DrawPixels** had been given *width* and *height*, beginning with final conversion of elements. The effective *format* is the same as that already described.

4.3.4 Pixel Draw/Read state

The state required for pixel operations consists of the parameters that are set with **PixelStore**, **PixelTransfer**, and **PixelMap**. This state has been summarized in Tables 3.1, 3.2, and 3.3. The current setting of **ReadBuffer**, an integer, is also required, along with the current raster position (section 2.12). State set with **PixelStore** is GL client state.

Microsoft Corp. Exhibit 1009

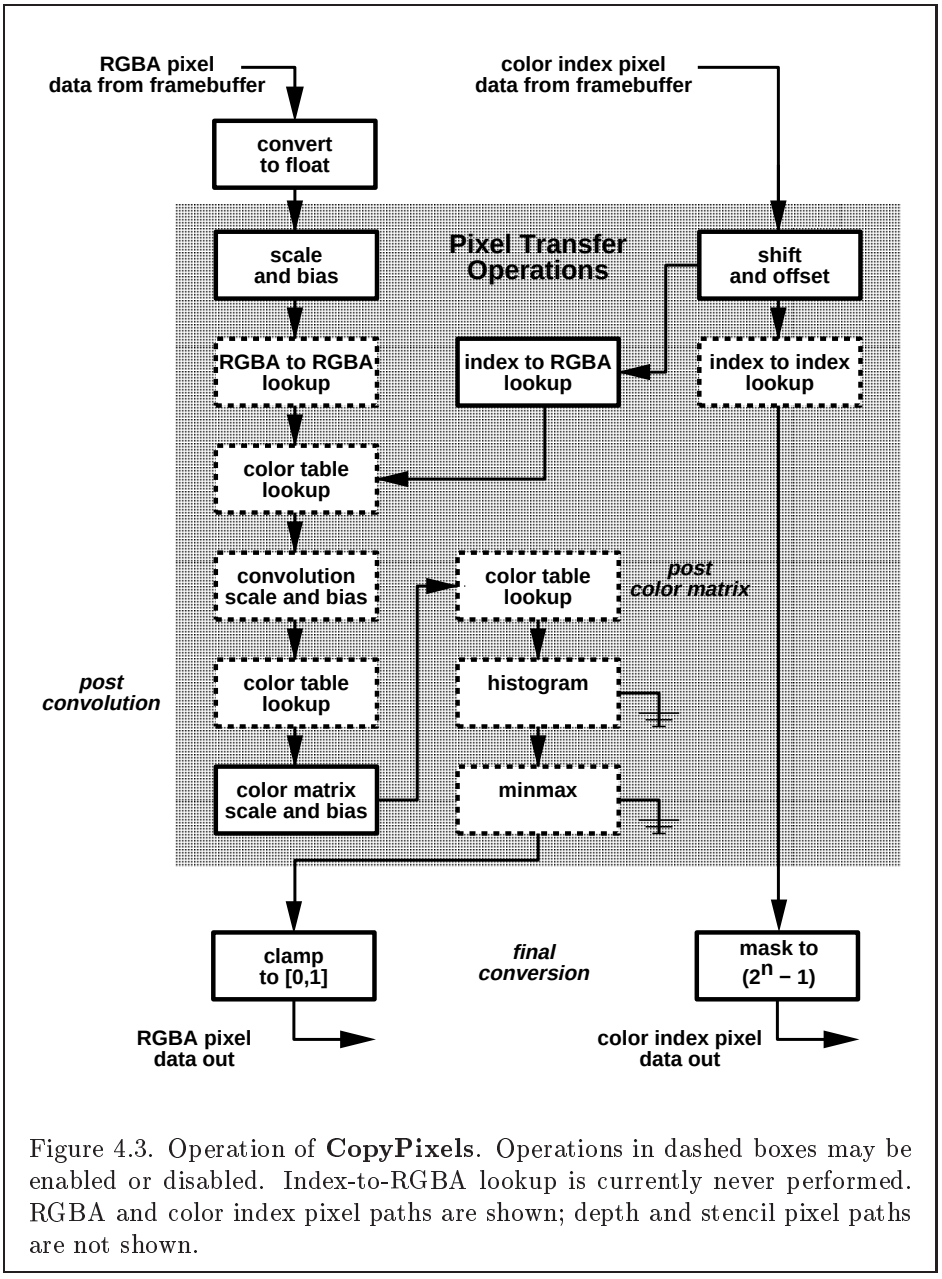


Figure 4.3. Operation of **CopyPixels**. Operations in dashed boxes may be enabled or disabled. Index-to-RGBA lookup is currently never performed. RGBA and color index pixel paths are shown; depth and stencil pixel paths are not shown.

Microsoft Corp. Exhibit 1009

Chapter 5

Special Functions

This chapter describes additional GL functionality that does not fit easily into any of the preceding chapters. This functionality consists of evaluators (used to model curves and surfaces), selection (used to locate rendered primitives on the screen), feedback (which returns GL results before rasterization), display lists (used to designate a group of GL commands for later execution by the GL), flushing and finishing (used to synchronize the GL command stream), and hints.

5.1 Evaluators

Evaluators provide a means to use a polynomial or rational polynomial mapping to produce vertex, normal, and texture coordinates, and colors. The values so produced are sent on to further stages of the GL as if they had been provided directly by the client. Transformations, lighting, primitive assembly, rasterization, and per-pixel operations are not affected by the use of evaluators.

Consider the R^k-valued polynomial $\mathbf{p}(u)$ defined by

$$\mathbf{p}(u) = \sum_{i=0}^{n} B_i^n(u)\mathbf{R}_i \tag{5.1}$$

with $\mathbf{R}_i \in R^k$ and

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i},$$

the ith Bernstein polynomial of degree n (recall that $0^0 \equiv 1$ and $\binom{n}{0} \equiv 1$). Each $\mathbf{R}_i$ is a *control point*. The relevant command is

Microsoft Corp. Exhibit 1009

target	k	Values
MAP1_VERTEX_3	3	x, y, z vertex coordinates
MAP1_VERTEX_4	4	x, y, z, w vertex coordinates
MAP1_INDEX	1	color index
MAP1_COLOR_4	4	R, G, B, A
MAP1_NORMAL	3	x, y, z normal coordinates
MAP1_TEXTURE_COORD_1	1	s texture coordinate
MAP1_TEXTURE_COORD_2	2	s, t texture coordinates
MAP1_TEXTURE_COORD_3	3	s, t, r texture coordinates
MAP1_TEXTURE_COORD_4	4	s, t, r, q texture coordinates

Table 5.1: Values specified by the *target* to **Map1**. Values are given in the order in which they are taken.

```
void Map1{fd}( enum type, T u1, T u2, int stride,
   int order, T points );
```

type is a symbolic constant indicating the range of the defined polynomial. Its possible values, along with the evaluations that each indicates, are given in Table 5.1. *order* is equal to $n + 1$; The error INVALID_VALUE is generated if *order* is less than one or greater than MAX_EVAL_ORDER. *points* is a pointer to a set of $n+1$ blocks of storage. Each block begins with k single-precision floating-point or double-precision floating-point values, respectively. The rest of the block may be filled with arbitrary data. Table 5.1 indicates how k depends on *type* and what the k values represent in each case.

stride is the number of single- or double-precision values (as appropriate) in each block of storage. The error INVALID_VALUE results if *stride* is less than k. The order of the polynomial, *order*, is also the number of blocks of storage containing control points.

u_1 and u_2 give two floating-point values that define the endpoints of the pre-image of the map. When a value u' is presented for evaluation, the formula used is

$$\mathbf{p}'(u') = \mathbf{p}(\frac{u' - u_1}{u_2 - u_1}).$$

The error INVALID_VALUE results if $u_1 = u_2$.

Map2 is analogous to **Map1**, except that it describes bivariate polyno-

Microsoft Corp. Exhibit 1009

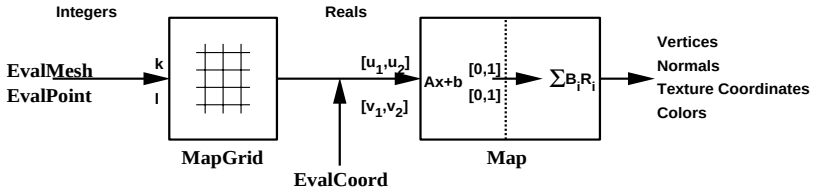


Figure 5.1. Map Evaluation.

mials of the form

$$\mathbf{p}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)\mathbf{R}_{ij}.$$

The form of the **Map2** command is

```
void Map2{fd}( enum target, T u1, T u2, int ustride,
    int uorder, T v1, T v2, int vstride, int vorder, T points );
```

target is a range type selected from the same group as is used for **Map1**, except that the string `MAP1` is replaced with `MAP2`. *points* is a pointer to $(n+1)(m+1)$ blocks of storage ($uorder = n+1$ and $vorder = m+1$; the error `INVALID_VALUE` is generated if either *uorder* or *vorder* is less than one or greater than `MAX_EVAL_ORDER`). The values comprising $\mathbf{R}_{ij}$ are located

$$(ustride)i + (vstride)j$$

values (either single- or double-precision floating-point, as appropriate) past the first value pointed to by *points*. u_1, u_2, v_1, and v_2 define the pre-image rectangle of the map; a domain point (u', v') is evaluated as

$$\mathbf{p}'(u',v') = \mathbf{p}(\frac{u'-u_1}{u_2-u_1}, \frac{v'-v_1}{v_2-v_1}).$$

The evaluation of a defined map is enabled or disabled with **Enable** and **Disable** using the constant corresponding to the map as described above. The error `INVALID_VALUE` results if either *ustride* or *vstride* is less than k, or if u_1 is equal to u_2, or if v_1 is equal to v_2.

Figure 5.1 describes map evaluation schematically; an evaluation of enabled maps is effected in one of two ways. The first way is to use

Microsoft Corp. Exhibit 1009

```
void EvalCoord{12}{fd}( T arg );
void EvalCoord{12}{fd}v( T arg );
```

EvalCoord1 causes evaluation of the enabled one-dimensional maps. The argument is the value (or a pointer to the value) that is the domain coordinate, u'. **EvalCoord2** causes evaluation of the enabled two-dimensional maps. The two values specify the two domain coordinates, u' and v', in that order.

When one of the **EvalCoord** commands is issued, all currently enabled maps of the indicated dimension are evaluated. Then, for each enabled map, it is as if a corresponding GL command were issued with the resulting coordinates, with one important difference. The difference is that when an evaluation is performed, the GL uses evaluated values instead of current values for those evaluations that are enabled (otherwise, the current values are used). The order of the effective commands is immaterial, except that **Vertex** (for vertex coordinate evaluation) must be issued last. Use of evaluators has no effect on the current color, normal, or texture coordinates. If **ColorMaterial** is enabled, evaluated color values affect the result of the lighting equation as if the current color was being modified, but no change is made to the tracking lighting parameters or to the current color.

No command is effectively issued if the corresponding map (of the indicated dimension) is not enabled. If more than one evaluation is enabled for a particular dimension (e.g. `MAP1_TEXTURE_COORD_1` and `MAP1_TEXTURE_COORD_2`), then only the result of the evaluation of the map with the highest number of coordinates is used.

Finally, if either `MAP2_VERTEX_3` or `MAP2_VERTEX_4` is enabled, then the normal to the surface is computed. Analytic computation, which sometimes yields normals of length zero, is one method which may be used. If automatic normal generation is enabled, then this computed normal is used as the normal associated with a generated vertex. Automatic normal generation is controlled with **Enable** and **Disable** with symbolic the constant `AUTO_NORMAL`. If automatic normal generation is disabled, then a corresponding normal map, if enabled, is used to produce a normal. If neither automatic normal generation nor a normal map are enabled, then no normal is sent with a vertex resulting from an evaluation (the effect is that the current normal is used).

For `MAP_VERTEX_3`, let $\mathbf{q} = \mathbf{p}$. For `MAP_VERTEX_4`, let $\mathbf{q} = (x/w, y/w, z/w)$, where $(x, y, z, w) = \mathbf{p}$. Then let

$$\mathbf{m} = \frac{\partial \mathbf{q}}{\partial u} \times \frac{\partial \mathbf{q}}{\partial v}.$$

Microsoft Corp. Exhibit 1009

Then the generated analytic normal, $\mathbf{n}$, is given by $\mathbf{n} = \mathbf{m}/\|\mathbf{m}\|$.

The second way to carry out evaluations is to use a set of commands that provide for efficient specification of a series of evenly spaced values to be mapped. This method proceeds in two steps. The first step is to define a grid in the domain. This is done using

```
void MapGrid1{fd}( int n, T u1', T u2' );
```

for a one-dimensional map or

```
void MapGrid2{fd}( int nu, T u1', T u2', int nv, T v1',
   T v2' );
```

for a two-dimensional map. In the case of **MapGrid1** u_1' and u_2' describe an interval, while n describes the number of partitions of the interval. The error `INVALID_VALUE` results if $n \leq 0$. For **MapGrid2**, (u_1', v_1') specifies one two-dimensional point and (u_2', v_2') specifies another. n_u gives the number of partitions between u_1' and u_2', and n_v gives the number of partitions between v_1' and v_2'. If either $n_u \leq 0$ or $n_v \leq 0$, then the error `INVALID_VALUE` occurs.

Once a grid is defined, an evaluation on a rectangular subset of that grid may be carried out by calling

```
void EvalMesh1( enum mode, int p1, int p2 );
```

mode is either `POINT` or `LINE`. The effect is the same as performing the following code fragment, with $\Delta u' = (u_2' - u_1')/n$:

```
Begin(type);
   for i = p1 to p2 step 1.0
      EvalCoord1(i * Δu' + u1');
End();
```

where **EvalCoord1f** or **EvalCoord1d** is substituted for **EvalCoord1** as appropriate. If *mode* is `POINT`, then *type* is `POINTS`; if *mode* is `LINE`, then *type* is `LINE_STRIP`. The one requirement is that if either $i = 0$ or $i = n$, then the value computed from $i * \Delta u' + u_1'$ is precisely u_1' or u_2', respectively.

The corresponding commands for two-dimensional maps are

```
void EvalMesh2( enum mode, int p1, int p2, int q1,
   int q2 );
```


Microsoft Corp. Exhibit 1009

mode must be `FILL`, `LINE`, or `POINT`. When *mode* is `FILL`, then these commands are equivalent to the following, with $\Delta u' = (u'_2 - u'_1)/n$ and $\Delta v' = (v'_2 - v'_1)/m$:

```
for i = q1 to q2 - 1 step 1.0
   Begin(QUAD_STRIP);
      for j = p1 to p2 step 1.0
         EvalCoord2(j * Δu' + u'1 , i * Δv' + v'1);
         EvalCoord2(j * Δu' + u'1 , (i+1) * Δv' + v'1);
   End();
```

If *mode* is `LINE`, then a call to **EvalMesh2** is equivalent to

```
for i = q1 to q2 step 1.0
   Begin(LINE_STRIP);
   for j = p1 to p2 step 1.0
      EvalCoord2(j * Δu' + u'1 , i * Δv' + v'1);
   End();;
for i = p1 to p2 step 1.0
   Begin(LINE_STRIP);
   for j = q1 to q2 step 1.0
      EvalCoord2(i * Δu' + u'1 , j * Δv' + v'1);
   End();
```

If *mode* is `POINT`, then a call to **EvalMesh2** is equivalent to

```
Begin(POINTS);
   for i = q1 to q2 step 1.0
      for j = p1 to p2 step 1.0
         EvalCoord2(j * Δu' + u'1 , i * Δv' + v'1);
End();
```

Again, in all three cases, there is the requirement that $0 * \Delta u' + u'_1 = u'_1$, $n * \Delta u' + u'_1 = u'_2$, $0 * \Delta v' + v'_1 = v'_1$, and $m * \Delta v' + v'_1 = v'_2$.

An evaluation of a single point on the grid may also be carried out:

```
void EvalPoint1( int p );
```

Calling it is equivalent to the command

```
EvalCoord1(p * Δu' + u'1);
```

with $\Delta u'$ and u'_1 defined as above.

Microsoft Corp. Exhibit 1009

APPENDIX T

`void` **EvalPoint2**(`int` p, `int` q);

is equivalent to the command

EvalCoord2(p * $\Delta u'$ + u'_1 , q * $\Delta v'$ + v'_1);

The state required for evaluators potentially consists of 9 one-dimensional map specifications and 9 two-dimensional map specifications, as well as corresponding flags for each specification indicating which are enabled. Each map specification consists of one or two orders, an appropriately sized array of control points, and a set of two values (for a one-dimensional map) or four values (for a two-dimensional map) to describe the domain. The maximum possible order, for either u or v, is implementation dependent (one maximum applies to both u and v), but must be at least 8. Each control point consists of between one and four floating-point values (depending on the type of the map). Initially, all maps have order 1 (making them constant maps). All vertex coordinate maps produce the coordinates $(0, 0, 0, 1)$ (or the appropriate subset); all normal coordinate maps produce $(0, 0, 1)$; RGBA maps produce $(1, 1, 1, 1)$; color index maps produce 1.0; texture coordinate maps produce $(0, 0, 0, 1)$; In the initial state, all maps are disabled. A flag indicates whether or not automatic normal generation is enabled for two-dimensional maps. In the initial state, automatic normal generation is disabled. Also required are two floating-point values and an integer number of grid divisions for the one-dimensional grid specification and four floating-point values and two integer grid divisions for the two-dimensional grid specification. In the initial state, the bounds of the domain interval for 1-D is 0 and 1.0, respectively; for 2-D, they are $(0, 0)$ and $(1.0, 1.0)$, respectively. The number of grid divisions is 1 for 1-D and 1 in both directions for 2-D. If any evaluation command is issued when no vertex map is enabled, nothing happens.

5.2 Selection

Selection is used by a programmer to determine which primitives are drawn into some region of a window. The region is defined by the current model-view and perspective matrices.

Selection works by returning an array of integer-valued *names*. This array represents the current contents of the *name stack*. This stack is controlled with the commands

Microsoft Corp. Exhibit 1009

```
void InitNames( void );
void PopName( void );
void PushName( uint name );
void LoadName( uint name );
```

InitNames empties (clears) the name stack. **PopName** pops one name off the top of the name stack. **PushName** causes *name* to be pushed onto the name stack. **LoadName** replaces the value on the top of the stack with *name*. Loading a name onto an empty stack generates the error `INVALID_OPERATION`. Popping a name off of an empty stack generates `STACK_UNDERFLOW`; pushing a name onto a full stack generates `STACK_OVERFLOW`. The maximum allowable depth of the name stack is implementation dependent but must be at least 64.

In selection mode, no fragments are rendered into the framebuffer. The GL is placed in selection mode with

```
int RenderMode( enum mode );
```

mode is a symbolic constant: one of `RENDER`, `SELECT`, or `FEEDBACK`. `RENDER` is the default, corresponding to rendering as described until now. `SELECT` specifies selection mode, and `FEEDBACK` specifies feedback mode (described below). Use of any of the name stack manipulation commands while the GL is not in selection mode has no effect.

Selection is controlled using

```
void SelectBuffer( sizei n, uint *buffer );
```

buffer is a pointer to an array of unsigned integers (called the selection array) to be potentially filled with names, and n is an integer indicating the maximum number of values that can be stored in that array. Placing the GL in selection mode before **SelectBuffer** has been called results in an error of `INVALID_OPERATION` as does calling **SelectBuffer** while in selection mode.

In selection mode, if a point, line, polygon, or the valid coordinates produced by a **RasterPos** command intersects the clip volume (section 2.11) then this primitive (or **RasterPos** command) causes a selection *hit*. In the case of polygons, no hit occurs if the polygon would have been culled, but selection is based on the polygon itself, regardless of the setting of **PolygonMode**. When in selection mode, whenever a name stack manipulation command is executed or **RenderMode** is called and there has been a hit since the last time the stack was manipulated or **RenderMode** was called, then a *hit record* is written into the selection array.

Microsoft Corp. Exhibit 1009

A hit record consists of the following items in order: a non-negative integer giving the number of elements on the name stack at the time of the hit, a minimum depth value, a maximum depth value, and the name stack with the bottommost element first. The minimum and maximum depth values are the minimum and maximum taken over all the window coordinate z values of each (post-clipping) vertex of each primitive that intersects the clipping volume since the last hit record was written. The minimum and maximum (each of which lies in the range $[0, 1]$) are each multiplied by $2^{32} - 1$ and rounded to the nearest unsigned integer to obtain the values that are placed in the hit record. No depth offset arithmetic (section 3.5.5) is performed on these values.

Hit records are placed in the selection array by maintaining a pointer into that array. When selection mode is entered, the pointer is initialized to the beginning of the array. Each time a hit record is copied, the pointer is updated to point at the array element after the one into which the topmost element of the name stack was stored. If copying the hit record into the selection array would cause the total number of values to exceed n, then as much of the record as fits in the array is written and an overflow flag is set.

Selection mode is exited by calling **RenderMode** with an argument value other than `SELECT`. Whenever **RenderMode** is called in selection mode, it returns the number of hit records copied into the selection array and resets the **SelectBuffer** pointer to its last specified value. Values are not guaranteed to be written into the selection array until **RenderMode** is called. If the selection array overflow flag was set, then **RenderMode** returns -1 and clears the overflow flag. The name stack is cleared and the stack pointer reset whenever **RenderMode** is called.

The state required for selection consists of the address of the selection array and its maximum size, the name stack and its associated pointer, a minimum and maximum depth value, and several flags. One flag indicates the current **RenderMode** value. In the initial state, the GL is in the `RENDER` mode. Another flag is used to indicate whether or not a hit has occurred since the last name stack manipulation. This flag is reset upon entering selection mode and whenever a name stack manipulation takes place. One final flag is required to indicate whether the maximum number of copied names would have been exceeded. This flag is reset upon entering selection mode. This flag, the address of the selection array, and its maximum size are GL client state.

Microsoft Corp. Exhibit 1009

5.3 Feedback

Feedback, like selection, is a GL mode. The mode is selected by calling **RenderMode** with `FEEDBACK`. When the GL is in feedback mode, no fragments are written to the framebuffer. Instead, information about primitives that would have been rasterized is fed back to the application using the GL.

Feedback is controlled using

`void` **FeedbackBuffer**(`sizei` *n*, `enum` *type*, `float` **buffer*);

buffer is a pointer to an array of floating-point values into which feedback information will be placed, and *n* is a number indicating the maximum number of values that can be written to that array. *type* is a symbolic constant describing the information to be fed back for each vertex (see Figure 5.2). The error `INVALID_OPERATION` results if the GL is placed in feedback mode before a call to **FeedbackBuffer** has been made, or if a call to **FeedbackBuffer** is made while in feedback mode.

While in feedback mode, each primitive that would be rasterized (or bitmap or call to **DrawPixels** or **CopyPixels**, if the raster position is valid) generates a block of values that get copied into the feedback array. If doing so would cause the number of entries to exceed the maximum, the block is partially written so as to fill the array (if there is any room left at all). The first block of values generated after the GL enters feedback mode is placed at the beginning of the feedback array, with subsequent blocks following. Each block begins with a code indicating the primitive type, followed by values that describe the primitive's vertices and associated data. Entries are also written for bitmaps and pixel rectangles. Feedback occurs after polygon culling (section 3.5.1) and **PolygonMode** interpretation of polygons (section 3.5.4) has taken place. It may also occur after polygons with more than three edges are broken up into triangles (if the GL implementation renders polygons by performing this decomposition). x, y, and z coordinates returned by feedback are window coordinates; if w is returned, it is in clip coordinates. No depth offset arithmetic (section 3.5.5) is performed on the z values. In the case of bitmaps and pixel rectangles, the coordinates returned are those of the current raster position.

The texture coordinates and colors returned are these resulting from the clipping operations described in Section 2.13.8. The colors returned are the primary colors.

The ordering rules for GL command interpretation also apply in feedback mode. Each command must be fully interpreted and its effects on both GL

Microsoft Corp. Exhibit 1009

Type	coordinates	color	texture	total values
2D	x, y	–	–	2
3D	x, y, z	–	–	3
3D_COLOR	x, y, z	k	–	$3 + k$
3D_COLOR_TEXTURE	x, y, z	k	4	$7 + k$
4D_COLOR_TEXTURE	x, y, z, w	k	4	$8 + k$

Table 5.2: Correspondence of feedback type to number of values per vertex. k is 1 in color index mode and 4 in RGBA mode.

state and the values to be written to the feedback buffer completed before a subsequent command may be executed.

The GL is taken out of feedback mode by calling **RenderMode** with an argument value other than FEEDBACK. When called while in feedback mode, **RenderMode** returns the number of values placed in the feedback array and resets the feedback array pointer to be *buffer*. The return value never exceeds the maximum number of values passed to **FeedbackBuffer**.

If writing a value to the feedback buffer would cause more values to be written than the specified maximum number of values, then the value is not written and an overflow flag is set. In this case, **RenderMode** returns -1 when it is called, after which the overflow flag is reset. While in feedback mode, values are not guaranteed to be written into the feedback buffer before **RenderMode** is called.

Figure 5.2 gives a grammar for the array produced by feedback. Each primitive is indicated with a unique identifying value followed by some number of vertices. A vertex is fed back as some number of floating-point values determined by the feedback *type*. Table 5.2 gives the correspondence between feedback *buffer* and the number of values returned for each vertex.

The command

`void` **PassThrough**(`float` *token*);

may be used as a marker in feedback mode. *token* is returned as if it were a primitive; it is indicated with its own unique identifying value. The ordering of any **PassThrough** commands with respect to primitive specification is maintained by feedback. **PassThrough** may not occur between **Begin** and **End**. It has no effect when the GL is not in feedback mode.

The state required for feedback is the pointer to the feedback array, the maximum number of values that may be placed there, and the feedback *type*.

Microsoft Corp. Exhibit 1009

An overflow flag is required to indicate whether the maximum allowable number of feedback values has been written; initially this flag is cleared. These state variables are GL client state. Feedback also relies on the same mode flag as selection to indicate whether the GL is in feedback, selection, or normal rendering mode.

5.4 Display Lists

A display list is simply a group of GL commands and arguments that has been stored for subsequent execution. The GL may be instructed to process a particular display list (possibly repeatedly) by providing a number that uniquely specifies it. Doing so causes the commands within the list to be executed just as if they were given normally. The only exception pertains to commands that rely upon client state. When such a command is accumulated into the display list (that is, when issued, not when executed), the client state in effect at that time applies to the command. Only server state is affected when the command is executed. As always, pointers which are passed as arguments to commands are dereferenced when the command is issued. (Vertex array pointers are dereferenced when the commands **ArrayElement**, **DrawArrays**, or **DrawElements** are accumulated into a display list.)

A display list is begun by calling

`void` **NewList**(`uint` *n*, `enum` *mode*);

n is a positive integer to which the display list that follows is assigned, and *mode* is a symbolic constant that controls the behavior of the GL during display list creation. If *mode* is `COMPILE`, then commands are not executed as they are placed in the display list. If *mode* is `COMPILE_AND_EXECUTE` then commands are executed as they are encountered, then placed in the display list. If $n = 0$, then the error `INVALID_VALUE` is generated.

After calling **NewList** all subsequent GL commands are placed in the display list (in the order the commands are issued) until a call to

`void` **EndList**(`void`);

occurs, after which the GL returns to its normal command execution state. It is only when **EndList** occurs that the specified display list is actually associated with the index indicated with **NewList**. The error `INVALID_OPERATION` is generated if **EndList** is called without a previous matching **NewList**,

Microsoft Corp. Exhibit 1009

APPENDIX T

feedback-list:
    feedback-item feedback-list
    feedback-item

feedback-item:
    point
    line-segment
    polygon
    bitmap
    pixel-rectangle
    passthrough

point:
    POINT_TOKEN vertex

line-segment:
    LINE_TOKEN vertex vertex
    LINE_RESET_TOKEN vertex vertex

polygon:
    POLYGON_TOKEN n polygon-spec

polygon-spec:
    polygon-spec vertex
    vertex vertex vertex

bitmap:
    BITMAP_TOKEN vertex

pixel-rectangle:
    DRAW_PIXEL_TOKEN vertex
    COPY_PIXEL_TOKEN vertex

passthrough:
    PASS_THROUGH_TOKEN f

vertex:
2D:
    f f
3D:
    f f f
3D_COLOR:
    f f f color
3D_COLOR_TEXTURE:
    f f f color tex
4D_COLOR_TEXTURE:
    f f f f color tex

color:
    f f f f
    f

tex:
    f f f f

Figure 5.2: Feedback syntax. f is a floating-point number. n is a floating-point integer giving the number of vertices in a polygon. The symbols ending with _TOKEN are symbolic floating-point constants. The labels under the "vertex" rule show the different data returned for vertices depending on the feedback *type*. LINE_TOKEN and LINE_RESET_TOKEN are identical except that the latter is returned only when the line stipple is reset for that line segment.

Microsoft Corp. Exhibit 1009

or if **NewList** is called a second time before calling **EndList**. The error `OUT_OF_MEMORY` is generated if **EndList** is called and the specified display list cannot be stored because insufficient memory is available. In this case GL implementations of revision 1.1 or greater insure that no change is made to the previous contents of the display list, if any, and that no other change is made to the GL state, except for the state changed by execution of GL commands when the display list mode is `COMPILE_AND_EXECUTE`.

Once defined, a display list is executed by calling

`void` **CallList**(`uint` *n*);

n gives the index of the display list to be called. This causes the commands saved in the display list to be executed, in order, just as if they were issued without using a display list. If $n = 0$, then the error `INVALID_VALUE` is generated.

The command

`void` **CallLists**(`sizei` *n*, `enum` *type*, `void` **lists*);

provides an efficient means for executing a number of display lists. *n* is an integer indicating the number of display lists to be called, and *lists* is a pointer that points to an array of offsets. Each offset is constructed as determined by *lists* as follows. First, *type* may be one of the constants `BYTE`, `UNSIGNED_BYTE`, `SHORT`, `UNSIGNED_SHORT`, `INT`, `UNSIGNED_INT`, or `FLOAT` indicating that the array pointed to by *lists* is an array of bytes, unsigned bytes, shorts, unsigned shorts, integers, unsigned integers, or floats, respectively. In this case each offset is found by simply converting each array element to an integer (floating point values are truncated). Further, *type* may be one of `2_BYTES`, `3_BYTES`, or `4_BYTES`, indicating that the array contains sequences of 2, 3, or 4 unsigned bytes, in which case each integer offset is constructed according to the following algorithm:

$offset \leftarrow 0$
for $i = 1$ **to** b
    $offset \leftarrow offset$ shifted left 8 bits
    $offset \leftarrow offset + byte$
    advance to next *byte* in the array

b is 2, 3, or 4, as indicated by *type*. If $n = 0$, **CallLists** does nothing.

Each of the n constructed offsets is taken in order and added to a display list base to obtain a display list number. For each number, the indicated display list is executed. The base is set by calling

Microsoft Corp. Exhibit 1009

```
void ListBase( uint base );
```

to specify the offset.

Indicating a display list index that does not correspond to any display list has no effect. **CallList** or **CallLists** may appear inside a display list. (If the *mode* supplied to **NewList** is `COMPILE_AND_EXECUTE`, then the appropriate lists are executed, but the **CallList** or **CallLists**, rather than those lists' constituent commands, is placed in the list under construction.) To avoid the possibility of infinite recursion resulting from display lists calling one another, an implementation dependent limit is placed on the nesting level of display lists during display list execution. This limit must be at least 64.

Two commands are provided to manage display list indices.

```
uint GenLists( sizei s );
```

returns an integer n such that the indices $n, \ldots, n+s-1$ are previously unused (i.e. there are s previously unused display list indices starting at n). **GenLists** also has the effect of creating an empty display list for each of the indices $n, \ldots, n+s-1$, so that these indices all become used. **GenLists** returns 0 if there is no group of s contiguous previously unused display list indices, or if $s = 0$.

```
boolean IsList( uint list );
```

returns `TRUE` if *list* is the index of some display list.

A contiguous group of display lists may be deleted by calling

```
void DeleteLists( uint list, sizei range );
```

where *list* is the index of the first display list to be deleted and *range* is the number of display lists to be deleted. All information about the display lists is lost, and the indices become unused. Indices to which no display list corresponds are ignored. If $range = 0$, nothing happens.

Certain commands, when called while compiling a display list, are not compiled into the display list but are executed immediately. These are: **IsList**, **GenLists**, **DeleteLists**, **FeedbackBuffer**, **SelectBuffer**, **RenderMode**, **VertexPointer**, **NormalPointer**, **ColorPointer**, **IndexPointer**, **TexCoordPointer**, **EdgeFlagPointer**, **InterleavedArrays**, **EnableClientState**, **DisableClientState**, **PushClientAttrib**, **PopClientAttrib**, **ReadPixels**, **PixelStore**, **GenTextures**, **DeleteTextures**, **AreTexturesResident**, **IsTexture**, **Flush**, **Finish**, as well as **IsEnabled** and all of the **Get** commands (see Chapter 6).

Microsoft Corp. Exhibit 1009

TexImage3D, **TexImage2D**, **TexImage1D**, **Histogram**, and **ColorTable** are executed immediately when called with the corresponding proxy arguments PROXY_TEXTURE_3D, PROXY_TEXTURE_2D, PROXY_TEXTURE_1D, PROXY_HISTOGRAM, and PROXY_COLOR_TABLE, PROXY_POST_CONVOLUTION_COLOR_TABLE, or PROXY_POST_COLOR_MATRIX_COLOR_TABLE.

Display lists require one bit of state to indicate whether a GL command should be executed immediately or placed in a display list. In the initial state, commands are executed immediately. If the bit indicates display list creation, an index is required to indicate the current display list being defined. Another bit indicates, during display list creation, whether or not commands should be executed as they are compiled into the display list. One integer is required for the current **ListBase** setting; its initial value is zero. Finally, state must be maintained to indicate which integers are currently in use as display list indices. In the initial state, no indices are in use.

5.5 Flush and Finish

The command

`void` **Flush**(`void`);

indicates that all commands that have previously been sent to the GL must complete in finite time.

The command

`void` **Finish**(`void`);

forces all previous GL commands to complete. **Finish** does not return until all effects from previously issued commands on GL client and server state and the framebuffer are fully realized.

5.6 Hints

Certain aspects of GL behavior, when there is room for variation, may be controlled with hints. A hint is specified using

`void` **Hint**(`enum` *target*, `enum` *hint*);

Microsoft Corp. Exhibit 1009

target is a symbolic constant indicating the behavior to be controlled, and *hint* is a symbolic constant indicating what type of behavior is desired. *target* may be one of `PERSPECTIVE_CORRECTION_HINT`, indicating the desired quality of parameter interpolation; `POINT_SMOOTH_HINT`, indicating the desired sampling quality of points; `LINE_SMOOTH_HINT`, indicating the desired sampling quality of lines; `POLYGON_SMOOTH_HINT`, indicating the desired sampling quality of polygons; and `FOG_HINT`, indicating whether fog calculations are done per pixel or per vertex. *hint* must be one of `FASTEST`, indicating that the most efficient option should be chosen; `NICEST`, indicating that the highest quality option should be chosen; and `DONT_CARE`, indicating no preference in the matter.

The interpretation of hints is implementation dependent. An implementation may ignore them entirely.

The initial value of all hints is `DONT_CARE`.

Microsoft Corp. Exhibit 1009

Chapter 6

State and State Requests

The state required to describe the GL machine is enumerated in section 6.2. Most state is set through the calls described in previous chapters, and can be queried using the calls described in section 6.1.

6.1 Querying GL State

6.1.1 Simple Queries

Much of the GL state is completely identified by symbolic constants. The values of these state variables can be obtained using a set of **Get** commands. There are four commands for obtaining simple state variables:

```
void GetBooleanv( enum value, boolean *data );
void GetIntegerv( enum value, int *data );
void GetFloatv( enum value, float *data );
void GetDoublev( enum value, double *data );
```

The commands obtain boolean, integer, floating-point, or double-precision state variables. *value* is a symbolic constant indicating the state variable to return. *data* is a pointer to a scalar or array of the indicated type in which to place the returned data. In addition

```
boolean IsEnabled( enum value );
```

can be used to determine if *value* is currently enabled (as with **Enable**) or disabled.

Microsoft Corp. Exhibit 1009

6.1.2 Data Conversions

If a **Get** command is issued that returns value types different from the type of the value being obtained, a type conversion is performed. If **GetBooleanv** is called, a floating-point or integer value converts to `FALSE` if and only if it is zero (otherwise it converts to `TRUE`). If **GetIntegerv** (or any of the **Get** commands below) is called, a boolean value is interpreted as either 1 or 0, and a floating-point value is rounded to the nearest integer, unless the value is an RGBA color component, a **DepthRange** value, a depth buffer clear value, or a normal coordinate. In these cases, the **Get** command converts the floating-point value to an integer according the `INT` entry of Table 4.7; a value not in $[-1, 1]$ converts to an undefined value. If **GetFloatv** is called, a boolean value is interpreted as either 1.0 or 0.0, an integer is coerced to floating-point, and a double-precision floating-point value is converted to single-precision. Analogous conversions are carried out in the case of **GetDoublev**. If a value is so large in magnitude that it cannot be represented with the requested type, then the nearest value representable using the requested type is returned.

Unless otherwise indicated, multi-valued state variables return their multiple values in the same order as they are given as arguments to the commands that set them. For instance, the two **DepthRange** parameters are returned in the order n followed by f. Similarly, points for evaluator maps are returned in the order that they appeared when passed to **Map1**. **Map2** returns $\mathbf{R}_{ij}$ in the $[(uorder)i + j]$th block of values (see page 166 for i, j, $uorder$, and $\mathbf{R}_{ij}$).

6.1.3 Enumerated Queries

Other commands exist to obtain state variables that are identified by a category (clip plane, light, material, etc.) as well as a symbolic constant. These are

```
void GetClipPlane( enum plane, double eqn[4] );
void GetLight{if}v( enum light, enum value, T data );
void GetMaterial{if}v( enum face, enum value, T data );
void GetTexEnv{if}v( enum env, enum value, T data );
void GetTexGen{if}v( enum coord, enum value, T data );
void GetTexParameter{if}v( enum target, enum value,
   T data );
void GetTexLevelParameter{if}v( enum target, int lod,
   enum value, T data );
```


Microsoft Corp. Exhibit 1009

```
void GetPixelMap{ui us f}v( enum map, T data );
void GetMap{ifd}v( enum map, enum value, T data );
```

GetClipPlane always returns four double-precision values in *eqn*; these are the coefficients of the plane equation of *plane* in eye coordinates (these coordinates are those that were computed when the plane was specified).

GetLight places information about *value* (a symbolic constant) for *light* (also a symbolic constant) in *data*. POSITION or SPOT_DIRECTION returns values in eye coordinates (again, these are the coordinates that were computed when the position or direction was specified).

GetMaterial, **GetTexGen**, **GetTexEnv**, and **GetTexParameter** are similar to **GetLight**, placing information about *value* for the target indicated by their first argument into *data*. The *face* argument to **GetMaterial** must be either FRONT or BACK, indicating the front or back material, respectively. The *env* argument to **GetTexEnv** must currently be TEXTURE_ENV. The *coord* argument to **GetTexGen** must be one of S, T, R, or Q. For **GetTexGen**, EYE_LINEAR coefficients are returned in the eye coordinates that were computed when the plane was specified; OBJECT_LINEAR coefficients are returned in object coordinates.

GetTexParameter and **GetTexLevelParameter** parameter *target* may be one of TEXTURE_1D, TEXTURE_2D, or TEXTURE_3D, indicating the currently bound one-, two-, or three-dimensional texture object. For **GetTexLevelParameter**, *target* may also be one of PROXY_TEXTURE_1D, PROXY_TEXTURE_2D, or PROXY_TEXTURE_3D, indicating the one-, two-, or three-dimensional proxy state vector. *value* is a symbolic value indicating which texture parameter is to be obtained. The *lod* argument to **GetTexLevelParameter** determines which level-of-detail's state is returned. If the *lod* argument is less than zero or if it is larger than the maximum allowable level-of-detail then the error INVALID_VALUE occurs. Queries of TEXTURE_RED_SIZE, TEXTURE_GREEN_SIZE, TEXTURE_BLUE_SIZE, TEXTURE_ALPHA_SIZE, TEXTURE_LUMINANCE_SIZE, and TEXTURE_INTENSITY_SIZE return the actual resolutions of the stored image array components, not the resolutions specified when the image array was defined. Queries of TEXTURE_WIDTH, TEXTURE_HEIGHT, TEXTURE_DEPTH, and TEXTURE_BORDER return the width, height, depth, and border as specified when the image array was created. The internal format of the image array is queried as TEXTURE_INTERNAL_FORMAT, or as TEXTURE_COMPONENTS for compatibility with GL version 1.0.

For **GetPixelMap**, the *map* must be a map name from Table 3.3. For **GetMap**, *map* must be one of the map types described in section 5.1, and

Microsoft Corp. Exhibit 1009

APPENDIX T

value must be one of ORDER, COEFF, or DOMAIN.

6.1.4 Texture Queries

The command

```
void GetTexImage( enum tex, int lod, enum format,
    enum type, void *img );
```

is used to obtain texture images. It is somewhat different from the other get commands; *tex* is a symbolic value indicating which texture is to be obtained. TEXTURE_1D indicates a one-dimensional texture, TEXTURE_2D indicates a two-dimensional texture, and TEXTURE_3D indicates a three-dimensional texture. *lod* is a level-of-detail number, *format* is a pixel format from Table 3.6, *type* is a pixel type from Table 3.5, and *img* is a pointer to a block of memory.

GetTexImage obtains component groups from a texture image with the indicated level-of-detail. The components are assigned among R, G, B, and A according to Table 6.1, starting with the first group in the first row, and continuing by obtaining groups in order from each row and proceeding from the first row to the last, and from the first image to the last for three-dimensional textures. These groups are then packed and placed in client memory. No pixel transfer operations are performed on this image, but pixel storage modes that are applicable to **ReadPixels** are applied.

For three-dimensional textures, pixel storage operations are applied as if the image were two-dimensional, except that the additional pixel storage state values PACK_IMAGE_HEIGHT and PACK_SKIP_IMAGES are applied. The correspondence of texels to memory locations is as defined for **TexImage3D** in section 3.8.1.

The row length, number of rows, image depth, and number of images are determined by the size of the texture image (including any borders). Calling **GetTexImage** with *lod* less than zero or larger than the maximum allowable causes the error INVALID_VALUE . Calling **GetTexImage** with *format* of COLOR_INDEX, STENCIL_INDEX, or DEPTH_COMPONENT causes the error INVALID_ENUM.

The command

```
boolean IsTexture( uint texture );
```

returns TRUE if *texture* is the name of a texture object. If *texture* is zero, or is a non-zero value that is not the name of a texture object, or if an error condition occurs, **IsTexture** returns FALSE. A name returned by **GenTextures**, but not yet bound, is not the name of a texture object.

Microsoft Corp. Exhibit 1009

Base Internal Format	R	G	B	A
ALPHA	0	0	0	A_i
LUMINANCE (or 1)	L_i	0	0	1
LUMINANCE_ALPHA (or 2)	L_i	0	0	A_i
INTENSITY	I_i	0	0	1
RGB (or 3)	R_i	G_i	B_i	1
RGBA (or 4)	R_i	G_i	B_i	A_i

Table 6.1: Texture, table, and filter return values. R_i, G_i, B_i, A_i, L_i, and I_i are components of the internal format that are assigned to pixel values R, G, B, and A. If a requested pixel value is not present in the internal format, the specified constant value is used.

6.1.5 Stipple Query

The command

`void` **GetPolygonStipple**(`void` **pattern*);

obtains the polygon stipple. The pattern is packed into memory according to the procedure given in section 4.3.2 for **ReadPixels**; it is as if the *height* and *width* passed to that command were both equal to 32, the *type* were `BITMAP`, and the *format* were `COLOR_INDEX`.

6.1.6 Color Matrix Query

The scale and bias variables are queried using **GetFloatv** with *pname* set to the appropriate variable name. The top matrix on the color matrix stack is returned by **GetFloatv** called with *pname* set to `COLOR_MATRIX`. The depth of the color matrix stack, and the maximum depth of the color matrix stack, are queried with **GetIntegerv**, setting *pname* to `COLOR_MATRIX_STACK_DEPTH` and `MAX_COLOR_MATRIX_STACK_DEPTH` respectively.

6.1.7 Color Table Query

The current contents of a color table are queried using

`void` **GetColorTable**(`enum` *target*, `enum` *format*, `enum` *type*,
`void` **table*);

Microsoft Corp. Exhibit 1009

target must be one of the *regular* color table names listed in table 3.4. *format* and *type* accept the same values as do the corresponding parameters of **GetTexImage**. The one-dimensional color table image is returned to client memory starting at *table*. No pixel transfer operations are performed on this image, but pixel storage modes that are applicable to **ReadPixels** are performed. Color components that are requested in the specified *format*, but which are not included in the internal format of the color lookup table, are returned as zero. The assignments of internal color components to the components requested by *format* are described in Table 6.1.

The functions

```
void GetColorTableParameter{if}v( enum target,
    enum pname, T params );
```

are used for integer and floating point query.

target must be one of the regular or proxy color table names listed in table 3.4. *pname* is one of COLOR_TABLE_SCALE, COLOR_TABLE_BIAS, COLOR_TABLE_FORMAT, COLOR_TABLE_WIDTH, COLOR_TABLE_RED_SIZE, COLOR_TABLE_GREEN_SIZE, COLOR_TABLE_BLUE_SIZE, COLOR_TABLE_ALPHA_SIZE, COLOR_TABLE_LUMINANCE_SIZE, or COLOR_TABLE_INTENSITY_SIZE. The value of the specified parameter is returned in *params*.

6.1.8 Convolution Query

The current contents of a convolution filter image are queried with the command

```
void GetConvolutionFilter( enum target, enum format,
    enum type, void *image );
```

target must be CONVOLUTION_1D or CONVOLUTION_2D. *format* and *type* accept the same values as do the corresponding parameters of **GetTexImage**. The one-dimensional or two-dimensional images is returned to client memory starting at *image*. Pixel processing and component mapping are identical to those of **GetTexImage**.

The current contents of a separable filter image are queried using

```
void GetSeparableFilter( enum target, enum format,
    enum type, void *row, void *column, void *span );
```


Microsoft Corp. Exhibit 1009

target must be SEPARABLE_2D. *format* and *type* accept the same values as do the corresponding parameters of **GetTexImage**. The row and column images are returned to client memory starting at *row* and *column* respectively. *span* is currently unused. Pixel processing and component mapping are identical to those of **GetTexImage**.

The functions

```
void GetConvolutionParameter{if}v( enum target,
    enum pname, T params );
```

are used for integer and floating point query. *target* must be CONVOLUTION_1D, CONVOLUTION_2D, or SEPARABLE_2D. *pname* is one of CONVOLUTION_BORDER_COLOR, CONVOLUTION_BORDER_MODE, CONVOLUTION_FILTER_SCALE, CONVOLUTION_FILTER_BIAS, CONVOLUTION_FORMAT, CONVOLUTION_WIDTH, CONVOLUTION_HEIGHT, MAX_CONVOLUTION_WIDTH, or MAX_CONVOLUTION_HEIGHT. The value of the specified parameter is returned in *params*.

6.1.9 Histogram Query

The current contents of the histogram table are queried using

```
void GetHistogram( enum target, boolean reset,
    enum format, enum type, void* values );
```

target must be HISTOGRAM. *type* and *format* accept the same values as do the corresponding parameters of **GetTexImage**. The one-dimensional histogram table image is returned to *values*. Pixel processing and component mapping are identical to those of **GetTexImage**.

If *reset* is TRUE, then all counters of all elements of the histogram are reset to zero. Counters are reset whether returned or not.

No counters are modified if *reset* is FALSE.

Calling

```
void ResetHistogram( enum target );
```

resets all counters of all elements of the histogram table to zero. *target* must be HISTOGRAM.

It is not an error to reset or query the contents of a histogram table with zero entries.

The functions

Microsoft Corp. Exhibit 1009

```
void GetHistogramParameter{if}v( enum target,
    enum pname, T params );
```

are used for integer and floating point query. *target* must be HISTOGRAM or PROXY_HISTOGRAM. *pname* is one of HISTOGRAM_FORMAT, HISTOGRAM_WIDTH, HISTOGRAM_RED_SIZE, HISTOGRAM_GREEN_SIZE, HISTOGRAM_BLUE_SIZE, HISTOGRAM_ALPHA_SIZE, or HISTOGRAM_LUMINANCE_SIZE. *pname* may be HISTOGRAM_SINK only for *target* HISTOGRAM. The value of the specified parameter is returned in *params*.

6.1.10 Minmax Query

The current contents of the minmax table are queried using

```
void GetMinmax( enum target, boolean reset,
    enum format, enum type, void* values );
```

target must be MINMAX. *type* and *format* accept the same values as do the corresponding parameters of **GetTexImage**. A one-dimensional image of width 2 is returned to *values*. Pixel processing and component mapping are identical to those of **GetTexImage**.

If *reset* is TRUE, then each minimum value is reset to the maximum representable value, and each maximum value is reset to the minimum representable value. All values are reset, whether returned or not.

No values are modified if *reset* is FALSE.

Calling

```
void ResetMinmax( enum target );
```

resets all minimum and maximum values of *target* to to their maximum and minimum representable values, respectively, *target* must be MINMAX.

The functions

```
void GetMinmaxParameter{if}v( enum target,
    enum pname, T params );
```

are used for integer and floating point query. *target* must be MINMAX. *pname* is MINMAX_FORMAT or MINMAX_SINK. The value of the specified parameter is returned in *params*.

Microsoft Corp. Exhibit 1009

6.1.11 Pointer and String Queries

The command

`void` **GetPointerv**(`enum` *pname*, `void` ***params*);

obtains the pointer or pointers named *pname* in the array *params*. The possible values for *pname* are `SELECTION_BUFFER_POINTER`, `FEEDBACK_BUFFER_POINTER`, `VERTEX_ARRAY_POINTER`, `NORMAL_ARRAY_POINTER`, `COLOR_ARRAY_POINTER`, `INDEX_ARRAY_POINTER`, `TEXTURE_COORD_ARRAY_POINTER`, and `EDGE_FLAG_ARRAY_POINTER`. Each returns a single pointer value.

Finally,

`ubyte` ***GetString**(`enum` *name*);

returns a pointer to a static string describing some aspect of the current GL connection. The possible values for *name* are `VENDOR`, `RENDERER`, `VERSION`, and `EXTENSIONS`. The format of the `RENDERER` and `VERSION` strings is implementation dependent. The `EXTENSIONS` string contains a space separated list of extension names (The extension names themselves do not contain any spaces); the `VERSION` string is laid out as follows:

<version number><space><vendor-specific information>

The version number is either of the form *major_number.minor_number* or *major_number.minor_number.release_number*, where the numbers all have one or more digits. The vendor specific information is optional. However, if it is present then it pertains to the server and the format and contents are implementation dependent.

GetString returns the version number (returned in the `VERSION` string) and the extension names (returned in the `EXTENSIONS` string) that can be supported on the connection. Thus, if the client and server support different versions and/or extensions, a compatible version and list of extensions is returned.

6.1.12 Saving and Restoring State

Besides providing a means to obtain the values of state variables, the GL also provides a means to save and restore groups of state variables. The **PushAttrib**, **PushClientAttrib**, **PopAttrib** and **PopClientAttrib** commands are used for this purpose. The commands

Microsoft Corp. Exhibit 1009

```
void PushAttrib( bitfield mask );
void PushClientAttrib( bitfield mask );
```

take a bitwise OR of symbolic constants indicating which groups of state variables to push onto an attribute stack. **PushAttrib** uses a server attribute stack while **PushClientAttrib** uses a client attribute stack. Each constant refers to a group of state variables. The classification of each variable into a group is indicated in the following tables of state variables. The error STACK_OVERFLOW is generated if **PushAttrib** or **PushClientAttrib** is executed while the corresponding stack depth is MAX_ATTRIB_STACK_DEPTH or MAX_CLIENT_ATTRIB_STACK_DEPTH respectively. The commands

```
void PopAttrib( void );
void PopClientAttrib( void );
```

reset the values of those state variables that were saved with the last corresponding **PushAttrib** or **PopClientAttrib**. Those not saved remain unchanged. The error STACK_UNDERFLOW is generated if **PopAttrib** or **PopClientAttrib** is executed while the respective stack is empty.

Table 6.2 shows the attribute groups with their corresponding symbolic constant names and stacks.

When **PushAttrib** is called with TEXTURE_BIT set, the priorities, border colors, filter modes, and wrap modes of the currently bound texture objects, as well as the current texture bindings and enables, are pushed onto the attribute stack. (Unbound texture objects are not pushed or restored.) When an attribute set that includes texture information is popped, the bindings and enables are first restored to their pushed values, then the bound texture objects' priorities, border colors, filter modes, and wrap modes are restored to their pushed values.

The depth of each attribute stack is implementation dependent but must be at least 16. The state required for each attribute stack is potentially 16 copies of each state variable, 16 masks indicating which groups of variables are stored in each stack entry, and an attribute stack pointer. In the initial state, both attribute stacks are empty.

In the tables that follow, a type is indicated for each variable. Table 6.3 explains these types. The type actually identifies all state associated with the indicated description; in certain cases only a portion of this state is returned. This is the case with all matrices, where only the top entry on the stack is returned; with clip planes, where only the selected clip plane is returned, with parameters describing lights, where only the value pertaining

Microsoft Corp. Exhibit 1009

Stack	Attribute	Constant
server	accum-buffer	ACCUM_BUFFER_BIT
server	color-buffer	COLOR_BUFFER_BIT
server	current	CURRENT_BIT
server	depth-buffer	DEPTH_BUFFER_BIT
server	enable	ENABLE_BIT
server	eval	EVAL_BIT
server	fog	FOG_BIT
server	hint	HINT_BIT
server	lighting	LIGHTING_BIT
server	line	LINE_BIT
server	list	LIST_BIT
server	pixel	PIXEL_MODE_BIT
server	point	POINT_BIT
server	polygon	POLYGON_BIT
server	polygon-stipple	POLYGON_STIPPLE_BIT
server	scissor	SCISSOR_BIT
server	stencil-buffer	STENCIL_BUFFER_BIT
server	texture	TEXTURE_BIT
server	transform	TRANSFORM_BIT
server	viewport	VIEWPORT_BIT
server		ALL_ATTRIB_BITS
client	vertex-array	CLIENT_VERTEX_ARRAY_BIT
client	pixel-store	CLIENT_PIXEL_STORE_BIT
client	select	can't be pushed or pop'd
client	feedback	can't be pushed or pop'd
client		ALL_CLIENT_ATTRIB_BITS

Table 6.2: Attribute groups

Microsoft Corp. Exhibit 1009

APPENDIX T

Type code	Explanation
B	Boolean
C	Color (floating-point R, G, B, and A values)
CI	Color index (floating-point index value)
T	Texture coordinates (floating-point s, t, r, q values)
N	Normal coordinates (floating-point x, y, z values)
V	Vertex, including associated data
Z	Integer
Z^+	Non-negative integer
Z_k, Z_{k*}	k-valued integer ($k*$ indicates k is minimum)
R	Floating-point number
R^+	Non-negative floating-point number
$R^{[a,b]}$	Floating-point number in the range $[a, b]$
R^k	k-tuple of floating-point numbers
P	Position (x, y, z, w floating-point coordinates)
D	Direction (x, y, z floating-point coordinates)
M^4	4×4 floating-point matrix
I	Image
A	Attribute stack entry, including mask
Y	Pointer (data type unspecified)
$n \times type$	n copies of type *type* ($n*$ indicates n is minimum)

Table 6.3: State variable types

to the selected light is returned; with textures, where only the selected texture or texture parameter is returned; and with evaluator maps, where only the selected map is returned. Finally, a "–" in the attribute column indicates that the indicated value is not included in any attribute group (and thus can not be pushed or popped with **PushAttrib**, **PushClientAttrib**, **PopAttrib**, or **PopClientAttrib**).

The M and m entries for initial minmax table values represent the maximum and minimum possible representable values, respectively.

Microsoft Corp. Exhibit 1009

6.2 State Tables

The tables on the following pages indicate which state variables are obtained with what commands. State variables that can be obtained using any of **GetBooleanv**, **GetIntegerv**, **GetFloatv**, or **GetDoublev** are listed with just one of these commands – the one that is most appropriate given the type of the data to be returned. These state variables cannot be obtained using **IsEnabled**. However, state variables for which **IsEnabled** is listed as the query command can also be obtained using **GetBooleanv**, **GetIntegerv**, **GetFloatv**, and **GetDoublev**. State variables for which any other command is listed as the query command can be obtained only by using that command.

State table entries which are required only by the imaging subset (see section 3.6.2) are typeset against a gray background.

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
-	Z_{11}	-	0	When $\neq 0$, indicates **begin/end** object	2.6.1	-
-	V	-	-	Previous vertex in **Begin/End line**	2.6.1	-
-	B	-	-	Indicates if *line-vertex* is the first	2.6.1	-
-	V	-	-	First vertex of a **Begin/End line loop**	2.6.1	-
-	Z^+	-	-	Line stipple counter	3.4	-
-	$n \times V$	-	-	Vertices inside of **Begin/End polygon**	2.6.1	-
-	Z^+	-	-	Number of *polygon-vertices*	2.6.1	-
-	$2 \times V$	-	-	Previous two vertices in a **Begin/End triangle strip**	2.6.1	-
-	Z_3	-	-	Number of vertices so far in triangle strip: 0, 1, or more	2.6.1	-
-	Z_2	-	-	Triangle strip A/B vertex pointer	2.6.1	-
-	$3 \times V$	-	-	Vertices of the quad under construction	2.6.1	-
-	Z_4	-	-	Number of vertices so far in quad strip: 0, 1, 2, or more	2.6.1	-

Table 6.4. GL Internal begin-end state variables (inaccessible)

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
CURRENT_COLOR	C	**GetIntegerv, GetFloatv**	1,1,1,1	Current color	2.7	current
CURRENT_INDEX	CI	**GetIntegerv, GetFloatv**	1	Current color index	2.7	current
CURRENT_TEXTURE_COORDS	T	**GetFloatv**	0,0,0,1	Current texture coordinates	2.7	current
CURRENT_NORMAL	N	**GetFloatv**	0,0,1	Current normal	2.7	current
-	C	–	-	Color associated with last vertex	2.6	–
-	CI	–	-	Color index associated with last vertex	2.6	–
-	T	–	-	Texture coordinates associated with last vertex	2.6	–
CURRENT_RASTER_POSITION	R^4	**GetFloatv**	0,0,0,1	Current raster position	2.12	current
CURRENT_RASTER_DISTANCE	R^+	**GetFloatv**	0	Current raster distance	2.12	current
CURRENT_RASTER_COLOR	C	**GetIntegerv, GetFloatv**	1,1,1,1	Color associated with raster position	2.12	current
CURRENT_RASTER_INDEX	CI	**GetIntegerv, GetFloatv**	1	Color index associated with raster position	2.12	current
CURRENT_RASTER_TEXTURE_COORDS	T	**GetFloatv**	0,0,0,1	Texture coordinates associated with raster position	2.12	current
CURRENT_RASTER_POSITION_VALID	B	**GetBooleanv**	*True*	Raster position valid bit	2.12	current
EDGE_FLAG	B	**GetBooleanv**	*True*	Edge flag	2.6.2	current

Table 6.5. Current Values and Associated Data

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
VERTEX_ARRAY	B	**IsEnabled**	*False*	Vertex array enable	2.8	vertex-array
VERTEX_ARRAY_SIZE	Z^+	**GetIntegerv**	4	Coordinates per vertex	2.8	vertex-array
VERTEX_ARRAY_TYPE	Z_4	**GetIntegerv**	FLOAT	Type of vertex coordinates	2.8	vertex-array
VERTEX_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between vertices	2.8	vertex-array
VERTEX_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the vertex array	2.8	vertex-array
NORMAL_ARRAY	B	**IsEnabled**	*False*	Normal array enable	2.8	vertex-array
NORMAL_ARRAY_TYPE	Z_5	**GetIntegerv**	FLOAT	Type of normal coordinates	2.8	vertex-array
NORMAL_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between normals	2.8	vertex-array
NORMAL_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the normal array	2.8	vertex-array
COLOR_ARRAY	B	**IsEnabled**	*False*	Color array enable	2.8	vertex-array
COLOR_ARRAY_SIZE	Z^+	**GetIntegerv**	4	Colors per vertex	2.8	vertex-array
COLOR_ARRAY_TYPE	Z_8	**GetIntegerv**	FLOAT	Type of color components	2.8	vertex-array
COLOR_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between colors	2.8	vertex-array
COLOR_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the color array	2.8	vertex-array
INDEX_ARRAY	B	**IsEnabled**	*False*	Index array enable	2.8	vertex-array
INDEX_ARRAY_TYPE	Z_4	**GetIntegerv**	FLOAT	Type of indices	2.8	vertex-array
INDEX_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between indices	2.8	vertex-array
INDEX_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the index array	2.8	vertex-array
TEXTURE_COORD_ARRAY	B	**IsEnabled**	*False*	Texture coordinate array enable	2.8	vertex-array
TEXTURE_COORD_ARRAY_SIZE	Z^+	**GetIntegerv**	4	Coordinates per element	2.8	vertex-array
TEXTURE_COORD_ARRAY_TYPE	Z_4	**GetIntegerv**	FLOAT	Type of texture coordinates	2.8	vertex-array
TEXTURE_COORD_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between texture coordinates	2.8	vertex-array
TEXTURE_COORD_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the texture coordinate array	2.8	vertex-array
EDGE_FLAG_ARRAY	B	**IsEnabled**	*False*	Edge flag array enable	2.8	vertex-array
EDGE_FLAG_ARRAY_STRIDE	Z^+	**GetIntegerv**	0	Stride between edge flags	2.8	vertex-array
EDGE_FLAG_ARRAY_POINTER	Y	**GetPointerv**	0	Pointer to the edge flag array	2.8	vertex-array

Table 6.6. Vertex Array Data

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
COLOR_MATRIX	$2 * \times M^4$	**GetFloatv**	Identity	Color matrix stack	3.6.3	–
MODELVIEW_MATRIX	$32 * \times M^4$	**GetFloatv**	Identity	Model-view matrix stack	2.10.2	–
PROJECTION_MATRIX	$2 * \times M^4$	**GetFloatv**	Identity	Projection matrix stack	2.10.2	–
TEXTURE_MATRIX	$2 * \times M^4$	**GetFloatv**	Identity	Texture matrix stack	2.10.2	–
VIEWPORT	$4 \times Z$	**GetIntegerv**	see 2.10.1	Viewport origin & extent	2.10.1	viewport
DEPTH_RANGE	$2 \times R^+$	**GetFloatv**	0,1	Depth range near & far	2.10.1	viewport
COLOR_MATRIX_STACK_DEPTH	Z^+	**GetIntegerv**	1	Color matrix stack pointer	3.6.3	–
MODELVIEW_STACK_DEPTH	Z^+	**GetIntegerv**	1	Model-view matrix stack pointer	2.10.2	–
PROJECTION_STACK_DEPTH	Z^+	**GetIntegerv**	1	Projection matrix stack pointer	2.10.2	–
TEXTURE_STACK_DEPTH	Z^+	**GetIntegerv**	1	Texture matrix stack pointer	2.10.2	–
MATRIX_MODE	Z_4	**GetIntegerv**	`MODELVIEW`	Current matrix mode	2.10.2	transform
NORMALIZE	B	**IsEnabled**	*False*	Current normal normalization on/off	2.10.3	transform/enable
RESCALE_NORMAL	B	**IsEnabled**	*False*	Current normal rescaling on/off	2.10.3	transform/enable
CLIP_PLANEi	$6 * \times R^4$	**GetClipPlane**	0,0,0,0	User clipping plane coefficients	2.11	transform
CLIP_PLANEi	$6 * \times B$	**IsEnabled**	*False*	ith user clipping plane enabled	2.11	transform/enable

Table 6.7. Transformation state

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
FOG_COLOR	C	**GetFloatv**	0,0,0,0	Fog color	3.10	fog
FOG_INDEX	CI	**GetFloatv**	0	Fog index	3.10	fog
FOG_DENSITY	R	**GetFloatv**	1.0	Exponential fog density	3.10	fog
FOG_START	R	**GetFloatv**	0.0	Linear fog start	3.10	fog
FOG_END	R	**GetFloatv**	1.0	Linear fog end	3.10	fog
FOG_MODE	Z_3	**GetIntegerv**	`EXP`	Fog mode	3.10	fog
FOG	B	**IsEnabled**	*False*	True if fog enabled	3.10	fog/enable
SHADE_MODEL	Z^+	**GetIntegerv**	`SMOOTH`	**ShadeModel** setting	2.13.7	lighting

Table 6.8. Coloring

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
LIGHTING	B	**IsEnabled**	*False*	True if lighting is enabled	2.13.1	lighting/enable
COLOR_MATERIAL	B	**IsEnabled**	*False*	True if color tracking is enabled	2.13.3	lighting/enable
COLOR_MATERIAL_PARAMETER	Z_5	**GetIntegerv**	AMBIENT_AND_DIFFUSE	Material properties tracking current color	2.13.3	lighting
COLOR_MATERIAL_FACE	Z_3	**GetIntegerv**	FRONT_AND_BACK	Face(s) affected by color tracking	2.13.3	lighting
AMBIENT	$2 \times C$	**GetMaterialfv**	(0.2,0.2,0.2,1.0)	Ambient material color	2.13.1	lighting
DIFFUSE	$2 \times C$	**GetMaterialfv**	(0.8,0.8,0.8,1.0)	Diffuse material color	2.13.1	lighting
SPECULAR	$2 \times C$	**GetMaterialfv**	(0.0,0.0,0.0,1.0)	Specular material color	2.13.1	lighting
EMISSION	$2 \times C$	**GetMaterialfv**	(0.0,0.0,0.0,1.0)	Emissive mat. color	2.13.1	lighting
SHININESS	$2 \times R$	**GetMaterialfv**	0.0	Specular exponent of material	2.13.1	lighting
LIGHT_MODEL_AMBIENT	C	**GetFloatv**	(0.2,0.2,0.2,1.0)	Ambient scene color	2.13.1	lighting
LIGHT_MODEL_LOCAL_VIEWER	B	**GetBooleanv**	*False*	Viewer is local	2.13.1	lighting
LIGHT_MODEL_TWO_SIDE	B	**GetBooleanv**	*False*	Use two-sided lighting	2.13.1	lighting
LIGHT_MODEL_COLOR_CONTROL	Z_2	**GetIntegerv**	SINGLE_COLOR	Color control	2.13.1	lighting

Table 6.9. Lighting (see also Table 2.7 for defaults)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
AMBIENT	$8 * \times C$	**GetLightfv**	(0.0,0.0,0.0,1.0)	Ambient intensity of light i	2.13.1	lighting
DIFFUSE	$8 * \times C$	**GetLightfv**	see 2.5	Diffuse intensity of light i	2.13.1	lighting
SPECULAR	$8 * \times C$	**GetLightfv**	see 2.5	Specular intensity of light i	2.13.1	lighting
POSITION	$8 * \times P$	**GetLightfv**	(0.0,0.0,1.0,0.0)	Position of light i	2.13.1	lighting
CONSTANT_ATTENUATION	$8 * \times R^+$	**GetLightfv**	1.0	Constant atten. factor	2.13.1	lighting
LINEAR_ATTENUATION	$8 * \times R^+$	**GetLightfv**	0.0	Linear atten. factor	2.13.1	lighting
QUADRATIC_ATTENUATION	$8 * \times R^+$	**GetLightfv**	0.0	Quadratic atten. factor	2.13.1	lighting
SPOT_DIRECTION	$8 * \times D$	**GetLightfv**	(0.0,0.0,-1.0)	Spotlight direction of light i	2.13.1	lighting
SPOT_EXPONENT	$8 * \times R^+$	**GetLightfv**	0.0	Spotlight exponent of light i	2.13.1	lighting
SPOT_CUTOFF	$8 * \times R^+$	**GetLightfv**	180.0	Spot. angle of light i	2.13.1	lighting
LIGHTi	$8 * \times B$	**IsEnabled**	*False*	True if light i enabled	2.13.1	lighting/enable
COLOR_INDEXES	$2 \times 3 \times R$	**GetMaterialfv**	0,1,1	a_m, d_m, and s_m for color index lighting	2.13.1	lighting

Table 6.10. Lighting (cont.)

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
POINT_SIZE	R^+	**GetFloatv**	1.0	Point size	3.3	point
POINT_SMOOTH	B	**IsEnabled**	*False*	Point antialiasing on	3.3	point/enable
LINE_WIDTH	R^+	**GetFloatv**	1.0	Line width	3.4	line
LINE_SMOOTH	B	**IsEnabled**	*False*	Line antialiasing on	3.4	line/enable
LINE_STIPPLE_PATTERN	Z^+	**GetIntegerv**	1's	Line stipple	3.4.2	line
LINE_STIPPLE_REPEAT	Z^+	**GetIntegerv**	1	Line stipple repeat	3.4.2	line
LINE_STIPPLE	B	**IsEnabled**	*False*	Line stipple enable	3.4.2	line/enable
CULL_FACE	B	**IsEnabled**	*False*	Polygon culling enabled	3.5.1	polygon/enable
CULL_FACE_MODE	Z_3	**GetIntegerv**	`BACK`	Cull front/back facing polygons	3.5.1	polygon
FRONT_FACE	Z_2	**GetIntegerv**	`CCW`	Polygon frontface CW/CCW indicator	3.5.1	polygon
POLYGON_SMOOTH	B	**IsEnabled**	*False*	Polygon antialiasing on	3.5	polygon/enable
POLYGON_MODE	$2 \times Z_3$	**GetIntegerv**	`FILL`	Polygon rasterization mode (front & back)	3.5.4	polygon
POLYGON_OFFSET_FACTOR	R	**GetFloatv**	0	Polygon offset factor	3.5.5	polygon
POLYGON_OFFSET_UNITS	R	**GetFloatv**	0	Polygon offset bias	3.5.5	polygon
POLYGON_OFFSET_POINT	B	**IsEnabled**	*False*	Polygon offset enable for `POINT` mode rasterization	3.5.5	polygon/enable
POLYGON_OFFSET_LINE	B	**IsEnabled**	*False*	Polygon offset enable for `LINE` mode rasterization	3.5.5	polygon/enable
POLYGON_OFFSET_FILL	B	**IsEnabled**	*False*	Polygon offset enable for `FILL` mode rasterization	3.5.5	polygon/enable
-	I	**GetPolygonStipple**	1's	Polygon stipple	3.5	polygon-stipple
POLYGON_STIPPLE	B	**IsEnabled**	*False*	Polygon stipple enable	3.5.2	polygon/enable

Table 6.11. Rasterization

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
TEXTURE_xD	$3 \times B$	**IsEnabled**	*False*	True if xD texturing is enabled; x is 1, 2, or 3	3.8.10	texture/enable
TEXTURE_BINDING_xD	$3 \times Z^+$	**GetIntegerv**	0	Texture object bound to TEXTURE_xD	3.8.8	texture
TEXTURE_xD	$n \times I$	**GetTexImage**	see 3.8	xD texture image at l.o.d. i	3.8	–
TEXTURE_WIDTH	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's specified width	3.8	–
TEXTURE_HEIGHT	$n \times Z^+$	**GetTexLevelParameter**	0	2D texture image i's specified height	3.8	–
TEXTURE_DEPTH	$n \times Z^+$	**GetTexLevelParameter**	0	3D texture image i's specified depth	3.8	–
TEXTURE_BORDER	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's specified border width	3.8	–
TEXTURE_INTERNAL_FORMAT (TEXTURE_COMPONENTS)	$n \times Z_{42}$	**GetTexLevelParameter**	1	xD texture image i's internal image format	3.8	–
TEXTURE_RED_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's red resolution	3.8	–
TEXTURE_GREEN_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's green resolution	3.8	–
TEXTURE_BLUE_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's blue resolution	3.8	–
TEXTURE_ALPHA_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's alpha resolution	3.8	–
TEXTURE_LUMINANCE_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's luminance resolution	3.8	–
TEXTURE_INTENSITY_SIZE	$n \times Z^+$	**GetTexLevelParameter**	0	xD texture image i's intensity resolution	3.8	–

Table 6.12. Texture Objects

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
TEXTURE_BORDER_COLOR	$2^+ \times C$	**GetTexParameter**	0,0,0,0	Texture border color	3.8	texture
TEXTURE_MIN_FILTER	$2^+ \times Z_6$	**GetTexParameter**	see 3.8	Texture minification function	3.8.5	texture
TEXTURE_MAG_FILTER	$2^+ \times Z_2$	**GetTexParameter**	see 3.8	Texture magnification function	3.8.6	texture
TEXTURE_WRAP_S	$3^+ \times Z_3$	**GetTexParameter**	REPEAT	Texture wrap mode S	3.8	texture
TEXTURE_WRAP_T	$2^+ \times Z_3$	**GetTexParameter**	REPEAT	Texture wrap mode T	3.8	texture
TEXTURE_WRAP_R	$1^+ \times Z_3$	**GetTexParameter**	REPEAT	Texture wrap mode R	3.8	texture
TEXTURE_PRIORITY	$2^+ \times R^{[0,1]}$	**GetTexParameterfv**	1	Texture object priority	3.8.8	texture
TEXTURE_RESIDENT	$2^+ \times B$	**GetTexParameteriv**	see 3.8.8	Texture residency	3.8.8	texture
TEXTURE_MIN_LOD	$n \times R$	**GetTexParameterfv**	-1000	Minimum level of detail	3.8	texture
TEXTURE_MAX_LOD	$n \times R$	**GetTexParameterfv**	1000	Maximum level of detail	3.8	texture
TEXTURE_BASE_LEVEL	$n \times R$	**GetTexParameterfv**	0	Base texture array	3.8	texture
TEXTURE_MAX_LEVEL	$n \times R$	**GetTexParameterfv**	1000	Maximum texture array level	3.8	texture

Table 6.13. Texture Objects (cont.)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
TEXTURE_ENV_MODE	Z_4	**GetTexEnviv**	`MODULATE`	Texture application function	3.8.9	texture
TEXTURE_ENV_COLOR	C	**GetTexEnvfv**	0,0,0,0	Texture environment color	3.8.9	texture
TEXTURE_GEN_x	$4 \times B$	**IsEnabled**	*False*	Texgen enabled (x is S, T, R, or Q)	2.10.4	texture/enable
EYE_PLANE	$4 \times R^4$	**GetTexGenfv**	see 2.10.4	Texgen plane equation coefficients (for S, T, R, and Q)	2.10.4	texture
OBJECT_PLANE	$4 \times R^4$	**GetTexGenfv**	see 2.10.4	Texgen object linear coefficients (for S, T, R, and Q)	2.10.4	texture
TEXTURE_GEN_MODE	$4 \times Z_3$	**GetTexGeniv**	`EYE_LINEAR`	Function used for texgen (for S, T, R, and Q	2.10.4	texture

Table 6.14. Texture Environment and Generation

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
SCISSOR_TEST	B	**IsEnabled**	*False*	Scissoring enabled	4.1.2	scissor/enable
SCISSOR_BOX	$4 \times Z$	**GetIntegerv**	see 4.1.2	Scissor box	4.1.2	scissor
ALPHA_TEST	B	**IsEnabled**	*False*	Alpha test enabled	4.1.3	color-buffer/enable
ALPHA_TEST_FUNC	Z_8	**GetIntegerv**	ALWAYS	Alpha test function	4.1.3	color-buffer
ALPHA_TEST_REF	R^+	**GetIntegerv**	0	Alpha test reference value	4.1.3	color-buffer
STENCIL_TEST	B	**IsEnabled**	*False*	Stenciling enabled	4.1.4	stencil-buffer/enable
STENCIL_FUNC	Z_8	**GetIntegerv**	ALWAYS	Stencil function	4.1.4	stencil-buffer
STENCIL_VALUE_MASK	Z^+	**GetIntegerv**	1's	Stencil mask	4.1.4	stencil-buffer
STENCIL_REF	Z^+	**GetIntegerv**	0	Stencil reference value	4.1.4	stencil-buffer
STENCIL_FAIL	Z_6	**GetIntegerv**	KEEP	Stencil fail action	4.1.4	stencil-buffer
STENCIL_PASS_DEPTH_FAIL	Z_6	**GetIntegerv**	KEEP	Stencil depth buffer fail action	4.1.4	stencil-buffer
STENCIL_PASS_DEPTH_PASS	Z_6	**GetIntegerv**	KEEP	Stencil depth buffer pass action	4.1.4	stencil-buffer
DEPTH_TEST	B	**IsEnabled**	*False*	Depth buffer enabled	4.1.5	depth-buffer/enable
DEPTH_FUNC	Z_8	**GetIntegerv**	LESS	Depth buffer test function	4.1.5	depth-buffer
BLEND	B	**IsEnabled**	*False*	Blending enabled	4.1.6	color-buffer/enable
BLEND_SRC	Z_{13}	**GetIntegerv**	ONE	Blending source function	4.1.6	color-buffer
BLEND_DST	Z_{12}	**GetIntegerv**	ZERO	Blending destination function	4.1.6	color-buffer
BLEND_EQUATION	Z_5	**GetIntegerv**	FUNC_ADD	Blending equation	4.1.6	color-buffer
BLEND_COLOR	C	**GetFloatv**	0,0,0,0	Constant blend color	4.1.6	color-buffer
DITHER	B	**IsEnabled**	*True*	Dithering enabled	4.1.7	color-buffer/enable
INDEX_LOGIC_OP (v1.0: GL_LOGIC_OP)	B	**IsEnabled**	*False*	Index logic op enabled	4.1.8	color-buffer/enable
COLOR_LOGIC_OP	B	**IsEnabled**	*False*	Color logic op enabled	4.1.8	color-buffer/enable
LOGIC_OP_MODE	Z_{16}	**GetIntegerv**	COPY	Logic op function	4.1.8	color-buffer

Table 6.15. Pixel Operations

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
DRAW_BUFFER	Z_{10*}	**GetIntegerv**	see 4.2.1	Buffers selected for drawing	4.2.1	color-buffer
INDEX_WRITEMASK	Z^+	**GetIntegerv**	1's	Color index writemask	4.2.2	color-buffer
COLOR_WRITEMASK	$4 \times B$	**GetBooleanv**	*True*	Color write enables; R, G, B, or A	4.2.2	color-buffer
DEPTH_WRITEMASK	B	**GetBooleanv**	*True*	Depth buffer enabled for writing	4.2.2	depth-buffer
STENCIL_WRITEMASK	Z^+	**GetIntegerv**	1's	Stencil buffer writemask	4.2.2	stencil-buffer
COLOR_CLEAR_VALUE	C	**GetFloatv**	0,0,0,0	Color buffer clear value (RGBA mode)	4.2.3	color-buffer
INDEX_CLEAR_VALUE	CI	**GetFloatv**	0	Color buffer clear value (color index mode)	4.2.3	color-buffer
DEPTH_CLEAR_VALUE	R^+	**GetIntegerv**	1	Depth buffer clear value	4.2.3	depth-buffer
STENCIL_CLEAR_VALUE	Z^+	**GetIntegerv**	0	Stencil clear value	4.2.3	stencil-buffer
ACCUM_CLEAR_VALUE	$4 \times R^+$	**GetFloatv**	0	Accumulation buffer clear value	4.2.3	accum-buffer

Table 6.16. Framebuffer Control

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
UNPACK_SWAP_BYTES	B	**GetBooleanv**	*False*	Value of UNPACK_SWAP_BYTES	4.3	pixel-store
UNPACK_LSB_FIRST	B	**GetBooleanv**	*False*	Value of UNPACK_LSB_FIRST	4.3	pixel-store
UNPACK_IMAGE_HEIGHT	Z^+	**GetIntegerv**	0	Value of UNPACK_IMAGE_HEIGHT	4.3	pixel-store
UNPACK_SKIP_IMAGES	Z^+	**GetIntegerv**	0	Value of UNPACK_SKIP_IMAGES	4.3	pixel-store
UNPACK_ROW_LENGTH	Z^+	**GetIntegerv**	0	Value of UNPACK_ROW_LENGTH	4.3	pixel-store
UNPACK_SKIP_ROWS	Z^+	**GetIntegerv**	0	Value of UNPACK_SKIP_ROWS	4.3	pixel-store
UNPACK_SKIP_PIXELS	Z^+	**GetIntegerv**	0	Value of UNPACK_SKIP_PIXELS	4.3	pixel-store
UNPACK_ALIGNMENT	Z^+	**GetIntegerv**	4	Value of UNPACK_ALIGNMENT	4.3	pixel-store
PACK_SWAP_BYTES	B	**GetBooleanv**	*False*	Value of PACK_SWAP_BYTES	4.3	pixel-store
PACK_LSB_FIRST	B	**GetBooleanv**	*False*	Value of PACK_LSB_FIRST	4.3	pixel-store
PACK_IMAGE_HEIGHT	Z^+	**GetIntegerv**	0	Value of PACK_IMAGE_HEIGHT	4.3	pixel-store
PACK_SKIP_IMAGES	Z^+	**GetIntegerv**	0	Value of PACK_SKIP_IMAGES	4.3	pixel-store
PACK_ROW_LENGTH	Z^+	**GetIntegerv**	0	Value of PACK_ROW_LENGTH	4.3	pixel-store
PACK_SKIP_ROWS	Z^+	**GetIntegerv**	0	Value of PACK_SKIP_ROWS	4.3	pixel-store
PACK_SKIP_PIXELS	Z^+	**GetIntegerv**	0	Value of PACK_SKIP_PIXELS	4.3	pixel-store
PACK_ALIGNMENT	Z^+	**GetIntegerv**	4	Value of PACK_ALIGNMENT	4.3	pixel-store

Table 6.17. Pixels

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
MAP_COLOR	B	**GetBooleanv**	*False*	True if colors are mapped	4.3	pixel
MAP_STENCIL	B	**GetBooleanv**	*False*	True if stencil values are mapped	4.3	pixel
INDEX_SHIFT	Z	**GetIntegerv**	0	Value of INDEX_SHIFT	4.3	pixel
INDEX_OFFSET	Z	**GetIntegerv**	0	Value of INDEX_OFFSET	4.3	pixel
x_SCALE	R	**GetFloatv**	1	Value of x_SCALE; x is RED, GREEN, BLUE, ALPHA, or DEPTH	4.3	pixel
x_BIAS	R	**GetFloatv**	0	Value of x_BIAS; x is one of RED, GREEN, BLUE, ALPHA, or DEPTH	4.3	pixel
COLOR_TABLE	B	**IsEnabled**	*False*	True if color table lookup is done	3.6.3	pixel/enable
POST_CONVOLUTION_COLOR_TABLE	B	**IsEnabled**	*False*	True if post convolution color table lookup is done	3.6.3	pixel/enable
POST_COLOR_MATRIX_COLOR_TABLE	B	**IsEnabled**	*False*	True if post color matrix color table lookup is done	3.6.3	pixel/enable
COLOR_TABLE	$3 \times I$	**GetColorTable**	*empty*	Color tables	3.6.3	–
COLOR_TABLE_FORMAT	$2 \times 3 \times Z_{42}$	**GetColorTable-Parameteriv**	RGBA	Color tables' internal image format	3.6.3	–
COLOR_TABLE_WIDTH	$2 \times 3 \times Z^+$	**GetColorTable-Parameteriv**	0	Color tables' specified width	3.6.3	–
COLOR_TABLE_x_SIZE	$6 \times 2 \times 3 \times Z^+$	**GetColorTable-Parameteriv**	0	Color table component resolution; x is RED, GREEN, BLUE, ALPHA, LUMINANCE, or INTENSITY	3.6.3	–
COLOR_TABLE_SCALE	$3 \times R^4$	**GetColorTable-Parameterfv**	1,1,1,1	Scale factors applied to color table entries	3.6.3	pixel
COLOR_TABLE_BIAS	$3 \times R^4$	**GetColorTable-Parameterfv**	0,0,0,0	Bias factors applied to color table entries	3.6.3	pixel

Table 6.18. Pixels (cont.)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
CONVOLUTION_1D	B	**IsEnabled**	*False*	True if 1D convolution is done	3.6.3	pixel/enable
CONVOLUTION_2D	B	**IsEnabled**	*False*	True if 2D convolution is done	3.6.3	pixel/enable
SEPARABLE_2D	B	**IsEnabled**	*False*	True if separable 2D convolution is done	3.6.3	pixel/enable
CONVOLUTION	$2 \times I$	**GetConvolution-Filter**	*empty*	Convolution filters	3.6.3	–
CONVOLUTION	$2 \times I$	**GetSeparable-Filter**	*empty*	Separable convolution filter	3.6.3	–
CONVOLUTION_BORDER_COLOR	$3 \times C$	**GetConvolution-Parameterfv**	0,0,0,0	Convolution border color	4.3	pixel
CONVOLUTION_BORDER_MODE	$3 \times Z_4$	**GetConvolution-Parameteriv**	REDUCE	Convolution border mode	4.3	pixel
CONVOLUTION_FILTER_SCALE	$3 \times R^4$	**GetConvolution-Parameterfv**	1,1,1,1	Scale factors applied to convolution filter entries	3.6.3	pixel
CONVOLUTION_FILTER_BIAS	$3 \times R^4$	**GetConvolution-Parameterfv**	0,0,0,0	Bias factors applied to convolution filter entries	3.6.3	pixel
CONVOLUTION_FORMAT	$3 \times Z_{42}$	**GetConvolution-Parameteriv**	RGBA	Convolution filter internal format	4.3	–
CONVOLUTION_WIDTH	$3 \times Z^+$	**GetConvolution-Parameteriv**	0	Convolution filter width	4.3	–
CONVOLUTION_HEIGHT	$2 \times Z^+$	**GetConvolution-Parameteriv**	0	Convolution filter height	4.3	–

Table 6.19. Pixels (cont.)

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
POST_CONVOLUTION_x_SCALE	R	**GetFloatv**	1	Component scale factors after convolution: x is RED, GREEN, BLUE, or ALPHA	3.6.3	pixel
POST_CONVOLUTION_x_BIAS	R	**GetFloatv**	0	Component bias factors after convolution: x is RED, GREEN, BLUE, or ALPHA	3.6.3	pixel
POST_COLOR_MATRIX_x_SCALE	R	**GetFloatv**	1	Component scale factors after color matrix; x is RED, GREEN, BLUE, or ALPHA	3.6.3	pixel
POST_COLOR_MATRIX_x_BIAS	R	**GetFloatv**	0	Component bias factors after color matrix; x is RED, GREEN, BLUE, or ALPHA	3.6.3	pixel
HISTOGRAM	B	**IsEnabled**	False	True if histogramming is enabled	3.6.3	pixel/enable
HISTOGRAM	I	**GetHistogram**	*empty*	Histogram table	3.6.3	–
HISTOGRAM_WIDTH	$2 \times Z^+$	**GetHistogram-Parameteriv**	0	Histogram table width	3.6.3	–
HISTOGRAM_FORMAT	$2 \times Z_{42}$	**GetHistogram-Parameteriv**	RGBA	Histogram table internal format	3.6.3	–
HISTOGRAM_x_SIZE	$5 \times 2 \times Z^+$	**GetHistogram-Parameteriv**	0	Histogram table component resolution; x is RED, GREEN, BLUE, ALPHA, or LUMINANCE	3.6.3	–
HISTOGRAM_SINK	B	**GetHistogram-Parameteriv**	False	True if histogramming consumes pixel groups	3.6.3	–

Table 6.20. Pixels (cont.)

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
MINMAX	B	**IsEnabled**	False	True if minmax is enabled	3.6.3	pixel/enable
MINMAX	R^n	**GetMinmax**	(M,M,M,M),(m,m,m,m)	Minmax table	3.6.3	–
MINMAX_FORMAT	Z_{42}	**GetMinmax-Parameteriv**	RGBA	Minmax table internal format	3.6.3	–
MINMAX_SINK	B	**GetMinmax-Parameteriv**	False	True if minmax consumes pixel groups	3.6.3	–
ZOOM_X	R	**GetFloatv**	1.0	x zoom factor	4.3	pixel
ZOOM_Y	R	**GetFloatv**	1.0	y zoom factor	4.3	pixel
x	$8 \times 32 * \times R$	**GetPixelMap**	0's	RGBA **PixelMap** translation tables; x is a map name from Table 3.3	4.3	–
x	$2 \times 32 * \times Z$	**GetPixelMap**	0's	Index **PixelMap** translation tables; x is a map name from Table 3.3	4.3	–
x_SIZE	Z^+	**GetIntegerv**	1	Size of table x	4.3	–
READ_BUFFER	Z_3	**GetIntegerv**	see 4.3.2	Read source buffer	4.3	pixel

Table 6.21. Pixels (cont.)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
ORDER	$9 \times Z_{8*}$	**GetMapiv**	1	1d map order	5.1	–
ORDER	$9 \times 2 \times Z_{8*}$	**GetMapiv**	1,1	2d map orders	5.1	–
COEFF	$9 \times 8 * \times R^n$	**GetMapfv**	see 5.1	1d control points	5.1	–
COEFF	$9 \times 8 * \times 8 * \times R^n$	**GetMapfv**	see 5.1	2d control points	5.1	–
DOMAIN	$9 \times 2 \times R$	**GetMapfv**	see 5.1	1d domain endpoints	5.1	–
DOMAIN	$9 \times 4 \times R$	**GetMapfv**	see 5.1	2d domain endpoints	5.1	–
MAP1_x	$9 \times B$	**IsEnabled**	*False*	1d map enables: x is map type	5.1	eval/enable
MAP2_x	$9 \times B$	**IsEnabled**	*False*	2d map enables: x is map type	5.1	eval/enable
MAP1_GRID_DOMAIN	$2 \times R$	**GetFloatv**	0,1	1d grid endpoints	5.1	eval
MAP2_GRID_DOMAIN	$4 \times R$	**GetFloatv**	0,1;0,1	2d grid endpoints	5.1	eval
MAP1_GRID_SEGMENTS	Z^+	**GetFloatv**	1	1d grid divisions	5.1	eval
MAP2_GRID_SEGMENTS	$2 \times Z^+$	**GetFloatv**	1,1	2d grid divisions	5.1	eval
AUTO_NORMAL	B	**IsEnabled**	*False*	True if automatic normal generation enabled	5.1	eval/enable

Table 6.22. Evaluators (**GetMap** takes a map name)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
PERSPECTIVE_CORRECTION_HINT	Z_3	**GetIntegerv**	DONT_CARE	Perspective correction hint	5.6	hint
POINT_SMOOTH_HINT	Z_3	**GetIntegerv**	DONT_CARE	Point smooth hint	5.6	hint
LINE_SMOOTH_HINT	Z_3	**GetIntegerv**	DONT_CARE	Line smooth hint	5.6	hint
POLYGON_SMOOTH_HINT	Z_3	**GetIntegerv**	DONT_CARE	Polygon smooth hint	5.6	hint
FOG_HINT	Z_3	**GetIntegerv**	DONT_CARE	Fog hint	5.6	hint

Table 6.23. Hints

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Minimum Value	Description	Sec.	Attribute
MAX_LIGHTS	Z^+	**GetIntegerv**	8	Maximum number of lights	2.13.1	–
MAX_CLIP_PLANES	Z^+	**GetIntegerv**	6	Maximum number of user clipping planes	2.11	–
MAX_COLOR_MATRIX_STACK_DEPTH	Z^+	**GetIntegerv**	2	Maximum color matrix stack depth	3.6.3	–
MAX_MODELVIEW_STACK_DEPTH	Z^+	**GetIntegerv**	32	Maximum model-view stack depth	2.10.2	–
MAX_PROJECTION_STACK_DEPTH	Z^+	**GetIntegerv**	2	Maximum projection matrix stack depth	2.10.2	–
MAX_TEXTURE_STACK_DEPTH	Z^+	**GetIntegerv**	2	Maximum number depth of texture matrix stack	2.10.2	–
SUBPIXEL_BITS	Z^+	**GetIntegerv**	4	Number of bits of subpixel precision in screen x_w and y_w	3	–
MAX_3D_TEXTURE_SIZE	Z^+	**GetIntegerv**	16	See the discussion in Section 3.8.	3.8	–
MAX_TEXTURE_SIZE	Z^+	**GetIntegerv**	64	See the discussion in Section 3.8.	3.8	–
MAX_PIXEL_MAP_TABLE	Z^+	**GetIntegerv**	32	Maximum size of a **PixelMap** translation table	3.6.3	–
MAX_NAME_STACK_DEPTH	Z^+	**GetIntegerv**	64	Maximum selection name stack depth	5.2	–
MAX_LIST_NESTING	Z^+	**GetIntegerv**	64	Maximum display list call nesting	5.4	–
MAX_EVAL_ORDER	Z^+	**GetIntegerv**	8	Maximum evaluator polynomial order	5.1	–
MAX_VIEWPORT_DIMS	$2 \times Z^+$	**GetIntegerv**	see 2.10.1	Maximum viewport dimensions	2.10.1	–
MAX_ATTRIB_STACK_DEPTH	Z^+	**GetIntegerv**	16	Maximum depth of the server attribute stack	6	–
MAX_CLIENT_ATTRIB_STACK_DEPTH	Z^+	**GetIntegerv**	16	Maximum depth of the client attribute stack	6	–
-	$3 \times Z^+$	-	32	Maximum size of a color table	3.6.3	–
-	Z^+	-	32	Maximum size of the histogram table	3.6.3	–

Table 6.24. Implementation Dependent Values

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Minimum Value	Description	Sec.	Attribute
AUX_BUFFERS	Z^+	**GetIntegerv**	0	Number of auxiliary buffers	4.2.1	–
RGBA_MODE	B	**GetBooleanv**	–	True if color buffers store rgba	2.7	–
INDEX_MODE	B	**GetBooleanv**	–	True if color buffers store indexes	2.7	–
DOUBLEBUFFER	B	**GetBooleanv**	–	True if front & back buffers exist	4.2.1	–
STEREO	B	**GetBooleanv**	–	True if left & right buffers exist	6	–
ALIASED_POINT_SIZE_RANGE	$2 \times R^+$	**GetFloatv**	1,1	Range (lo to hi) of aliased point sizes	3.3	–
SMOOTH_POINT_SIZE_RANGE (v1.1: POINT_SIZE_RANGE)	$2 \times R^+$	**GetFloatv**	1,1	Range (lo to hi) of antialiased point sizes	3.3	–
SMOOTH_POINT_SIZE_GRANULARITY (v1.1: POINT_SIZE_GRANULARITY)	R^+	**GetFloatv**	–	Antialiased point size granularity	3.3	–
ALIASED_LINE_WIDTH_RANGE	$2 \times R^+$	**GetFloatv**	1,1	Range (lo to hi) of aliased line widths	3.4	–
SMOOTH_LINE_WIDTH_RANGE (v1.1: LINE_WIDTH_RANGE)	$2 \times R^+$	**GetFloatv**	1,1	Range (lo to hi) of antialiased line widths	3.4	–
SMOOTH_LINE_WIDTH_GRANULARITY (v1.1: LINE_WIDTH_GRANULARITY)	R^+	**GetFloatv**	–	Antialiased line width granularity	3.4	–
MAX_CONVOLUTION_WIDTH	$3 \times Z^+$	**GetConvolution-Parameteriv**	3	Maximum width of convolution filter	4.3	–
MAX_CONVOLUTION_HEIGHT	$2 \times Z^+$	**GetConvolution-Parameteriv**	3	Maximum height of convolution filter	4.3	–
MAX_ELEMENTS_INDICES	Z^+	**GetIntegerv**	–	Recommended maximum number of **DrawRangeElements** indices	2.8	–
MAX_ELEMENTS_VERTICES	Z^+	**GetIntegerv**	–	Recommended maximum number of **DrawRangeElements** vertices	2.8	–

Table 6.25. More Implementation Dependent Values

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
x_BITS	Z^+	**GetIntegerv**	-	Number of bits in x color buffer component; x is one of RED, GREEN, BLUE, ALPHA, or INDEX	4	–
DEPTH_BITS	Z^+	**GetIntegerv**	-	Number of depth buffer planes	4	–
STENCIL_BITS	Z^+	**GetIntegerv**	-	Number of stencil planes	4	–
ACCUM_x_BITS	Z^+	**GetIntegerv**	-	Number of bits in x accumulation buffer component (x is RED, GREEN, BLUE, or ALPHA	4	–

Table 6.26. Implementation Dependent Pixel Depths

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
LIST_BASE	Z^+	**GetIntegerv**	0	Setting of **ListBase**	5.4	list
LIST_INDEX	Z^+	**GetIntegerv**	0	number of display list under construction; 0 if none	5.4	–
LIST_MODE	Z^+	**GetIntegerv**	0	Mode of display list under construction; undefined if none	5.4	–
-	$16 * \times A$	–	empty	Server attribute stack	6	–
ATTRIB_STACK_DEPTH	Z^+	**GetIntegerv**	0	Server attribute stack pointer	6	–
-	$16 * \times A$	–	empty	Client attribute stack	6	–
CLIENT_ATTRIB_STACK_DEPTH	Z^+	**GetIntegerv**	0	Client attribute stack pointer	6	–
NAME_STACK_DEPTH	Z^+	**GetIntegerv**	0	Name stack depth	5.2	–
RENDER_MODE	Z_3	**GetIntegerv**	RENDER	**RenderMode** setting	5.2	–
SELECTION_BUFFER_POINTER	Y	**GetPointerv**	0	Selection buffer pointer	5.2	select
SELECTION_BUFFER_SIZE	Z^+	**GetIntegerv**	0	Selection buffer size	5.2	select
FEEDBACK_BUFFER_POINTER	Y	**GetPointerv**	0	Feedback buffer pointer	5.3	feedback
FEEDBACK_BUFFER_SIZE	Z^+	**GetIntegerv**	0	Feedback buffer size	5.3	feedback
FEEDBACK_BUFFER_TYPE	Z_5	**GetIntegerv**	2D	Feedback type	5.3	feedback
-	$n \times Z_8$	**GetError**	0	Current error code(s)	2.5	–
-	$n \times B$	–	*False*	True if there is a corresponding error	2.5	–

Table 6.27. Miscellaneous

Microsoft Corp. Exhibit 1009

Appendix A

Invariance

The OpenGL specification is not pixel exact. It therefore does not guarantee an exact match between images produced by different GL implementations. However, the specification does specify exact matches, in some cases, for images produced by the same implementation. The purpose of this appendix is to identify and provide justification for those cases that require exact matches.

A.1 Repeatability

The obvious and most fundamental case is repeated issuance of a series of GL commands. For any given GL and framebuffer state *vector*, and for any GL command, the resulting GL and framebuffer state must be identical whenever the command is executed on that initial GL and framebuffer state.

One purpose of repeatability is avoidance of visual artifacts when a double-buffered scene is redrawn. If rendering is not repeatable, swapping between two buffers rendered with the same command sequence may result in visible changes in the image. Such false motion is distracting to the viewer. Another reason for repeatability is testability.

Repeatability, while important, is a weak requirement. Given only repeatability as a requirement, two scenes rendered with one (small) polygon changed in position might differ at every pixel. Such a difference, while within the law of repeatability, is certainly not within its spirit. Additional invariance rules are desirable to ensure useful operation.

Microsoft Corp. Exhibit 1009

A.2 Multi-pass Algorithms

Invariance is necessary for a whole set of useful multi-pass algorithms. Such algorithms render multiple times, each time with a different GL mode vector, to eventually produce a result in the framebuffer. Examples of these algorithms include:

- "Erasing" a primitive from the framebuffer by redrawing it, either in a different color or using the XOR logical operation.
- Using stencil operations to compute capping planes.

On the other hand, invariance rules can greatly increase the complexity of high-performance implementations of the GL. Even the weak repeatability requirement significantly constrains a parallel implementation of the GL. Because GL implementations are required to implement ALL GL capabilities, not just a convenient subset, those that utilize hardware acceleration are expected to alternate between hardware and software modules based on the current GL mode vector. A strong invariance requirement forces the behavior of the hardware and software modules to be identical, something that may be very difficult to achieve (for example, if the hardware does floating-point operations with different precision than the software).

What is desired is a compromise that results in many compliant, high-performance implementations, and in many software vendors choosing to port to OpenGL.

A.3 Invariance Rules

For a given instantiation of an OpenGL rendering context:

Rule 1 *For any given GL and framebuffer state vector, and for any given GL command, the resulting GL and framebuffer state must be identical each time the command is executed on that initial GL and framebuffer state.*

Rule 2 *Changes to the following state values have no side effects (the use of any other state value is not affected by the change):*

Required:

- *Framebuffer contents (all bitplanes)*
- *The color buffers enabled for writing*

Microsoft Corp. Exhibit 1009

- *The values of matrices other than the top-of-stack matrices*
- *Scissor parameters (other than enable)*
- *Writemasks (color, index, depth, stencil)*
- *Clear values (color, index, depth, stencil, accumulation)*
- *Current values (color, index, normal, texture coords, edgeflag)*
- *Current raster color, index and texture coordinates.*
- *Material properties (ambient, diffuse, specular, emission, shininess)*

Strongly suggested:

- *Matrix mode*
- *Matrix stack depths*
- *Alpha test parameters (other than enable)*
- *Stencil parameters (other than enable)*
- *Depth test parameters (other than enable)*
- *Blend parameters (other than enable)*
- *Logical operation parameters (other than enable)*
- *Pixel storage and transfer state*
- *Evaluator state (except as it affects the vertex data generated by the evaluators)*
- *Polygon offset parameters (other than enables, and except as they affect the depth values of fragments)*

Corollary 1 *Fragment generation is invariant with respect to the state values marked with • in Rule 2.*

Corollary 2 *The window coordinates (x, y, and z) of generated fragments are also invariant with respect to*

Required:

- *Current values (color, color index, normal, texture coords, edgeflag)*
- *Current raster color, color index, and texture coordinates*
- *Material properties (ambient, diffuse, specular, emission, shininess)*

Microsoft Corp. Exhibit 1009

Rule 3 *The arithmetic of each per-fragment operation is invariant except with respect to parameters that directly control it (the parameters that control the alpha test, for instance, are the alpha test enable, the alpha test function, and the alpha test reference value).*

Corollary 3 *Images rendered into different color buffers sharing the same framebuffer, either simultaneously or separately using the same command sequence, are pixel identical.*

A.4 What All This Means

Hardware accelerated GL implementations are expected to default to software operation when some GL state vectors are encountered. Even the weak repeatability requirement means, for example, that OpenGL implementations cannot apply hysteresis to this swap, but must instead guarantee that a given mode vector implies that a subsequent command *always* is executed in either the hardware or the software machine.

The stronger invariance rules constrain when the switch from hardware to software rendering can occur, given that the software and hardware renderers are not pixel identical. For example, the switch can be made when blending is enabled or disabled, but it should not be made when a change is made to the blending parameters.

Because floating point values may be represented using different formats in different renderers (hardware and software), many OpenGL state values may change subtly when renderers are swapped. This is the type of state value change that Rule 1 seeks to avoid.

Microsoft Corp. Exhibit 1009

Appendix B

Corollaries

The following observations are derived from the body and the other appendixes of the specification. Absence of an observation from this list in no way impugns its veracity.

1. The `CURRENT_RASTER_TEXTURE_COORDS` must be maintained correctly at all times, including periods while texture mapping is not enabled, and when the GL is in color index mode.

2. When requested, texture coordinates returned in feedback mode are always valid, including periods while texture mapping is not enabled, and when the GL is in color index mode.

3. The error semantics of upward compatible OpenGL revisions may change. Otherwise, only additions can be made to upward compatible revisions.

4. GL query commands are not required to satisfy the semantics of the **Flush** or the **Finish** commands. All that is required is that the queried state be consistent with complete execution of all previously executed GL commands.

5. Application specified point size and line width must be returned as specified when queried. Implementation dependent clamping affects the values only while they are in use.

6. Bitmaps and pixel transfers do not cause selection hits.

7. The mask specified as the third argument to **StencilFunc** affects the operands of the stencil comparison function, but has no direct effect on

Microsoft Corp. Exhibit 1009

the update of the stencil buffer. The mask specified by **StencilMask** has no effect on the stencil comparison function; it limits the effect of the update of the stencil buffer.

8. Polygon shading is completed before the polygon mode is interpreted. If the shade model is `FLAT`, all of the points or lines generated by a single polygon will have the same color.

9. A display list is just a group of commands and arguments, so errors generated by commands in a display list must be generated when the list is executed. If the list is created in `COMPILE` mode, errors should not be generated while the list is being created.

10. **RasterPos** does not change the current raster index from its default value in an RGBA mode GL context. Likewise, **RasterPos** does not change the current raster color from its default value in a color index GL context. Both the current raster index and the current raster color can be queried, however, regardless of the color mode of the GL context.

11. A material property that is attached to the current color via **ColorMaterial** always takes the value of the current color. Attempts to change that material property via **Material** calls have no effect.

12. **Material** and **ColorMaterial** can be used to modify the RGBA material properties, even in a color index context. Likewise, **Material** can be used to modify the color index material properties, even in an RGBA context.

13. There is no atomicity requirement for OpenGL rendering commands, even at the fragment level.

14. Because rasterization of non-antialiased polygons is point sampled, polygons that have no area generate no fragments when they are rasterized in `FILL` mode, and the fragments generated by the rasterization of "narrow" polygons may not form a continuous array.

15. OpenGL does not force left- or right-handedness on any of its coordinates systems. Consider, however, the following conditions: (1) the object coordinate system is right-handed; (2) the only commands used to manipulate the model-view matrix are **Scale** (with positive scaling values only), **Rotate**, and **Translate**; (3) exactly one of either **Frustum** or **Ortho** is used to set the projection matrix; (4) the near value

Microsoft Corp. Exhibit 1009

is less than the far value for **DepthRange**. If these conditions are all satisfied, then the eye coordinate system is right-handed and the clip, normalized device, and window coordinate systems are left-handed.

16. ColorMaterial has no effect on color index lighting.

17. (No pixel dropouts or duplicates.) Let two polygons share an identical edge (that is, there exist vertices A and B of an edge of one polygon, and vertices C and D of an edge of the other polygon, and the coordinates of vertex A (resp. B) are identical to those of vertex C (resp. D), and the state of the the coordinate transfomations is identical when A, B, C, and D are specified). Then, when the fragments produced by rasterization of both polygons are taken together, each fragment intersecting the interior of the shared edge is produced exactly once.

18. OpenGL state continues to be modified in `FEEDBACK` mode and in `SELECT` mode. The contents of the framebuffer are not modified.

19. The current raster position, the user defined clip planes, the spot directions and the light positions for `LIGHT`i, and the eye planes for texgen are transformed when they are specified. They are not transformed during a **PopAttrib**, or when copying a context.

20. Dithering algorithms may be different for different components. In particular, alpha may be dithered differently from red, green, or blue, and an implementation may choose to not dither alpha at all.

21. For any GL and framebuffer state, and for any group of GL commands and arguments, the resulting GL and framebuffer state is identical whether the GL commands and arguments are executed normally or from a display list.

Microsoft Corp. Exhibit 1009

Appendix C

Version 1.1

OpenGL version 1.1 is the first revision since the original version 1.0 was released on 1 July 1992. Version 1.1 is upward compatible with version 1.0, meaning that any program that runs with a 1.0 GL implementation will also run unchanged with a 1.1 GL implementation. Several additions were made to the GL, especially to the texture mapping capabilities, but also to the geometry and fragment operations. Following are brief descriptions of each addition.

C.1 Vertex Array

Arrays of vertex data may be transferred to the GL with many fewer commands than were previously necessary. Six arrays are defined, one each storing vertex positions, normal coordinates, colors, color indices, texture coordinates, and edge flags. The arrays may be specified and enabled independently, or one of the pre-defined configurations may be selected with a single command.

The primary goal was to decrease the number of subroutine calls required to transfer non-display listed geometry data to the GL. A secondary goal was to improve the efficiency of the transfer; especially to allow direct memory access (DMA) hardware to be used to effect the transfer. The additions match those of the `EXT_vertex_array` extension, except that static array data are not supported (because they complicated the interface, and were not being used), and the pre-defined configurations are added (both to reduce subroutine count even further, and to allow for efficient transfer of array data).

Microsoft Corp. Exhibit 1009

C.2 Polygon Offset

Depth values of fragments generated by the rasterization of a polygon may be shifted toward or away from the origin, as an affine function of the window coordinate depth slope of the polygon. Shifted depth values allow coplanar geometry, especially facet outlines, to be rendered without depth buffer artifacts. They may also be used by future shadow generation algorithms.

The additions match those of the `EXT_polygon_offset` extension, with two exceptions. First, the offset is enabled separately for `POINT`, `LINE`, and `FILL` rasterization modes, all sharing a single affine function definition. (Shifting the depth values of the outline fragments, instead of the fill fragments, allows the contents of the depth buffer to be maintained correctly.) Second, the offset bias is specified in units of depth buffer resolution, rather than in the [0,1] depth range.

C.3 Logical Operation

Fragments generated by RGBA rendering may be merged into the framebuffer using a logical operation, just as color index fragments are in GL version 1.0. Blending is disabled during such operation because it is rarely desired, because many systems could not support it, and to match the semantics of the `EXT_blend_logic_op` extension, on which this addition is loosely based.

C.4 Texture Image Formats

Stored texture arrays have a format, known as the *internal format*, rather than a simple count of components. The internal format is represented as a single enumerated value, indicating both the organization of the image data (`LUMINANCE`, `RGB`, etc.) and the number of bits of storage for each image component. Clients can use the internal format specification to suggest the desired storage precision of texture images. New *base formats*, `ALPHA` and `INTENSITY`, provide new texture environment operations. These additions match those of a subset of the `EXT_texture` extension.

C.5 Texture Replace Environment

A common use of texture mapping is to replace the color values of generated fragments with texture color data. This could be specified only indirectly

Microsoft Corp. Exhibit 1009

in GL version 1.0, which required that client specified "white" geometry be modulated by a texture. GL version 1.1 allows such replacement to be specified explicitly, possibly improving performance. These additions match those of a subset of the `EXT_texture` extension.

C.6 Texture Proxies

Texture proxies allow a GL implementation to advertise different maximum texture image sizes as a function of some other texture parameters, especially of the internal image format. Clients may use the proxy query mechanism to tailor their use of texture resources at run time. The proxy interface is designed to allow such queries without adding new routines to the GL interface. These additions match those of a subset of the `EXT_texture` extension, except that implementations return allocation information consistent with support for complete mipmap arrays.

C.7 Copy Texture and Subtexture

Texture array data can be specified from framebuffer memory, as well as from client memory, and rectangular subregions of texture arrays can be redefined either from client or framebuffer memory. These additions match those defined by the `EXT_copy_texture` and `EXT_subtexture` extensions.

C.8 Texture Objects

A set of texture arrays and their related texture state can be treated as a single object. Such treatment allows for greater implementation efficiency when multiple arrays are used. In conjunction with the subtexture capability, it also allows clients to make gradual changes to existing texture arrays, rather than completely redefining them. These additions match those of the `EXT_texture_object` extension, with slight additions to the texture residency semantics.

C.9 Other Changes

1. Color indices may now be specified as unsigned bytes.

Microsoft Corp. Exhibit 1009

APPENDIX T

2. Texture coordinates s, t, and r are divided by q during the rasterization of points, pixel rectangles, and bitmaps. This division was documented only for lines and polygons in the 1.0 version.

3. The line rasterization algorithm was changed so that vertical lines on pixel borders rasterize correctly.

4. Separate pixel transfer discussions in chapter 3 and chapter 4 were combined into a single discussion in chapter 3.

5. Texture alpha values are returned as 1.0 if there is no alpha channel in the texture array. This behavior was unspecified in the 1.0 version, and was incorrectly documented in the reference manual.

6. Fog start and end values may now be negative.

7. Evaluated color values direct the evaluation of the lighting equation if **ColorMaterial** is enabled.

C.10 Acknowledgements

OpenGL 1.1 is the result of the contributions of many people, representing a cross section of the computer industry. Following is a partial list of the contributors, including the company that they represented at the time of their contribution:

Kurt Akeley, Silicon Graphics
Bill Armstrong, Evans & Sutherland
Andy Bigos, 3Dlabs
Pat Brown, IBM
Jim Cobb, Evans & Sutherland
Dick Coulter, Digital Equipment
Bruce D'Amora, GE Medical Systems
John Dennis, Digital Equipment
Fred Fisher, Accel Graphics
Chris Frazier, Silicon Graphics
Todd Frazier, Evans & Sutherland
Tim Freese, NCD
Ken Garnett, NCD
Mike Heck, Template Graphics Software
Dave Higgins, IBM
Phil Huxley, 3Dlabs

Microsoft Corp. Exhibit 1009

Dale Kirkland, Intergraph
Hock San Lee, Microsoft
Kevin LeFebvre, Hewlett Packard
Jim Miller, IBM
Tim Misner, SunSoft
Jeremy Morris, 3Dlabs
Israel Pinkas, Intel
Bimal Poddar, IBM
Lyle Ramshaw, Digital Equipment
Randi Rost, Hewlett Packard
John Schimpf, Silicon Graphics
Mark Segal, Silicon Graphics
Igor Sinyak, Intel
Jeff Stevenson, Hewlett Packard
Bill Sweeney, SunSoft
Kelvin Thompson, Portable Graphics
Neil Trevett, 3Dlabs
Linas Vepstas, IBM
Andy Vesper, Digital Equipment
Henri Warren, Megatek
Paula Womack, Silicon Graphics
Mason Woo, Silicon Graphics
Steve Wright, Microsoft

Microsoft Corp. Exhibit 1009

Appendix D

Version 1.2

OpenGL version 1.2, released on March 16, 1998, is the second revision since the original version 1.0. Version 1.2 is upward compatible with version 1.1, meaning that any program that runs with a 1.1 GL implementation will also run unchanged with a 1.2 GL implementation.

Several additions were made to the GL, especially to texture mapping capabilities and the pixel processing pipeline. Following are brief descriptions of each addition.

D.1 Three-Dimensional Texturing

Three-dimensional textures can be defined and used. In-memory formats for three-dimensional images, and pixel storage modes to support them, are also defined. The additions match those of the `EXT_texture3D` extension.

One important application of three-dimensional textures is rendering volumes of image data.

D.2 BGRA Pixel Formats

`BGRA` extends the list of host-memory color formats. Specifically, it provides a component order matching file and framebuffer formats common on Windows platforms. The additions match those of the `EXT_bgra` extension.

D.3 Packed Pixel Formats

Packed pixels in host memory are represented entirely by one unsigned byte, one unsigned short, or one unsigned integer. The fields with the packed pixel

Microsoft Corp. Exhibit 1009

are not proper machine types, but the pixel as a whole is. Thus the pixel storage modes and their unpacking counterparts all work correctly with packed pixels.

The additions match those of the `EXT_packed_pixels` extension, with the further addition of reversed component order packed formats.

D.4 Normal Rescaling

Normals may be rescaled by a constant factor derived from the modelview matrix. Rescaling can operate faster than renormalization in many cases, while resulting in the same unit normals.

The additions are based on the `EXT_rescale_normal` extension.

D.5 Separate Specular Color

Lighting calculations are modified to produce a primary color consisting of emissive, ambient and diffuse terms of the usual GL lighting equation, and a secondary color consisting of the specular term. Only the primary color is modified by the texture environment; the secondary color is added to the result of texturing to produce a single post-texturing color. This allows highlights whose color is based on the light source creating them, rather than surface properties.

The additions match those of the `EXT_separate_specular_color` extension.

D.6 Texture Coordinate Edge Clamping

GL normally clamps such that the texture coordinates are limited to exactly the range $[0, 1]$. When a texture coordinate is clamped using this algorithm, the texture sampling filter straddles the edge of the texture image, taking half its sample values from within the texture image, and the other half from the texture border. It is sometimes desirable to clamp a texture without requiring a border, and without using the constant border color.

A new texture clamping algorithm, `CLAMP_TO_EDGE`, clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel. The color returned when clamping is derived only from texels at the edge of the texture image.

The additions match those of the `SGIS_texture_edge_clamp` extension.

Microsoft Corp. Exhibit 1009

D.7 Texture Level of Detail Control

Two constraints related to the texture level of detail parameter λ are added. One constraint clamps λ to a specified floating point range. The other limits the selection of mipmap image arrays to a subset of the arrays that would otherwise be considered.

Together these constraints allow a large texture to be loaded and used initially at low resolution, and to have its resolution raised gradually as more resolution is desired or available. Image array specification is necessarily integral, rather than continuous. By providing separate, continuous clamping of the λ parameter, it is possible to avoid "popping" artifacts when higher resolution images are provided.

The additions match those of the `SGIS_texture_lod` extension.

D.8 Vertex Array Draw Element Range

A new form of **DrawElements** that provides explicit information on the range of vertices referred to by the index set is added. Implementations can take advantage of this additional information to process vertex data without having to scan the index data to determine which vertices are referenced.

The additions match those of the `EXT_draw_range_elements` extension.

D.9 Imaging Subset

The remaining new features are primarily intended for advanced image processing applications, and may not be present in all GL implementations. The are collectively referred to as the *imaging subset*.

D.9.1 Color Tables

A new RGBA-format color lookup mechanism is defined in the pixel transfer process, providing additional lookup capabilities beyond the existing lookup. The key difference is that the new lookup tables are treated as one-dimensional images with internal formats, like texture images and convolution filter images. Thus the new tables can operate on a subset of the components of passing pixel groups. For example, a table with internal format `ALPHA` modifies only the A component of each pixel group, leaving the R, G, and B components unmodified.

Microsoft Corp. Exhibit 1009

Three independent lookups may be performed: prior to convolution; after convolution and prior to color matrix transformation; after color matrix transformation and prior to gathering pipeline statistics.

Methods to initialize the color lookup tables from the framebuffer, in addition to the standard memory source mechanisms, are provided.

Portions of a color lookup table may be redefined without reinitializing the entire table. The affected portions may be specified either from host memory or from the framebuffer.

The additions match those of the `EXT_color_table` and `EXT_color_subtable` extensions.

D.9.2 Convolution

One- or two-dimensional convolution operations are executed following the first color table lookup in the pixel transfer process. The convolution kernels are themselves treated as one- and two-dimensional images, which can be loaded from application memory or from the framebuffer.

The convolution framework is designed to accommodate three-dimensional convolution, but that API is left for a future extension.

The additions match those of the `EXT_convolution` and `HP_convolution_border_modes` extensions.

D.9.3 Color Matrix

A 4x4 matrix transformation and associated matrix stack are added to the pixel transfer path. The matrix operates on RGBA pixel groups, using the equation

$$C' = MC,$$

where

$$C = \begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix}$$

and M is the 4×4 matrix on the top of the color matrix stack. After the matrix multiplication, each resulting color component is scaled and biased by a programmed amount. Color matrix multiplication follows convolution.

The color matrix can be used to reassign and duplicate color components. It can also be used to implement simple color space conversions.

The additions match those of the `SGI_color_matrix` extension.

Microsoft Corp. Exhibit 1009

D.9.4 Pixel Pipeline Statistics

Pixel operations that count occurences of specific color component values (histogram) and that track the minimum and maximum color component values (minmax) are performed at the end of the pixel transfer pipeline. An optional mode allows pixel data to be discarded after the histogram and/or minmax operations are completed. Otherwise the pixel data continues on to the next operation unaffected.

The additions match those of the `EXT_histogram` extension.

D.9.5 Constant Blend Color

A constant color that can be used to define blend weighting factors may be defined. A typical usage is blending two RGB images. Without the constant blend factor, one image must have an alpha channel with each pixel set to the desired blend factor.

The additions match those of the `EXT_blend_color` extension.

D.9.6 New Blending Equations

Blending equations other than the normal weighted sum of source and destination components may be used.

Two of the new equations produce the minimum (or maximum) color components of the source and destination colors. Taking the maximum is useful for applications such as maximum projection in medical imaging.

The other two equations are similar to the default blending equation, but produce the difference of its left and right hand sides, rather than the sum. Image differences are useful in many image processing applications.

The additions match those of the `EXT_blend_minmax` and `EXT_blend_subtract` extensions.

D.10 Acknowledgements

OpenGL 1.2 is the result of the contributions of many people, representing a cross section of the computer industry. Following is a partial list of the contributors, including the company that they represented at the time of their contribution:

Kurt Akeley, Silicon Graphics

Bill Armstrong, Evans & Sutherland

Otto Berkes, Microsoft

Microsoft Corp. Exhibit 1009

Pierre-Luc Bisaillon, Matrox Graphics
Drew Bliss, Microsoft
David Blythe, Silicon Graphics
Jon Brewster, Hewlett Packard
Dan Brokenshire, IBM
Pat Brown, IBM
Newton Cheung, S3
Bill Clifford, Digital
Jim Cobb, Parametric Technology
Bruce D'Amora, IBM
Kevin Dallas, Microsoft
Mahesh Dandapani, Rendition
Daniel Daum, AccelGraphics
Suzy Deffeyes, IBM
Peter Doyle, Intel
Jay Duluk, Raycer
Craig Dunwoody, Silicon Graphics
Dave Erb, IBM
Fred Fisher, AccelGraphics / Dynamic Pictures
Celeste Fowler, Silicon Graphics
Allen Gallotta, ATI
Ken Garnett, NCD
Michael Gold, Nvidia / Silicon Graphics
Craig Groeschel, Metro Link
Jan Hardenbergh, Mitsubishi Electric
Mike Heck, Template Graphics Software
Dick Hessel, Raycer Graphics
Paul Ho, Silicon Graphics
Shawn Hopwood, Silicon Graphics
Jim Hurley, Intel
Phil Huxley, 3Dlabs
Dick Jay, Template Graphics Software
Paul Jensen, 3Dfx
Brett Johnson, Hewlett Packard
Michael Jones, Silicon Graphics
Tim Kelley, Real3D
Jon Khazam, Intel
Louis Khouw, Sun
Dale Kirkland, Intergraph
Chris Kitrick, Raycer

Microsoft Corp. Exhibit 1009

APPENDIX T

Don Kuo, S3
Herb Kuta, Quantum 3D
Phil Lacroute, Silicon Graphics
Prakash Ladia, S3
Jon Leech, Silicon Graphics
Kevin Lefebvre, Hewlett Packard
David Ligon, Raycer Graphics
Kent Lin, S3
Dan McCabe, S3
Jack Middleton, Sun
Tim Misner, Intel
Bill Mitchell, National Institute of Standards
Jeremy Morris, 3Dlabs
Gene Munce, Intel
William Newhall, Real3D
Matthew Papakipos, Nvidia / Raycer
Garry Paxinos, Metro Link
Hanspeter Pfister, Mitsubishi Electric
Richard Pimentel, Parametric Technology
Bimal Poddar, IBM / Intel
Rob Putney, IBM
Mike Quinlan, Real3D
Nate Robins, University of Utah
Detlef Roettger, Elsa
Randi Rost, Hewlett Packard
Kevin Rushforth, Sun
Richard S. Wright, Real3D
Hock San Lee, Microsoft
John Schimpf, Silicon Graphics
Stefan Seeboth, ELSA
Mark Segal, Silicon Graphics
Bob Seitsinger, S3
Min-Zhi Shao, S3
Colin Sharp, Rendition
Igor Sinyak, Intel
Bill Sweeney, Sun
William Sweeney, Sun
Nathan Tuck, Raycer
Doug Twillenger, Sun
John Tynefeld, 3dfx

Microsoft Corp. Exhibit 1009

Kartik Venkataraman, Intel
Andy Vesper, Digital Equipment
Henri Warren, Digital Equipment / Megatek
Paula Womack, Silicon Graphics
Steve Wright, Microsoft
David Yu, Silicon Graphics
Randy Zhao, S3

Microsoft Corp. Exhibit 1009

Appendix E

Version 1.2.1

OpenGL version 1.2.1, released on October 14, 1998, introduced ARB extensions (see Appendix F). The only ARB extension defined in this version is multitexture, allowing application of multiple textures to a fragment in one rendering pass. Multitexture is based on the `SGIS_multitexture` extension, simplified by removing the ability to route texture coordinate sets to arbitrary texture units.

A new corollary discussing display list and immediate mode invariance was added to Appendix B on April 1, 1999.

Microsoft Corp. Exhibit 1009

Appendix F

ARB Extensions

OpenGL extensions that have been approved by the OpenGL Architectural Review Board (ARB) are described in this chapter. These extensions are not required to be supported by a conformant OpenGL implementation, but are expected to be widely available; they define functionality that is likely to move into the required feature set in a future revision of the specification.

In order not to compromise the readability of the core specification, ARB extensions are not integrated into the core language; instead, they are presented in this chapter, as changes to the core.

F.1 Naming Conventions

To distinguish ARB extensions from core OpenGL features and from vendor-specific extensions, the following naming conventions are used:

- A unique *name string* of the form `"GL_ARB_`*name*`"` is associated with each extension. If the extension is supported by an implementation, this string will be present in the `EXTENSIONS` string described in section 6.1.11.

- All functions defined by the extension will have names of the form ***Function*ARB**

- All enumerants defined by the extension will have names of the form *NAME*`_ARB`.

Microsoft Corp. Exhibit 1009

F.2 Multitexture

Multitexture adds support for multiple texture units. The capabilities of the multiple texture units are identical, except that evaluation and feedback are supported only for texture unit 0. Each texture unit has its own state vector which includes texture vertex array specification, texture image and filtering parameters, and texture environment application.

The texture environments of the texture units are applied in a pipelined fashion whereby the output of one texture environment is used as the input fragment color for the next texture environment. Changes to texture client state and texture server state are each routed through one of two selectors which control which instance of texture state is affected.

The specification is written using four texture units though the actual number supported is implementation dependent and can be larger or smaller than four.

The name string for multitexture is `GL_ARB_multitexture`.

F.2.1 Dependencies

Multitexture requires features of OpenGL 1.1.

F.2.2 Issues

The extension currently requires a separate texture coordinate input for each texture unit. Modification to allow routing and/or broadcasting texcoords and **TexGen** output would be useful, possibly as a future extension layered on multitexture.

F.2.3 Changes to Section 2.6 (Begin/End Paradigm)

Amend paragraphs 2 and 3

Each vertex is specified with two, three, or four coordinates. In addition, a *current normal*, multiple *current texture coordinate sets*, and *current color* may be used in processing each vertex. Normals are used by the GL in lighting calculations; the current normal is a three-dimensional vector that may be set by sending three coordinates that specify it. Texture coordinates determine how a texture image is mapped onto a primitive. Multiple sets of texture coordinates may be used to specify how multiple texture images are mapped onto a primitive. The number of texture units supported is implementation dependent but must be at least one. The number of active textures supported can be queried with the state `MAX_TEXTURE_UNITS_ARB`.

Microsoft Corp. Exhibit 1009

Primary and secondary colors are associated with each vertex (see section 3.9). These *associated* colors are either based on the current color or produced by lighting, depending on whether or not lighting is enabled. Texture coordinates are similarly associated with each vertex. Multiple sets of texture coordinates may be associated with a vertex. Figure F.1 summarizes the association of auxiliary data with a transformed vertex to produce a *processed vertex*.

Amend paragraph 6

Before colors have been assigned to a vertex, the state required by a vertex is the vertex's coordinates, the current normal, the current edge flag (see section 2.6.2), the current material properties (see section 2.13.2), and the multiple current texture coordinate sets. Because color assignment is done vertex-by-vertex, a processed vertex comprises the vertex's coordinates, its edge flag, its assigned colors, and its multiple texture coordinate sets.

F.2.4 Changes to Section 2.7 (Vertex Specification)

Amend paragraph 2

Current values are used in associating auxiliary data with a vertex as described in section 2.6. A current value may be changed at any time by issuing an appropriate command. The commands

```
void TexCoord{1234}{sifd}( T coords );
void TexCoord{1234}{sifd}v( T coords );
```

specify the current homogeneous texture coordinates, named s, t, r, and q. The **TexCoord1** family of commands set the s coordinate to the provided single argument while setting t and r to 0 and q to 1. Similarly, **TexCoord2** sets s and t to the specified values, r to 0 and q to 1; **TexCoord3** sets s, t, and r, with q set to 1, and **TexCoord4** sets all four texture coordinates.

Implementations may support more than one texture unit, and thus more than one set of texture coordinates. The commands

```
void MultiTexCoord{1234}{sifd}ARB(enum texture,T
    coords)
void MultiTexCoord{1234}{sifd}vARB(enum texture,T
    coords)
```

take the coordinate set to be modified as the *texture* parameter. *texture* is a symbolic constant of the form `TEXTURE`i`_ARB`, indicating that texture coordinate set i is to be modified. The constants obey `TEXTURE`i`_ARB` =

Microsoft Corp. Exhibit 1009

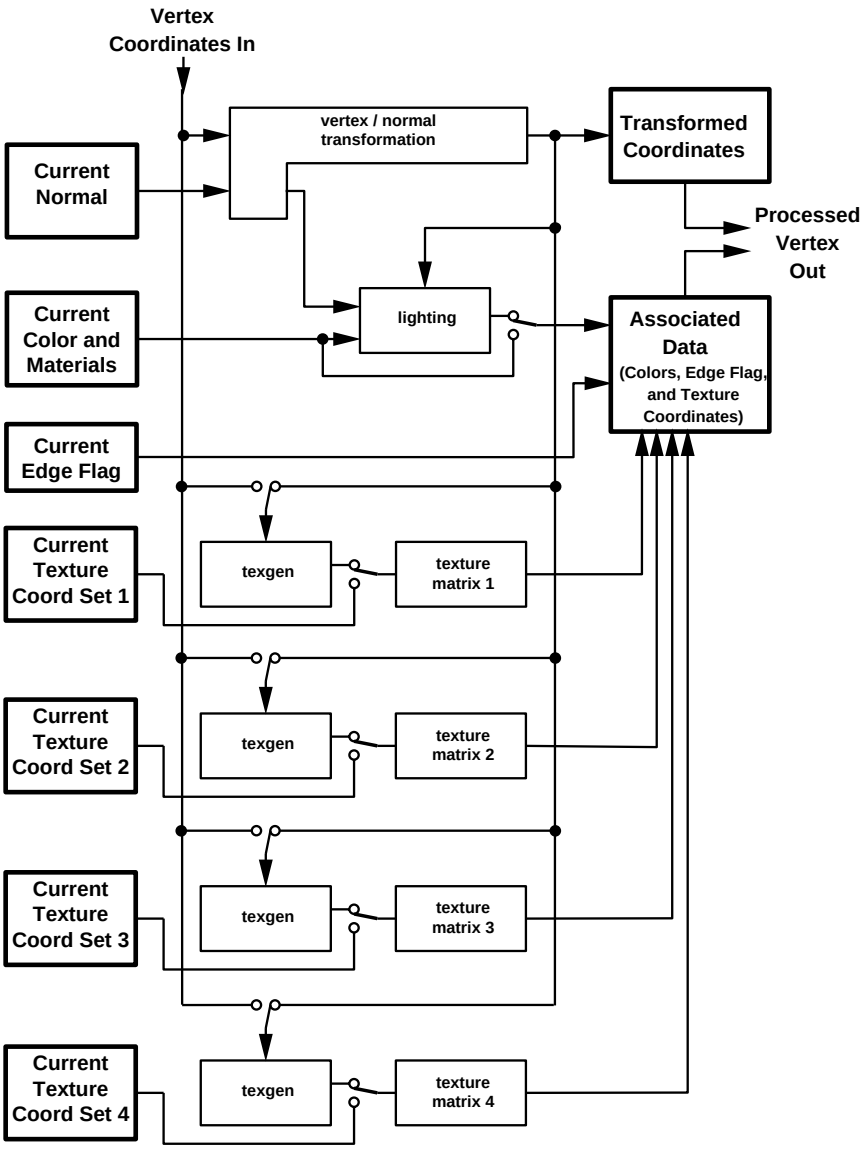


Figure F.1. Association of current values with a vertex. The heavy lined boxes represent GL state. Four texture units are shown; however, multitexturing may support a different number of units depending on the implementation.

Microsoft Corp. Exhibit 1009

TEXTURE0_ARB + i (i is in the range 0 to $k-1$, where k is the implementation-dependent number of texture units defined by MAX_TEXTURE_UNITS_ARB).

The **TexCoord** commands are exactly equivalent to the corresponding **MultiTexCoordARB** commands with *texture* set to TEXTURE0_ARB.

Gets of CURRENT_TEXTURE_COORDS return the texture coordinate set defined by the value of ACTIVE_TEXTURE_ARB.

Specifying an invalid texture coordinate set for the *texture* argument of **MultiTexCoordARB** results in undefined behavior.

F.2.5 Changes to Section 2.8 (Vertex Arrays)

Amend paragraph 1

The vertex specification commands described in section 2.7 accept data in almost any format, but their use requires many command executions to specify even simple geometry. Vertex data may also be placed into arrays that are stored in the client's address space. Blocks of data in these arrays may then be used to specify multiple geometric primitives through the execution of a single GL command. The client may specify up to 5 plus the value of MAX_TEXTURE_UNITS_ARB arrays: one each to store vertex coordinates, edge flags, colors, color indices, normals, and one or more texture coordinate sets. The commands ...

Insert between paragraph 2 and 3

In implementations which support more than one texture unit, the command

`void` **ClientActiveTextureARB**(`enum` *texture*);

is used to select the vertex array client state parameters to be modified by the **TexCoordPointer** command and the array affected by **EnableClientState** and **DisableClientState** with parameter TEXTURE_COORD_ARRAY. This command sets the client state variable CLIENT_ACTIVE_TEXTURE_ARB. Each texture unit has a client state vector which is selected when this command is invoked. This state vector includes the vertex array state. This call also selects which texture units' client state vector is used for queries of client state.

Specifying an invalid *texture* generates the error INVALID_ENUM. Valid values of *texture* are the same as for the **MultiTexCoordARB** commands described in section 2.7.

Amend final paragraph

Microsoft Corp. Exhibit 1009

If the number of supported texture units (the value of MAX_TEXTURE_UNITS_ARB) is k, then the client state required to implement vertex arrays consists of $5 + k$ boolean values, $5 + k$ memory pointers, $5 + k$ integer stride values, $4 + k$ symbolic constants representing array types, and $3 + k$ integers representing values per element. In the initial state, the boolean values are each disabled, the memory pointers are each null, the strides are each zero, the array types are each FLOAT, and the integers representing values per element are each four.

F.2.6 Changes to Section 2.10.2 (Matrices)

Amend paragraph 8

For each texture unit, a 4×4 matrix is applied to the corresponding texture coordinates. This matrix is applied as

$$\begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix} \begin{pmatrix} s \\ t \\ r \\ q \end{pmatrix},$$

where the left matrix is the current texture matrix. The matrix is applied to the coordinates resulting from texture coordinate generation (which may simply be the current texture coordinates), and the resulting transformed coordinates become the texture coordinates associated with a vertex. Setting the matrix mode to TEXTURE causes the already described matrix operations to apply to the texture matrix.

There is also a corresponding texture matrix stack for each texture unit. To change the stack affected by matrix operations, set the *active texture unit selector* by calling

`void` **ActiveTextureARB**(`enum` *texture*);

The selector also affects calls modifying texture environment state, texture coordinate generation state, texture binding state, and queries of all these state values as well as current texture coordinates and current raster texture coordinates.

Specifying an invalid *texture* generates the error INVALID_ENUM. Valid values of *texture* are the same as for the **MultiTexCoordARB** commands described in section 2.7.

The active texture unit selector may be queried by calling **GetIntegerv** with *pname* set to ACTIVE_TEXTURE_ARB.

Microsoft Corp. Exhibit 1009

There is a stack of matrices for each of matrix modes MODELVIEW, PROJECTION, and COLOR, and for each texture unit. For MODELVIEW mode, the stack depth is at least 32 (that is, there is a stack of at least 32 modelview matrices). For the other modes, the depth is at least 2. Texture matrix stacks for all texture units have the same depth. The current matrix in any mode is the matrix on the top of the stack for that mode.

`void` **PushMatrix**(`void`);

pushes the stack down by one, duplicating the current matrix in both the top of the stack and the entry below it.

`void` **PopMatrix**(`void`);

pops the top entry off of the stack, replacing the current matrix with the matrix that was the second entry in the stack. The pushing or popping takes place on the stack corresponding to the current matrix mode. Popping a matrix off a stack with only one entry generates the error STACK_UNDERFLOW; pushing a matrix onto a full stack generates STACK_OVERFLOW.

When the current matrix mode is TEXTURE, the texture matrix stack of the active texture unit is pushed or popped.

The state required to implement transformations consists of a four-valued integer indicating the current matrix mode, one stack of at least two 4×4 matrices for each of COLOR, PROJECTION, each texture unit, TEXTURE, and a stack of at least 32 4×4 matrices for MODELVIEW. Each matrix stack has an associated stack pointer. Initially, there is only one matrix on each stack, and all matrices are set to the identity. The initial matrix mode is MODELVIEW. The initial value of ACTIVE_TEXTURE_ARB is TEXTURE0_ARB.

F.2.7 Changes to Section 2.10.4 (Generating Texture Coordinates)

Amend paragraph 4

The state required for texture coordinate generation for each texture unit comprises a three-valued integer for each coordinate indicating coordinate generation mode, and a bit for each coordinate to indicate whether texture coordinate generation is enabled or disabled. In addition, four coefficients are required for the four coordinates for each of EYE_LINEAR and OBJECT_LINEAR. The initial state has the texture generation function disabled for all texture coordinates. The initial values of p_i for s are all 0 except p_1 which is one; for t all the p_i are zero except p_2, which is 1. The values of p_i for r and q are all 0. These values of p_i apply for both

Microsoft Corp. Exhibit 1009

the EYE_LINEAR and OBJECT_LINEAR versions. Initially all texture generation modes are EYE_LINEAR.

For implementations which support more than one texture unit, there is texture coordinate generation state for each unit. The texture coordinate generation state which is affected by the **TexGen**, **Enable**, and **Disable** operations is set with **ActiveTextureARB**.

F.2.8 Changes to Section 2.12 (Current Raster Position)

Amend paragraph 2

The state required for the current raster position consists of three window coordinates x_w, y_w, and z_w, a clip coordinate w_c value, an eye coordinate distance, a valid bit, and associated data consisting of a color and multiple texture coordinate sets. It is set using one of the **RasterPos** commands:

```
void RasterPos{234}{sifd}( T coords );
void RasterPos{234}{sifd}v( T coords );
```

RasterPos4 takes four values indicating x, y, z, and w. **RasterPos3** (or **RasterPos2**) is analogous, but sets only x, y, and z with w implicitly set to 1 (or only x and y with z implicitly set to 0 and w implicitly set to 1).

Gets of CURRENT_RASTER_TEXTURE_COORDS are affected by the setting of the state ACTIVE_TEXTURE_ARB.

Modify figure 2.7

Amend paragraph 5

The current raster position requires five single-precision floating-point values for its x_w, y_w, and z_w window coordinates, its w_c clip coordinate, and its eye coordinate distance, a single valid bit, a color (RGBA and color index), and texture coordinates for each texture unit. In the initial state, the coordinates and texture coordinates are all $(0, 0, 0, 1)$, the eye coordinate distance is 0, the valid bit is set, the associated RGBA color is $(1, 1, 1, 1)$ and the associated color index color is 1. In RGBA mode, the associated color index always has its initial value; in color index mode, the RGBA color always maintains its initial value.

F.2.9 Changes to Section 3.8 (Texturing)

Amend paragraphs 1 and 2

Texturing maps a portion of one or more specified images onto each primitive for which texturing is enabled. This mapping is accomplished by using the color of an image at the location indicated by a fragment's (s, t, r)

Microsoft Corp. Exhibit 1009

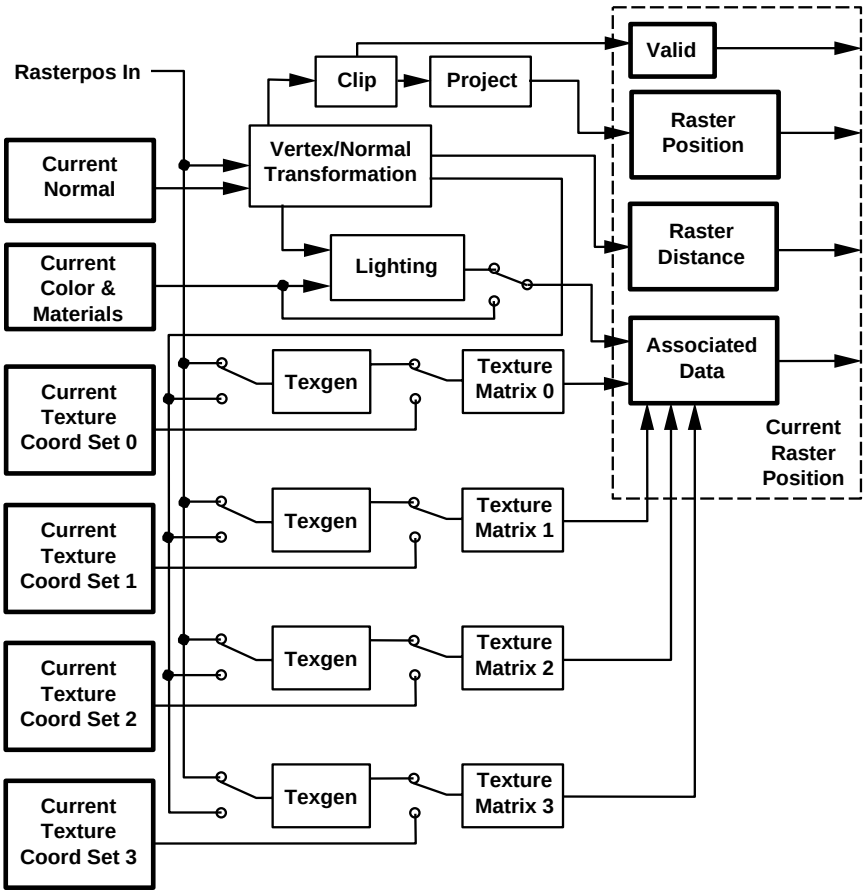


Figure F.2. The current raster position and how it is set. Four texture units are shown; however, multitexturing may support a different number of units depending on the implementation.

Microsoft Corp. Exhibit 1009

coordinates to modify the fragment's primary RGBA color. Texturing does not affect the secondary color.

An implementation may support texturing using more than one image at a time. In this case the fragment carries multiple sets of texture coordinates (s, t, r) which are used to index separate images to produce color values which are collectively used to modify the fragment's RGBA color. Texturing is specified only for RGBA mode; its use in color index mode is undefined. The following subsections (up to and including Section 3.8.5) specify the GL operation with a single texture and Section 3.8.10 specifies the details of how multiple texture units interact.

F.2.10 Changes to Section 3.8.5 (Texture Minification)

Amend second paragraph under the **Mipmapping** *subheading*

Each array in a mipmap is defined using **TexImage3D**, **TexImage2D**, **CopyTexImage2D**, **TexImage1D**, or **CopyTexImage1D**; the array being set is indicated with the level-of-detail argument *level*. Level-of-detail numbers proceed from `TEXTURE_BASE_LEVEL` for the original texture array through $p = \max\{n, m, l\} +$ `TEXTURE_BASE_LEVEL` with each unit increase indicating an array of half the dimensions of the previous one as already described. If texturing is enabled (and `TEXTURE_MIN_FILTER` is one that requires a mipmap) at the time a primitive is rasterized and if the set of arrays `TEXTURE_BASE_LEVEL` through $q = \min\{p,$ `TEXTURE_MAX_LEVEL`$\}$ is incomplete, then it is as if texture mapping were disabled for that texture unit. The set of arrays `TEXTURE_BASE_LEVEL` through q is incomplete if the internal formats of all the mipmap arrays were not specified with the same symbolic constant, if the border widths of the mipmap arrays are not the same, if the dimensions of the mipmap arrays do not follow the sequence described above, if `TEXTURE_MAX_LEVEL` $<$ `TEXTURE_BASE_LEVEL`, or if `TEXTURE_BASE_LEVEL` $> p$. Array levels k where $k <$ `TEXTURE_BASE_LEVEL` or $k > q$ are insignificant.

F.2.11 Changes to Section 3.8.8 (Texture Objects)

Insert following the last paragraph

The texture object name space, including the initial one-, two-, and three-dimensional texture objects, is shared among all texture units. A texture object may be bound to more than one texture unit simultaneously. After a texture object is bound, any GL operations on that target object affect any other texture units to which the same texture object is bound.

Texture binding is affected by the setting of the state `ACTIVE_TEXTURE_ARB`.

Microsoft Corp. Exhibit 1009

If a texture object is deleted, it as if all texture units which are bound to that texture object are rebound to texture object zero.

F.2.12 Changes to Section 3.8.10 (Texture Application)

Amend second paragraph

Each texture unit is enabled and bound to texture objects independently from the other texture units. Each texture unit follows the precendence rules for one-, two-, and three-dimensional textures. Thus texture units can be performing texture mapping of different dimensionalities simultaneously. Each unit has its own enable and binding states.

Each texture unit is paired with an environment function, as shown in figure F.3. The second texture function is computed using the texture value from the second texture, the fragment resulting from the first texture function computation and the second texture unit's environment function. If there is a third texture, the fragment resulting from the second texture function is combined with the third texture value using the third texture unit's environment function and so on. The texture unit selected by **ActiveTextureARB** determines which texture unit's environment is modified by **TexEnv** calls.

Texturing is enabled and disabled individually for each texture unit. If texturing is disabled for one of the units, then the fragment resulting from the previous unit, is passed unaltered to the following unit.

The required state, per texture unit, is three bits indicating whether each of one-, two-, or three-dimensional texturing is enabled or disabled. In the intial state, all texturing is disabled for all texture units.

F.2.13 Changes to Section 5.1 (Evaluators)

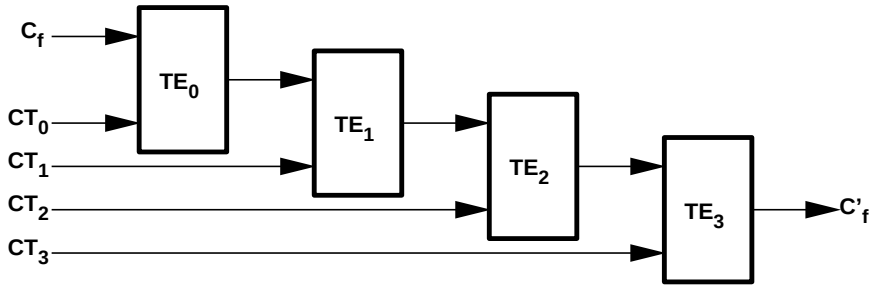
Amend paragraph 7

The evaluation of a defined map is enabled or disabled with **Enable** and **Disable** using the constant corresponding to the map as described above. The evaluator map generates only coordinates for texture unit `TEXTURE0_ARB`. The error `INVALID_VALUE` results if either *ustride* or *vstride* is less than k, or if u_1 is equal to $u2$, or if v_1 is equal to v_2. If the value of `ACTIVE_TEXTURE_ARB` is not `TEXTURE0_ARB`, calling **Map[12]** generates the error `INVALID_OPERATION`.

F.2.14 Changes to Section 5.3 (Feedback)

Amend paragraph 4

Microsoft Corp. Exhibit 1009



C_f = fragment color input to texturing

C'_f = fragment color output from texturing

CT_i = texture color from texture lookup *i*

TE_i = texture environment *i*

Figure F.3. Multitexture pipeline. Four texture units are shown; however, multitexturing may support a different number of units depending on the implementation. The input fragment color is successively combined with each texture according to the state of the corresponding texture environment, and the resulting fragment color passed as input to the next texture unit in the pipeline.

Microsoft Corp. Exhibit 1009

The texture coordinates and colors returned are those resulting from the clipping operations described in Section 2.13.8. Only coordinates for texture unit TEXTURE0_ARB are returned even for implementations which support multiple texture units. The colors returned are the primary colors.

F.2.15 Changes to Section 6.1.2 (Data Conversions)

Insert following the last paragraph

Most texture state variables are qualified by the value of ACTIVE_TEXTURE_ARB to determine which server texture state vector is queried. Client texture state variables such as texture coordinate array pointers are qualified by the value of CLIENT_ACTIVE_TEXTURE_ARB. Tables 6.5, 6.6, 6.7, 6.12, 6.14, and 6.25 indicate those state variables which are qualified by ACTIVE_TEXTURE_ARB or CLIENT_ACTIVE_TEXTURE_ARB during state queries.

F.2.16 Changes to Section 6.1.12 (Saving and Restoring State)

Insert following paragraph 3

Operations on groups containing replicated texture state push or pop texture state within that group for all texture units. When state for a group is pushed, all state corresponding to TEXTURE0_ARB is pushed first, followed by state corresponding to TEXTURE1_ARB, and so on up to and including the state corresponding to TEXTUREk_ARB where $k+1$ is the value of MAX_TEXTURE_UNITS_ARB. When state for a group is popped, the replicated texture state is restored in the opposite order that it was pushed, starting with state corresponding to TEXTUREk_ARB and ending with TEXTURE0_ARB. Identical rules are observed for client texture state push and pop operations. Matrix stacks are never pushed or popped with **PushAttrib**, **PushClientAttrib**, **PopAttrib**, or **PopClientAttrib**.

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
Modified state in table 6.5						
CURRENT_TEXTURE_COORDS	$1 * \times T$	**GetFloatv**	0,0,0,1	Current texture coordinates	2.7	current
CURRENT_RASTER_TEXTURE_COORDS	$1 * \times T$	**GetFloatv**	0,0,0,1	Texture coordinates associated with raster position	2.12	current
Modified state in table 6.6						
TEXTURE_COORD_ARRAY	$1 * \times B$	**IsEnabled**	*False*	Texture coordinate array enable	2.8	vertex-array
TEXTURE_COORD_ARRAY_SIZE	$1 * \times Z^+$	**GetIntegerv**	4	Coordinates per element	2.8	vertex-array
TEXTURE_COORD_ARRAY_TYPE	$1 * \times Z_4$	**GetIntegerv**	FLOAT	Type of texture coordinates	2.8	vertex-array
TEXTURE_COORD_ARRAY_STRIDE	$1 * \times Z^+$	**GetIntegerv**	0	Stride between texture coordinates	2.8	vertex-array
TEXTURE_COORD_ARRAY_POINTER	$1 * \times Y$	**GetPointerv**	0	Pointer to the texture coordinate array	2.8	vertex-array

Table F.1. Changes to State Tables

Microsoft Corp. Exhibit 1009

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
Modified state in table 6.7						
TEXTURE_MATRIX	$1 * \times 2 * \times M^4$	**GetFloatv**	Identity	Texture matrix stack	2.10.2	–
TEXTURE_STACK_DEPTH	$1 * \times Z^+$	**GetIntegerv**	1	Texture matrix stack pointer	2.10.2	–
Modified state in table 6.12						
TEXTURE_xD	$1 * \times 3 \times B$	**IsEnabled**	*False*	True if xD texturing is enabled; x is 1, 2, or 3	3.8.10	texture/enable
TEXTURE_BINDING_xD	$1 * \times 3 \times Z^+$	**GetIntegerv**	0	Texture object bound to TEXTURE_xD	3.8.8	texture
Modified state in table 6.14						
TEXTURE_ENV_MODE	$1 * \times Z_4$	**GetTexEnviv**	MODULATE	Texture application function	3.8.9	texture
TEXTURE_ENV_COLOR	$1 * \times C$	**GetTexEnvfv**	0,0,0,0	Texture environment color	3.8.9	texture
TEXTURE_GEN_x	$1 * \times 4 \times B$	**IsEnabled**	*False*	Texgen enabled (x is S, T, R, or Q)	2.10.4	texture/enable
EYE_PLANE	$1 * \times 4 \times R^4$	**GetTexGenfv**	see 2.10.4	Texgen plane equation coefficients (for S, T, R, and Q)	2.10.4	texture
OBJECT_PLANE	$1 * \times 4 \times R^4$	**GetTexGenfv**	see 2.10.4	Texgen object linear coefficients (for S, T, R, and Q)	2.10.4	texture
TEXTURE_GEN_MODE	$1 * \times 4 \times Z_3$	**GetTexGeniv**	EYE_LINEAR	Function used for texgen (for S, T, R, and Q	2.10.4	texture

Table F.2. Changes to State Tables (cont.)

Microsoft Corp. Exhibit 1009

APPENDIX T

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
Added to table 6.6						
CLIENT_ACTIVE_TEXTURE_ARB	Z_{1*}	**GetIntegerv**	TEXTURE0_ARB	Client active texture unit selector	2.7	vertex-array
Added to table 6.14						
ACTIVE_TEXTURE_ARB	Z_{1*}	**GetIntegerv**	TEXTURE0_ARB	Active texture unit selector	2.7	texture

Table F.3. New State Introduced by Multitexture

Microsoft Corp. Exhibit 1009

Get value Added to table 6.25	Type	Get Cmnd	Minimum Value	Description	Sec.	Attribute
MAX_TEXTURE_UNITS_ARB	Z^+	**GetIntegerv**	1	Number of texture units (not to exceed 32)	2.6	–

Table F.4. New Implementation-Dependent Values Introduced by Multitexture

Microsoft Corp. Exhibit 1009

Index of OpenGL Commands

x_BIAS, 78, 208
x_SCALE, 78, 208
2D, 174, 176, 217
2_BYTES, 177
3D, 174, 176
3D_COLOR, 174, 176
3D_COLOR_TEXTURE, 174, 176
3_BYTES, 177
4D_COLOR_TEXTURE, 174, 176
4_BYTES, 177

1, 113, 120, 131, 136, 137, 185, 202, 253
2, 113, 120, 136, 137, 185, 202, 253
3, 113, 120, 136, 137, 185, 202, 253
4, 113, 120, 136, 137, 185

ACCUM, 155
Accum, 155, 156
ACCUM_BUFFER_BIT, 154, 191
ACTIVE_TEXTURE_ARB, 243–246, 248, 249, 251
ActiveTextureARB, 244, 246, 249
ADD, 155, 156
ALL_ATTRIB_BITS, 191
ALL_CLIENT_ATTRIB_BITS, 191
ALPHA, 78, 92, 103, 104, 114, 115, 136, 137, 159, 160, 185, 208, 210, 216, 226, 232
ALPHA12, 115
ALPHA16, 115
ALPHA4, 115
ALPHA8, 115
ALPHA_BIAS, 101
ALPHA_SCALE, 101
ALPHA_TEST, 143
AlphaFunc, 143
ALWAYS, 143–145, 205
AMBIENT, 50, 51
AMBIENT_AND_DIFFUSE, 50, 51, 53
AND, 151
AND_INVERTED, 151
AND_REVERSE, 151
AreTexturesResident, 134, 178
ArrayElement, 19, 23, 24, 175
AUTO_NORMAL, 167
AUXi, 151, 152
AUXn, 151, 158
AUX0, 151, 158

BACK, 49, 51, 52, 70, 73, 151, 152, 158, 159, 183, 201
BACK_LEFT, 151, 152, 158
BACK_RIGHT, 151, 152, 158
Begin, 12, 15–20, 23, 24, 28, 55, 62, 67, 70, 73, 168, 169, 174
BGR, 92, 159, 162
BGRA, 92, 94, 98, 159, 230
BindTexture, 133
BITMAP, 72, 80, 83, 90, 91, 98, 110, 160, 185
Bitmap, 110
BITMAP_TOKEN, 176
BLEND, 135, 137, 146, 150
BlendColor, 77, 146
BlendEquation, 77, 146, 147
BlendFunc, 77, 146, 147, 149
BLUE, 78, 92, 159, 160, 208, 210, 216
BLUE_BIAS, 101
BLUE_SCALE, 101
BYTE, 22, 91, 160, 161, 177

Microsoft Corp. Exhibit 1009

C3F_V3F, 25, 26
C4F_N3F_V3F, 25, 26
C4UB_V2F, 25, 26
C4UB_V3F, 25, 26
CallList, 19, 177, 178
CallLists, 19, 177, 178
CCW, 48, 201
CLAMP, 124, 127
CLAMP_TO_EDGE, 124, 125, 127, 231
CLEAR, 151
Clear, 153, 154
ClearAccum, 154
ClearColor, 154
ClearDepth, 154
ClearIndex, 154
ClearStencil, 154
CLIENT_ACTIVE_TEXTURE_ARB, 243, 251
CLIENT_PIXEL_STORE_BIT, 191
CLIENT_VERTEX_ARRAY_BIT, 191
ClientActiveTextureARB, 243
CLIP_PLANE*i*, 39
CLIP_PLANE0, 39
ClipPlane, 38
COEFF, 184
COLOR, 31, 34, 81, 85, 86, 120, 162, 245
Color, 19–21, 43, 56
Color3, 20
Color4, 20
COLOR_ARRAY, 23, 27
COLOR_ARRAY_POINTER, 189
COLOR_BUFFER_BIT, 153, 191
COLOR_INDEX, 72, 80, 83, 90, 92, 102, 110, 159, 162, 184, 185
COLOR_INDEXES, 50, 54
COLOR_LOGIC_OP, 150
COLOR_MATERIAL, 51, 53
COLOR_MATRIX, 185
COLOR_MATRIX_STACK_DEPTH, 185
COLOR_TABLE, 80, 82, 103
COLOR_TABLE_ALPHA_SIZE, 186
COLOR_TABLE_BIAS, 80, 81, 186
COLOR_TABLE_BLUE_SIZE, 186
COLOR_TABLE_FORMAT, 186
COLOR_TABLE_GREEN_SIZE, 186
COLOR_TABLE_INTENSITY_SIZE, 186
COLOR_TABLE_LUMINANCE_SIZE, 186
COLOR_TABLE_RED_SIZE, 186
COLOR_TABLE_SCALE, 80, 81, 186
COLOR_TABLE_WIDTH, 186
ColorMask, 152, 153
ColorMaterial, 51–53, 167, 223, 228
ColorPointer, 19, 21, 22, 27, 178
ColorSubTable, 81, 82
ColorTable, 79, 81–83, 108, 109, 179
ColorTableParameter, 80
ColorTableParameterfv, 80
Colorub, 56
Colorui, 56
Colorus, 56
COMPILE, 175, 223
COMPILE_AND_EXECUTE, 175, 177, 178
CONSTANT_ALPHA, 77, 148, 149
CONSTANT_ATTENUATION, 50
CONSTANT_BORDER, 105, 106
CONSTANT_COLOR, 77, 148, 149
CONVOLUTION_1D, 84, 86, 103, 117, 186, 187
CONVOLUTION_2D, 83–85, 103, 117, 186, 187
CONVOLUTION_BORDER_COLOR, 106, 187
CONVOLUTION_BORDER_MODE, 105, 187
CONVOLUTION_FILTER_BIAS, 83–85, 187
CONVOLUTION_FILTER_SCALE, 83–86, 187
CONVOLUTION_FORMAT, 187
CONVOLUTION_HEIGHT, 187
CONVOLUTION_WIDTH, 187
ConvolutionFilter1D, 84–86
ConvolutionFilter2D, 83–86

Microsoft Corp. Exhibit 1009

ConvolutionParameter, 84, 105
ConvolutionParameterfv, 83, 84, 106
ConvolutionParameteriv, 85, 106
COPY, 150, 151, 205
COPY_INVERTED, 151
COPY_PIXEL_TOKEN, 176
CopyColorSubTable, 81, 82
CopyColorTable, 81, 82
CopyConvolutionFilter1D, 85
CopyConvolutionFilter2D, 85
CopyPixels, 75, 78, 81, 85, 86, 103, 120, 156, 162, 163, 173
CopyTexImage1D, 103, 120, 121, 129, 248
CopyTexImage2D, 103, 118, 120, 121, 129, 248
CopyTexImage3D, 121
CopyTexSubImage1D, 103, 121, 123
CopyTexSubImage2D, 103, 121, 122
CopyTexSubImage3D, 103, 121, 122
CULL_FACE, 70
CullFace, 70
CURRENT_BIT, 191
CURRENT_RASTER_TEXTURE_COORDS, 222, 246
CURRENT_TEXTURE_COORDS, 243
CW, 48

DECAL, 135, 137
DECR, 144
DeleteLists, 178
DeleteTextures, 133, 178
DEPTH, 162, 208
DEPTH_BIAS, 78, 101
DEPTH_BUFFER_BIT, 153, 191
DEPTH_COMPONENT, 80, 83, 90, 92, 112, 158, 159, 162, 184
DEPTH_SCALE, 78, 101
DEPTH_TEST, 145
DepthFunc, 145
DepthMask, 153
DepthRange, 30, 182, 224
DIFFUSE, 50, 51
Disable, 35, 38, 39, 44, 51, 60, 64, 67, 70, 72, 74, 108, 109, 138, 143–146, 149, 150, 166, 167, 246, 249
DisableClientState, 19, 23, 27, 178, 243
DITHER, 150
DOMAIN, 184
DONT_CARE, 180, 213
DOUBLE, 22
DRAW_PIXEL_TOKEN, 176
DrawArrays, 23, 24, 175
DrawBuffer, 151, 152
DrawElements, 24, 25, 175, 232
DrawPixels, 72, 75, 76, 78, 80, 83, 89–93, 98, 100, 103, 110, 112, 113, 156, 158, 160, 162, 173
DrawRangeElements, 25, 215
DST_ALPHA, 148
DST_COLOR, 148

EDGE_FLAG_ARRAY, 23, 27
EDGE_FLAG_ARRAY_POINTER, 189
EdgeFlag, 18, 19
EdgeFlagPointer, 19, 21, 22, 178
EdgeFlagv, 18
EMISSION, 50, 51
Enable, 35, 38, 39, 44, 51, 60, 64, 67, 70, 72, 74, 108, 109, 138, 143–146, 149, 150, 166, 167, 181, 246, 249
ENABLE_BIT, 191
EnableClientState, 19, 23, 27, 178, 243
End, 12, 15–20, 23, 24, 28, 55, 62, 70, 73, 168, 169, 174
EndList, 175, 177
EQUAL, 143–145
EQUIV, 151
EVAL_BIT, 191
EvalCoord, 19, 167
EvalCoord1, 167–169
EvalCoord1d, 168
EvalCoord1f, 168

Microsoft Corp. Exhibit 1009

EvalCoord2, 167, 169, 170
EvalMesh1, 168
EvalMesh2, 168, 169
EvalPoint, 19
EvalPoint1, 169
EvalPoint2, 170
EXP, 139, 140, 198
EXP2, 139
EXT_bgra, 230
EXT_blend_color, 234
EXT_blend_logic_op, 226
EXT_blend_minmax, 234
EXT_blend_subtract, 234
EXT_color_subtable, 233
EXT_color_table, 233
EXT_convolution, 233
EXT_copy_texture, 227
EXT_draw_range_elements, 232
EXT_histogram, 234
EXT_packed_pixels, 231
EXT_polygon_offset, 226
EXT_rescale_normal, 231
EXT_separate_specular_color, 231
EXT_subtexture, 227
EXT_texture, 226, 227
EXT_texture3D, 230
EXT_texture_object, 227
EXT_vertex_array, 225
EXTENSIONS, 77, 189, 239
EYE_LINEAR, 37, 38, 183, 204, 245, 246, 253
EYE_PLANE, 37

FALSE, 18, 19, 46–48, 76, 78, 87, 88, 98, 101, 109, 110, 134, 158, 182, 184, 187, 188
FASTEST, 180
FEEDBACK, 171, 173, 174, 224
FEEDBACK_BUFFER_POINTER, 189
FeedbackBuffer, 173, 174, 178
FILL, 73–75, 169, 201, 223, 226
Finish, 178, 179, 222
FLAT, 54, 223
FLOAT, 22, 26, 27, 91, 160, 161, 177, 196, 244, 252
Flush, 178, 179, 222
FOG, 138
Fog, 139, 140
FOG_BIT, 191
FOG_COLOR, 139
FOG_DENSITY, 139
FOG_END, 139
FOG_HINT, 180
FOG_INDEX, 140
FOG_MODE, 139, 140
FOG_START, 139
FRONT, 49, 51, 70, 73, 151, 152, 158, 159, 183
FRONT_AND_BACK, 49, 51–53, 70, 73, 151, 152
FRONT_LEFT, 151, 152, 158
FRONT_RIGHT, 151, 152, 158
FrontFace, 48, 70
Frustum, 32, 33, 223
FUNC_ADD, 147, 149, 205
FUNC_REVERSE_SUBTRACT, 147
FUNC_SUBTRACT, 147

GenLists, 178
GenTextures, 133, 134, 178, 184
GEQUAL, 143–145
Get, 30, 178, 181, 182, 243, 246
GetBooleanv, 181, 182, 193
GetClipPlane, 182, 183
GetColorTable, 83, 158, 185
GetColorTableParameter, 186
GetConvolutionFilter, 158, 186
GetConvolutionParameter, 187
GetConvolutionParameteriv, 83, 84
GetDoublev, 181, 182, 193
GetError, 11
GetFloatv, 181, 182, 185, 193
GetHistogram, 88, 158, 187
GetHistogramParameter, 188
GetIntegerv, 25, 181, 182, 185, 193, 244
GetLight, 182, 183
GetMap, 183

Microsoft Corp. Exhibit 1009

GetMaterial, 182, 183
GetMinmax, 158, 188
GetMinmaxParameter, 188
GetPixelMap, 183
GetPointerv, 189
GetPolygonStipple, 185
GetSeparableFilter, 158, 186
GetString, 189
GetTexEnv, 182, 183
GetTexGen, 182, 183
GetTexImage, 103, 132, 184, 186–188
GetTexImage1D, 158
GetTexImage2D, 158
GetTexImage3D, 158
GetTexLevelParameter, 182, 183
GetTexParameter, 182, 183
GetTexParameterfv, 132, 134
GetTexParameteriv, 132, 134
GL_ARB_multitexture, 240
GREATER, 143–145
GREEN, 78, 92, 159, 160, 208, 210, 216
GREEN_BIAS, 101
GREEN_SCALE, 101

Hint, 179
HINT_BIT, 191
HISTOGRAM, 87, 88, 109, 187, 188
Histogram, 87, 88, 109, 179
HISTOGRAM_ALPHA_SIZE, 188
HISTOGRAM_BLUE_SIZE, 188
HISTOGRAM_FORMAT, 188
HISTOGRAM_GREEN_SIZE, 188
HISTOGRAM_LUMINANCE_SIZE, 188
HISTOGRAM_RED_SIZE, 188
HISTOGRAM_SINK, 188
HISTOGRAM_WIDTH, 188
HP_convolution_border_modes, 233

INCR, 144
INDEX, 216
Index, 19, 21
INDEX_ARRAY, 23, 27
INDEX_ARRAY_POINTER, 189
INDEX_LOGIC_OP, 150
INDEX_OFFSET, 78, 101, 208
INDEX_SHIFT, 78, 101, 208
IndexMask, 152, 153
IndexPointer, 19, 22, 178
InitNames, 171
INT, 22, 91, 160, 161, 177
INTENSITY, 87, 88, 103, 104, 114, 115, 136, 137, 185, 208, 226
INTENSITY12, 87, 88, 115
INTENSITY16, 87, 88, 115
INTENSITY4, 87, 88, 115
INTENSITY8, 87, 88, 115
InterleavedArrays, 19, 25, 26, 178
INVALID_ENUM, 12, 13, 38, 49, 77, 83, 87, 88, 90, 120, 132, 184, 243, 244
INVALID_OPERATION, 13, 19, 77, 90, 94, 133, 151, 156, 158, 159, 171, 173, 175, 249
INVALID_VALUE, 12, 13, 22, 25, 30, 33, 49, 60, 64, 76, 78–80, 82–84, 87, 113, 114, 116, 121–123, 130, 134, 139, 143, 154, 165, 166, 168, 175, 177, 183, 184, 249
INVERT, 144, 151
IsEnabled, 178, 181, 193
IsList, 178
IsTexture, 178, 184

KEEP, 144, 145, 205

LEFT, 151, 152, 158
LEQUAL, 143–145
LESS, 143–145, 205
Light, 49, 50
LIGHT*i*, 49, 51, 224
LIGHT0, 49
LIGHT_MODEL_AMBIENT, 50
LIGHT_MODEL_COLOR_CONTROL, 50
LIGHT_MODEL_LOCAL_VIEWER, 50
LIGHT_MODEL_TWO_SIDE, 50

Microsoft Corp. Exhibit 1009

LIGHTING, 44
LIGHTING_BIT, 191
LightModel, 49, 50
LINE, 73–75, 168, 169, 201, 226
LINE_BIT, 191
LINE_LOOP, 15
LINE_RESET_TOKEN, 176
LINE_SMOOTH, 64
LINE_SMOOTH_HINT, 180
LINE_STIPPLE, 67
LINE_STRIP, 15, 168
LINE_TOKEN, 176
LINEAR, 124, 127, 130, 131, 139
LINEAR_ATTENUATION, 50
LINEAR_MIPMAP_LINEAR, 124, 129, 130
LINEAR_MIPMAP_NEAREST, 124, 129, 130
LINES, 16, 67
LineStipple, 66
LineWidth, 62
LIST_BIT, 191
ListBase, 178, 179
LOAD, 155
LoadIdentity, 31
LoadMatrix, 31, 32
LoadName, 171
LOGIC_OP, 150
LogicOp, 150, 151
LUMINANCE, 92, 99, 103, 104, 113–115, 136, 137, 159, 160, 185, 208, 210, 226
LUMINANCE12, 115
LUMINANCE12_ALPHA12, 115
LUMINANCE12_ALPHA4, 115
LUMINANCE16, 115
LUMINANCE16_ALPHA16, 115
LUMINANCE4, 115
LUMINANCE4_ALPHA4, 115
LUMINANCE6_ALPHA2, 115
LUMINANCE8, 115
LUMINANCE8_ALPHA8, 115
LUMINANCE_ALPHA, 92, 99, 103, 104, 113–115, 136, 137, 159, 160, 162, 185
Map1, 165, 166, 182
MAP1_COLOR_4, 165
MAP1_INDEX, 165
MAP1_NORMAL, 165
MAP1_TEXTURE_COORD_1, 165, 167
MAP1_TEXTURE_COORD_2, 165, 167
MAP1_TEXTURE_COORD_3, 165
MAP1_TEXTURE_COORD_4, 165
MAP1_VERTEX_3, 165
MAP1_VERTEX_4, 165
Map2, 165, 166, 182
MAP2_VERTEX_3, 167
MAP2_VERTEX_4, 167
Map[12], 249
MAP_COLOR, 78, 101, 102
MAP_STENCIL, 78, 102
MAP_VERTEX_3, 167
MAP_VERTEX_4, 167
MapGrid1, 168
MapGrid2, 168
Material, 19, 49, 50, 54, 223
MatrixMode, 31
MAX, 147
MAX_3D_TEXTURE_SIZE, 116
MAX_ATTRIB_STACK_DEPTH, 190
MAX_CLIENT_ATTRIB_STACK_DEPTH, 190
MAX_COLOR_MATRIX_STACK_DEPTH, 185
MAX_CONVOLUTION_HEIGHT, 83, 187
MAX_CONVOLUTION_WIDTH, 83, 84, 187
MAX_ELEMENTS_INDICES, 25
MAX_ELEMENTS_VERTICES, 25
MAX_EVAL_ORDER, 165, 166
MAX_PIXEL_MAP_TABLE, 79, 101
MAX_TEXTURE_SIZE, 116
MAX_TEXTURE_UNITS_ARB, 240, 243, 244, 251
MIN, 147
MINMAX, 88, 109, 188

Microsoft Corp. Exhibit 1009

Minmax, 88, 110
MINMAX_FORMAT, 188
MINMAX_SINK, 188
MODELVIEW, 31, 34, 245
MODULATE, 135, 136
MULT, 155, 156
MultiTexCoord, 241
MultiTexCoordARB, 243, 244
MultMatrix, 31, 32

N3F_V3F, 25, 26
NAND, 151
NEAREST, 124, 127, 130, 131
NEAREST_MIPMAP_LINEAR, 124, 129–131
NEAREST_MIPMAP_NEAREST, 124, 129–131
NEVER, 143–145
NewList, 175, 177, 178
NICEST, 180
NO_ERROR, 11, 12
NONE, 151, 152
NOOP, 151
NOR, 151
Normal, 19, 20
Normal3, 8, 9, 20
Normal3d, 8
Normal3dv, 9
Normal3f, 8
Normal3fv, 9
NORMAL_ARRAY, 23, 27
NORMAL_ARRAY_POINTER, 189
NORMALIZE, 35
NormalPointer, 19, 22, 27, 178
NOTEQUAL, 143–145

OBJECT_LINEAR, 37, 38, 183, 245, 246
OBJECT_PLANE, 37
ONE, 148, 149, 205
ONE_MINUS_CONSTANT_ALPHA, 77, 148, 149
ONE_MINUS_CONSTANT_COLOR, 77, 148, 149
ONE_MINUS_DST_ALPHA, 148
ONE_MINUS_DST_COLOR, 148
ONE_MINUS_SRC_ALPHA, 148
ONE_MINUS_SRC_COLOR, 148
OR, 151
OR_INVERTED, 151
OR_REVERSE, 151
ORDER, 184
Ortho, 32, 33, 223
OUT_OF_MEMORY, 12, 13, 177

PACK_ALIGNMENT, 158, 207
PACK_IMAGE_HEIGHT, 158, 184, 207
PACK_LSB_FIRST, 158, 207
PACK_ROW_LENGTH, 158, 207
PACK_SKIP_IMAGES, 158, 184, 207
PACK_SKIP_PIXELS, 158, 207
PACK_SKIP_ROWS, 158, 207
PACK_SWAP_BYTES, 158, 207
PASS_THROUGH_TOKEN, 176
PassThrough, 174
PERSPECTIVE_CORRECTION_HINT, 180
PIXEL_MAP_A_TO_A, 79, 101
PIXEL_MAP_B_TO_B, 79, 101
PIXEL_MAP_G_TO_G, 79, 101
PIXEL_MAP_I_TO_A, 79, 102
PIXEL_MAP_I_TO_B, 79, 102
PIXEL_MAP_I_TO_G, 79, 102
PIXEL_MAP_I_TO_I, 79, 102
PIXEL_MAP_I_TO_R, 79, 102
PIXEL_MAP_R_TO_R, 79, 101
PIXEL_MAP_S_TO_S, 79, 102
PIXEL_MODE_BIT, 191
PixelMap, 75, 78, 79, 162
PixelStore, 19, 75, 76, 78, 158, 162, 178
PixelTransfer, 75, 78, 107, 162
PixelZoom, 100
POINT, 73, 74, 168, 169, 201, 226
POINT_BIT, 191
POINT_SMOOTH, 60
POINT_SMOOTH_HINT, 180
POINT_TOKEN, 176
POINTS, 15, 168

Microsoft Corp. Exhibit 1009

PointSize, 60
POLYGON, 16, 19
POLYGON_BIT, 191
POLYGON_OFFSET_FILL, 74
POLYGON_OFFSET_LINE, 74
POLYGON_OFFSET_POINT, 74
POLYGON_SMOOTH, 70
POLYGON_SMOOTH_HINT, 180
POLYGON_STIPPLE, 72
POLYGON_STIPPLE_BIT, 191
POLYGON_TOKEN, 176
PolygonMode, 69, 73, 75, 171, 173
PolygonOffset, 74
PolygonStipple, 72
PopAttrib, 189, 190, 192, 224, 251
PopClientAttrib, 19, 178, 189, 190, 192, 251
PopMatrix, 34, 245
PopName, 171
POSITION, 50, 183
POST_COLOR_MATRIX_*x*_BIAS, 78
POST_COLOR_MATRIX_*x*_SCALE, 78
POST_COLOR_MATRIX_ALPHA_BIAS, 108
POST_COLOR_MATRIX_ALPHA_SCALE, 108
POST_COLOR_MATRIX_BLUE_BIAS, 108
POST_COLOR_MATRIX_BLUE_SCALE, 108
POST_COLOR_MATRIX_COLOR_TABLE, 80, 109
POST_COLOR_MATRIX_GREEN_BIAS, 108
POST_COLOR_MATRIX_GREEN_SCALE, 108
POST_COLOR_MATRIX_RED_BIAS, 108
POST_COLOR_MATRIX_RED_SCALE, 108
POST_CONVOLUTION_*x*_BIAS, 78
POST_CONVOLUTION_*x*_SCALE, 78
POST_CONVOLUTION_ALPHA_BIAS, 107
POST_CONVOLUTION_ALPHA_SCALE, 107
POST_CONVOLUTION_BLUE_BIAS, 107
POST_CONVOLUTION_BLUE_SCALE, 107
POST_CONVOLUTION_COLOR_TABLE, 80, 108
POST_CONVOLUTION_GREEN_BIAS, 107
POST_CONVOLUTION_GREEN_SCALE, 107
POST_CONVOLUTION_RED_BIAS, 107
POST_CONVOLUTION_RED_SCALE, 107
PrioritizeTextures, 134, 135
PROJECTION, 31, 34, 245
PROXY_COLOR_TABLE, 80, 82, 179
PROXY_HISTOGRAM, 87, 88, 179, 188
PROXY_POST_COLOR_MATRIX_COLOR_TABLE, 80, 179
PROXY_POST_CONVOLUTION_COLOR_TABLE, 80, 179
PROXY_TEXTURE_1D, 117, 132, 179, 183
PROXY_TEXTURE_2D, 116, 132, 179, 183
PROXY_TEXTURE_3D, 112, 132, 179, 183
PushAttrib, 189, 190, 192, 251
PushClientAttrib, 19, 178, 189, 190, 192, 251
PushMatrix, 34, 245
PushName, 171

Q, 36, 38, 183
QUAD_STRIP, 17
QUADRATIC_ATTENUATION, 50
QUADS, 18, 19

R, 36, 38, 183

Microsoft Corp. Exhibit 1009

R3_G3_B2, 115
RasterPos, 41, 171, 223, 246
RasterPos2, 41, 246
RasterPos3, 41, 246
RasterPos4, 41, 246
ReadBuffer, 158, 159, 162
ReadPixels, 75, 78, 91–93, 103, 156–160, 162, 178, 184–186
Rect, 28, 70
RED, 78, 92, 159, 160, 208, 210, 216
RED_BIAS, 101
RED_SCALE, 101
REDUCE, 105, 107, 209
RENDER, 171, 172, 217
RENDERER, 189
RenderMode, 171–174, 178
REPEAT, 124, 125, 127, 128, 131, 203
REPLACE, 135, 136, 144
REPLICATE_BORDER, 105, 106
RESCALE_NORMAL, 35
ResetHistogram, 187
ResetMinmax, 188
RETURN, 155, 156
RGB, 92, 94, 98, 103, 104, 113–115, 136, 137, 159, 162, 185, 226
RGB10, 115
RGB10_A2, 115
RGB12, 115
RGB16, 115
RGB4, 115
RGB5, 115
RGB5_A1, 115
RGB8, 115
RGBA, 81, 82, 85–88, 92, 94, 98, 103, 104, 113–115, 136, 137, 159, 162, 185, 208–211
RGBA12, 115
RGBA16, 115
RGBA2, 115
RGBA4, 115
RGBA8, 115
RIGHT, 151, 152, 158
Rotate, 32, 223
S, 36, 37, 183
Scale, 32, 33, 223
Scissor, 143
SCISSOR_BIT, 191
SCISSOR_TEST, 143
SELECT, 171, 172, 224
SelectBuffer, 171, 172, 178
SELECTION_BUFFER_POINTER, 189
SEPARABLE_2D, 85, 103, 117, 187
SeparableFilter2D, 84
SEPARATE_SPECULAR_COLOR, 47
SET, 151
SGI_color_matrix, 233
SGIS_multitexture, 238
SGIS_texture_edge_clamp, 231
SGIS_texture_lod, 232
ShadeModel, 54
SHININESS, 50
SHORT, 22, 91, 160, 161, 177
SINGLE_COLOR, 46, 47, 199
SMOOTH, 54, 198
SPECULAR, 50, 51
SPHERE_MAP, 37, 38
SPOT_CUTOFF, 50
SPOT_DIRECTION, 50, 183
SPOT_EXPONENT, 50
SRC_ALPHA, 148
SRC_ALPHA_SATURATE, 148
SRC_COLOR, 148
STACK_OVERFLOW, 13, 34, 171, 190, 245
STACK_UNDERFLOW, 13, 34, 171, 190, 245
STENCIL, 162
STENCIL_BUFFER_BIT, 154, 191
STENCIL_INDEX, 80, 83, 90, 92, 100, 112, 156, 158, 159, 162, 184
STENCIL_TEST, 144
StencilFunc, 144, 222
StencilMask, 153, 156, 223
StencilOp, 144, 145

Microsoft Corp. Exhibit 1009

T, 36, 183
T2F_C3F_V3F, 25, 26
T2F_C4F_N3F_V3F, 25, 26
T2F_C4UB_V3F, 25, 26
T2F_N3F_V3F, 25, 26
T2F_V3F, 25, 26
T4F_C4F_N3F_V4F, 25, 26
T4F_V4F, 25, 26
TABLE_TOO_LARGE, 13, 80, 87
TexCoord, 19, 20, 241, 243
TexCoord1, 20, 241
TexCoord2, 20, 241
TexCoord3, 20, 241
TexCoord4, 20, 241
TexCoordPointer, 19, 21, 22, 27, 178, 243
TexEnv, 135, 249
TexGen, 36–38, 240, 246
TexImage, 121
TexImage1D, 76, 103, 105, 113, 117, 118, 120, 121, 129, 132, 179, 248
TexImage2D, 76, 87, 88, 103, 105, 113, 116–118, 120, 121, 129, 132, 179, 248
TexImage3D, 76, 112–114, 116–118, 121, 129, 132, 178, 184, 248
TexParameter, 123
TexParameter[if], 126, 130
TexParameterf, 134
TexParameterfv, 134
TexParameteri, 134
TexParameteriv, 134
TexSubImage, 121
TexSubImage1D, 103, 121, 123
TexSubImage2D, 103, 120–122
TexSubImage3D, 120–122
TEXTURE, 31, 34, 244, 245
TEXTURE*i*_ARB, 241
TEXTURE0_ARB, 243, 245, 249, 251, 254
TEXTURE1_ARB, 251
TEXTURE_*x*D, 202, 253
TEXTURE_1D, 103, 117, 120, 121, 124, 132, 133, 138, 183, 184
TEXTURE_2D, 103, 116, 120, 121, 124, 132, 133, 138, 183, 184
TEXTURE_3D, 112, 121, 124, 132, 133, 138, 183, 184
TEXTURE_ALPHA_SIZE, 183
TEXTURE_BASE_LEVEL, 116, 117, 124, 126, 127, 129–132, 248
TEXTURE_BIT, 190, 191
TEXTURE_BLUE_SIZE, 183
TEXTURE_BORDER, 183
TEXTURE_BORDER_COLOR, 124, 129, 131, 132
TEXTURE_COMPONENTS, 183
TEXTURE_COORD_ARRAY, 23, 27, 243
TEXTURE_COORD_ARRAY_POINTER, 189
TEXTURE_DEPTH, 183
TEXTURE_ENV, 135, 183
TEXTURE_ENV_COLOR, 135
TEXTURE_ENV_MODE, 135
TEXTURE_GEN_MODE, 37, 38
TEXTURE_GEN_Q, 38
TEXTURE_GEN_R, 38
TEXTURE_GEN_S, 38
TEXTURE_GEN_T, 38
TEXTURE_GREEN_SIZE, 183
TEXTURE_HEIGHT, 183
TEXTURE_INTENSITY_SIZE, 183
TEXTURE_INTERNAL_FORMAT, 183
TEXTURE_LUMINANCE_SIZE, 183
TEXTURE_MAG_FILTER, 124, 131
TEXTURE_MAX_LEVEL, 116, 124, 130, 132, 248
TEXTURE_MAX_LOD, 124–126, 132
TEXTURE_MIN_FILTER, 124, 127, 129–131, 248
TEXTURE_MIN_LOD, 124–126, 132
TEXTURE_PRIORITY, 124, 132, 134
TEXTURE_RED_SIZE, 183
TEXTURE_RESIDENT, 132, 134

Microsoft Corp. Exhibit 1009

TEXTURE_WIDTH, 183
TEXTURE_WRAP_R, 124, 128
TEXTURE_WRAP_S, 124, 127, 128
TEXTURE_WRAP_T, 124, 128
TRANSFORM_BIT, 191
Translate, 32, 223
TRIANGLE_FAN, 17
TRIANGLE_STRIP, 16
TRIANGLES, 17, 19
TRUE, 18, 19, 40, 46–48, 76, 78, 87, 88, 134, 153, 158, 178, 182, 184, 187, 188

UNPACK_ALIGNMENT, 76, 93, 112, 207
UNPACK_IMAGE_HEIGHT, 76, 112, 207
UNPACK_LSB_FIRST, 76, 98, 207
UNPACK_ROW_LENGTH, 76, 90, 93, 112, 207
UNPACK_SKIP_IMAGES, 76, 112, 117, 207
UNPACK_SKIP_PIXELS, 76, 93, 98, 207
UNPACK_SKIP_ROWS, 76, 93, 98, 207
UNPACK_SWAP_BYTES, 76, 90, 92, 207
UNSIGNED_BYTE, 22, 24, 26, 91, 95, 160, 161, 177
UNSIGNED_BYTE_2_3_3_REV, 91, 93–95, 161
UNSIGNED_BYTE_3_3_2, 91, 93–95, 161
UNSIGNED_INT, 22, 24, 91, 97, 160, 161, 177
UNSIGNED_INT_10_10_10_2, 91, 94, 97, 161
UNSIGNED_INT_2_10_10_10_REV, 91, 94, 97, 161
UNSIGNED_INT_8_8_8_8, 91, 94, 97, 161
UNSIGNED_INT_8_8_8_8_REV, 91, 94, 97, 161
UNSIGNED_SHORT, 22, 24, 91, 96, 160, 161, 177
UNSIGNED_SHORT_1_5_5_5_REV, 91, 94, 96, 161
UNSIGNED_SHORT_4_4_4_4, 91, 94, 96, 161
UNSIGNED_SHORT_4_4_4_4_REV, 91, 94, 96, 161
UNSIGNED_SHORT_5_5_5_1, 91, 94, 96, 161
UNSIGNED_SHORT_5_6_5, 91, 93, 94, 96, 161
UNSIGNED_SHORT_5_6_5_REV, 91, 93, 94, 96, 161

V2F, 25, 26
V3F, 25, 26
VENDOR, 189
VERSION, 189
Vertex, 7, 19, 20, 41, 167
Vertex2, 20, 28
Vertex2sv, 7
Vertex3, 20
Vertex3f, 7
Vertex4, 20
VERTEX_ARRAY, 23, 27
VERTEX_ARRAY_POINTER, 189
VertexPointer, 19, 22, 27, 178
Viewport, 30
VIEWPORT_BIT, 191

XOR, 151

ZERO, 144, 148, 149, 205

Microsoft Corp. Exhibit 1009

Appendix U - Claim Chart Showing Teachings of Potmesil, Hornbacker, and Lindstrom Pertinent to Challenged Claims of U.S. Patent No. 8,924,506

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
1.Preamble: A method of retrieving large-scale images over network communications channels for display on a limited communication bandwidth computer device, said method comprising:	Potmesil, Abstract: We describe **a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth**. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. **We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets**. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1328: **The geographical system outlined in this paper is based on these**

[1] For easier readability, color figures from Potmesil are copied from an online copy of the reference available at http://www.ra.ethz.ch/cdstore/www6/technical/paper130/paper130.html. The figures are identical to those in Ex. [XX].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	assumptions: • the amount of available geographical data by far exceeds the storage capacity of any one client machine: the system needs to be network based with data stored in server hosts (An extreme example is the USGS's 1-meter resolution monochrome image of the United States - when completed it will be available on 3,300 CD-ROM's!), • the system is model based: servers provide clients with models of spatial and other data and all image rendering is done locally by client hosts, • there is a large variety of data, relevant to this system, located on many servers in many formats: a directory system is needed to find such data, • some geography-related data (weather satellite and radar images, traffic reports, news, hotel reservations) need to be accessed in (almost) real time: the system must be network-based to obtain such data, • **the system will used in** traditional computers (PC's, workstations, NC's) as well as in many unforeseen or futuristic devices (ITV's, game boxes, exercise bicycles, multi-media kiosks, **cellular phones**, sunglasses, heads-up displays on car windshields), • a user may need to write custom application programs to visualize some particular data while applications developed by others display related data. Hornbacker, Abstract: A computer network server using HTTP (Web) server software combined with foreground view composer software (50), background view composer

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	software (80), view tile cache (60), view tile cache garbage collector (70) and image files (90) **provides image view data to client workstations (20) using graphical web browsers to display the view of an image from the server. Problems with specialized client workstation image view software are eliminated by using the internet and industry standards based graphical web browsers for the client software**. Network and system performance problems that previously existed when accessing large image files from a network file server are eliminated by tiling the image view so that computation and transmission of the view data can be done in an incremental fashion. The vied tiles are cached on the client workstation to further reduce network traffic. View tiles are cached on the server to reduce the amount to view tile computation and to increase responsiveness of the image view server. Hornbacker at 6:20-7:1: **The preferred view tile format is 128 pixel by 128 pixel GIF image files**. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. **The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes.** With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	support for the format becomes common. Hornbacker at 13:28-14:11: **With a low-bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput,** it 83 seconds (250 KB / 3KB/second) to transfer the image file to the workstation application for viewing. With the image view server only the image data to be displayed needs to be transmitted. A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles. **The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB /3 KB/second). This method of image viewing provides better response to the user with much lower demand on the network connection**. A local-area-network typically utilizes a 10 megabit-per-second media so the savings from the efficiency of the image view server does not seem obvious. However, if the 10 megabit-per-second network is shared by 100 users, then the average bandwidth per user is only about 12.5 kilobytes-per-second so the efficiency of the image view server is still a benefit. Hornbacker at 14:26-28: The graphical Web browser is available on all common workstation types as well other devices such as **notebook computers, palm-top computers**, Network Computers and Web television adapters to provide a widely available solution.
1.A: issuing, from a limited communication bandwidth computer	Potmesil at 1328: **The geographical system outlined in this paper is based on these assumptions**:

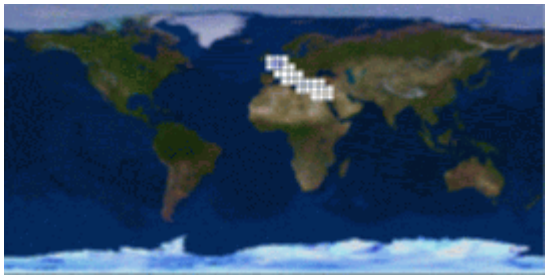
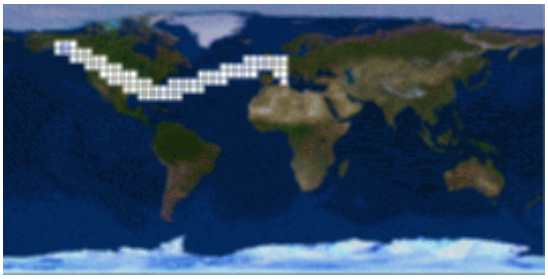
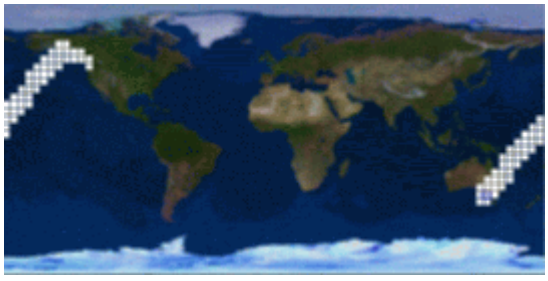
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
device to a remote computer, a request for an update data parcel	• the amount of available geographical data by far exceeds the storage capacity of any one client machine: the system needs to be network based with data stored in server hosts (An extreme example is the USGS's 1-meter resolution monochrome image of the United States - when completed it will be available on 3,300 CD-ROM's!), • the system is model based: servers provide clients with models of spatial and other data and all image rendering is done locally by client hosts, • there is a large variety of data, relevant to this system, located on many servers in many formats: a directory system is needed to find such data, • some geography-related data (weather satellite and radar images, traffic reports, news, hotel reservations) need to be accessed in (almost) real time: the system must be network-based to obtain such data, • **the system will used in** traditional computers (PC's, workstations, NC's) as well as in many unforeseen or futuristic devices (ITV's, game boxes, exercise bicycles, multi-media kiosks, **cellular phones**, sunglasses, heads-up displays on car windshields), • a user may need to write custom application programs to visualize some particular data while applications developed by others display related data. Potmesil, Abstract: We describe **a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth**. Much information available on the WWW, such as weather reports, home pages of National

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. **We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets**. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1327: In this paper, we describe a WWW-based system - consisting of browsers, servers, and connecting protocols - which allows users to view, search and post geographically-indexed information. Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1329-30: 2. Geospatial Servers The concept of a geography server system recognizes that **a digital map or a 3D geographical model is held by many independent sources, distributed over a network. The objective of a browser is to gather all the necessary geographical layers, on as-needed basis, without having to store them locally and to display them**. Our architecture of a geography server system has three major components: a directory scheme for finding servers, a common interface protocol for talking to the servers, and a strategy for implementing the servers themselves. We have developed four different types of servers, so far, in this project: the first three contain actual geographical geometry - (1) points sampled on grids, (2) random points with names, and (3) lines and polygons with names - while the last type stores metadata - information about where to find spatial and geographical information. 2.1 A Tile Server

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. **The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile**. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil at 1332-33:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. **The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them**. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and image pyramids for 2D lattice data such as images, elevations or gradients. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. **The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image**. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3.1 Tile Caching **The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: *x*, *y*, *z*, *level-of-detail* and *time*. In a 2D mode, the *level-of-detail* parameter is used as a discrete *z* level in a 3D pyramid**. Time in this context is interpreted as discrete time slices. **The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there.** This process is shown in Figure 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user, least-recently visible tiles, or least-recently arrived tiles.  (a)  (b)  (c) (d) **Figure 2** Contents of the browser's cache memory after (a) flying from Egypt to Britain,

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	(b) to Alaska, (c) to Australia, and (d) hovering above Australia. **The caching process receives information about the current view from the compositing process**. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. **Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache**. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. **There are several implemented caching strategies:** • **obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)],** • **obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)],** • **obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight.** **When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile**. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	receive and draw the tile data. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image),

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	• mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. 3.3.2 A GIF Applet **This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents.** Optionally, in addition to the GIF transparency value, an alpha-blending value can be specified to make the image background partially visible. **Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and (3) the multi-resolution Mars image server at the Los Alamos National Laboratory.** Hornbacker, 14:26-28: The graphical Web browser is available on all common workstation types as well other devices such as notebook computers, **palm-top computers**, Network Computers and Web television adapters to provide a widely available solution. Hornbacker, Abstract: A computer network server using HTTP (Web) server software combined

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	with foreground view composer software (50), background view composer software (80), view tile cache (60), view tile cache garbage collector (70) and image files (90) provides image view data to client workstations (20) using graphical web browsers to display the view of an image from the server. Problems with specialized client workstation image view software are eliminated by using the internet and industry standards based graphical web browsers for the client software. Network and system performance problems that previously existed when accessing large image files from a network file server are eliminated by **tiling the image view so that computation and transmission of the view data can be done in an incremental fashion**. The vied tiles are cached on the client workstation to further reduce network traffic. View tiles are cached on the server to reduce the amount to view tile computation and to increase responsiveness of the image view server. Hornbacker, 3:10-27: These objects, and others which will become apparent from the following disclosure, are achieved by this invention which comprises in one aspect method of identifying and delivering a graphical image from a computer network file server comprising providing a network file server on which are stored digital document image files, said server adapted to receive **requests from a Web browser in Uniform Resource Locator (URL) code, to identify the image file and format selections being requested, to compose the requested view into a grid of view tiles, and to transmit HTML code for view tiles to the requesting Web browser**. Another aspect of the invention comprises apparatus comprising a computer network server adapted to store digital document image files, programmed to **receive requests from a client Web browser in URL code, the URL specifying a view which identifies an image file and format**, to compose the requested view, and to transmit HTML code for the resultant view to the client Web browser to display.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	A further aspect of the invention is the computer program recorded on magnetic or optical media for use on a network server comprising code which interprets HTTP requests from a workstation for a particular view of a digital document image file stored in memory, retrieves the digital document image file, **composes a grid of view tiles corresponding to the requested view of the image**, computes HTML code for the grid of view tiles in a form which can be transmitted from the server to the workstation. Hornbacker, 5:16-25 In operation according to an embodiment of the method of the invention, using the Web browser software on the client workstation**, a user requests an image view 110 (FIG. 2) having a scale and region specified by by means of a specially formatted Uniformed Resource Locator (URL) code using HTTP language which the Web server can decode as a request to be passed to the image view composition software and that identifies the image file to be viewed**, the scale of the view and the region of the image to view. The network image server sends HTML data to the client with pre-computed hyperlinks, such that following a hyperlink by clicking on an area of an image will send a specific request to the server to deliver a different area of the drawing or to change the resolution of the image. The resultant HTML from this request will also contain pre-computed hyperlinks for other options the user may exercise. Hornbacker, 6:13-19 The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation. The use of tiling enables effective image data caching 60 at the image view server and by the browser

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	10 at the client workstation. Hornbacker, 7:26-8:6 The use of view tiling also **allows the image view server to effectively pre-compute view tiles that may be required by the next view request**. The image view server background view composer computes view tiles that surround the most recent view request in anticipation a request for a shifted view. When the shifted view is requested, the foreground view composer can use the pre-computed view tiles and eliminate the time to compute new view tiles for the view. **For frequently accessed images there is a good chance that the view tiles for a view may already exist in the view tile cache since the view tile cache maintains the most recently accessed view tiles.** Since millions of view tiles may be created and eventually exceed the storage capacity of the image view server, the view tile cache garbage collector removes the least recently accessed view tiles in the case where the maximum storage allocation or minimum storage free space limits are reached. Hornbacker, 8:30-9:28 **To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL**. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form: V < SCALE > < TILE NUMBER > .GIF The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256. The < TILE NUMBER > value is a 5 character

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF **The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view**. A view tile name with these extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:24-28 For example, a specific view request might include tiles B2, C2, B3, and C3 (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 12:24-13:10 **Performance and usability of document viewing can be increased by using progressive display of tiled images**. By using an image file format that allows a rough view of the image to be displayed while the remainder of the image content is downloaded, a rough view of the document can be seen more quickly. Since most Web browsers can only transfer 1 to 4 GIF images at a time, usually not all of the view tiles in the view array can be progressively displayed at the same time. Therefore, it is preferred that to implement progressive display, algorithms at the client are provided to accept an alternate data format that would allow the whole document viewing area screen to take advantage of the progressive display while still taking advantage of the benefits of tiling and caching at the client. This can be accomplished in a Web browser environment using algorithms written in Java, JavaScript, or ActiveX technologies. By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile.
1.B: wherein the update data parcel is selected based on an operator controlled image viewpoint on	Potmesil, Abstract: We describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and

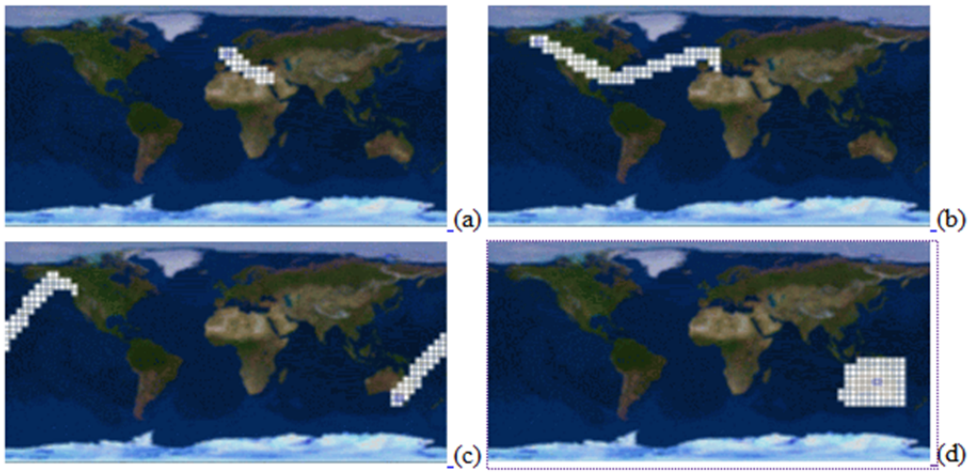
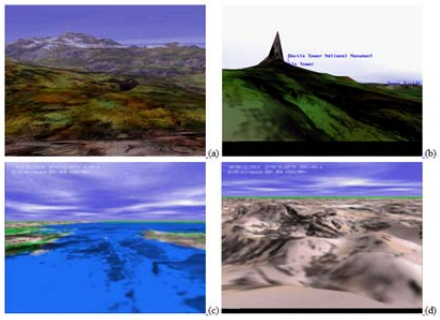
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
the computer device relative to a predetermined image and	visualized when displayed directly or via clickable anchors on top of 2D maps or **in full 3D environments**. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and **a 3D flight-simulator browser capable of continuous flight around the Earth**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. **The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space** as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1327-29: Internet-based computers and communications can be very effective in enhancing our ability to visualize and to search 3D environments in the great outdoors where we move, work, play and learn. In this paper, we describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. In addition, very large data bases of geographical information itself, such as terrain elevation,

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	satellite and aerial images, detailed street maps and geometrical models of buildings and similar man-made structures (present, past and future) are also becoming available. We seek to build an integrated system which will allow its users to browse in such spatial data, make queries and post new data. Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and **a 3D flight-simulator browser**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. **The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space** as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1332: 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. **The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there**. This process is shown in Figure 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user, least-recently visible tiles, or least-recently arrived tiles. **The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache**. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)], • obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)],

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. Potmesil at 1340-41: 3.4 A 3D Geographical Browser We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. The browser uses the OpenGL library to render 3D graphics. To make the three-dimensional browser truly global, we represent the Earth as an ellipsoid or geodetic datum called World Geodetic System 1984 (WGS84) [4]. Potmesil, Fig. 2:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	 **Figure 2** Contents of the browser's cache memory after (a) flying from Egypt to Britain, (b) to Alaska, (c) to Australia, and (d) hovering above Australia. Potmesil, Fig. 8:  Figure 8 Four views from the 3D browser: (a) Tuolumne Meadows, Yosemite National Park, California in fog, (b) Devils Tower, Wyoming with names, (c) east from Atlantic Ocean across Strait of Gibraltar, and (d) southwest from the top of Mount Everest towards India. Lindstrom, §§ 1, 4.2.1, 4.2.6.
1.C: the update data parcel contains data that is used to generate a display on the limited	Potmesil, Abstract: We describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
communication bandwidth computer device;	White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. **We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1328: Two geographical browsers have been developed for this system: **a 2D map browser** capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and **a 3D flight-simulator browser**. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D** space as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to **prepare them for display** before the user reaches it. Around this core concept of tile caching, various

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1332: 3. A Geographical Browser **In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them.** The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. **The browser consists of two processes: caching and compositing**. The former process is responsible for managing the local cache **while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image**. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. Potmesil at 1333-1335: 3.2 Tile Compositing

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The tile compositing process composites tile data from the off-screen cached tiles into the on-screen window image. While compositing tiles, it checks whether all mapplets have drawn their layer(s). If there are layers that have to be drawn before a tile can be shown, the process must wait. This process is also responsible for synchronizing all mapplets, obtaining the user's tracking data from a tracking device and obtaining real time or computing simulated time. This assures that all mapplets are in the same space and time. Directions where and how the browser should move in space can come from one of these sources: • a user can click on an anchor in an HTML document concurrently displayed by an HTML browser, • a user can use a mouse or some other tracking device (hand gestures, force-feedback joystick, GPS receiver), or • a mapplet can take control of the browser and compute directions procedurally (e.g., the great circle) or in any other way, perhaps even including the two above methods. The two processes run independently and asynchronously. The cache manager keeps rearranging the cache memory even while the user has stopped and the image is not regenerated. 3.3 Mapplets: Geographical Applets The core of the geographical browser, which consists of **the display and caching processes**, is programmable with small application programs called mapplets. They are preferably written in a platform-independent and down-loadable code such as Java. The programmability of the browser gives a user the ability to mix-and-match mapplets and to view data in novel ways - not foreseen by the authors of the browser. In this section, we describe some of the mapplets that we have developed. **Mapplets obtain pertinent geographical and other data from Internet**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	servers, convert them, if needed, from external representations, and render them via the browser's graphical and image-processing libraries. These are the basic rules that apply to mapplets: • After the core browser has been started, a user may launch additional mapplets - typically, from a mapplet HTML page. By default, the image mapplet, described in Section 3.3.1, is always started with the core browser. • Mapplets are ordered top to bottom in a stack, a mapplet can draw into one or more top-to-bottom ordered adjacent layers. • Before drawing a layer, a mapplet may have to wait for specified lower layers to be drawn first. • **A mapplet draws static data (which changes infrequently) into the off-screen tiles, and dynamic data (which changes from frame to frame) into the on-screen window**. • If a mapplet needs to redraw one of its layers in a cached tile, it invalidates the contents of the tile. All other running mapplets must then redraw their layers in that tile. This means that the mapplets may have to reload their server's data or must maintain their own independent tile cache. • Before **compositing the cached tiles into the final window image**, the browser may have to wait for specified layers to be drawn. By default it always waits for the image mapplet to draw its layer(s). • When a mapplet **draws directly into the on-screen window**, it likely requires a separate drawing process, in order to maintain the browser's interactive update rate. • A mapplet can register with the core browser to receive events from the user's tracking device. An event can be received by all the

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	registered mapplets or can be passed from top to bottom mapplets until a mapplet acts on it. There are several libraries that the core browser makes available to the user mapplets: • a socket library provides a general client/server network connection functions, • an HTTP library provides an interface for the HTTP/1.0 protocol [3] on both the client and server sides. This library also implements an interface to an HTML browser (Netscape Navigator, Mosaic) running concurrently. Moreover, it provides a uniform interface for filling out URL templates. • a caching library allows HTTP documents to be cached in the local client machine in memory or on disc, • **a graphical library draws geometrical primitives either to the off-screen tiles or directly to the on-screen window, as it clips geometrical primitives to within a tile, it puts any clipped parts on a waiting list and draws them later when the adjacent tiles become available**, • **an image processing library performs some elementary image processing functions; as in the graphical library case, if an image-processing function, such as a filter, needs pixels from adjoining tiles, the library needs to preserve them and provide them to adjacent tiles**, • a geographical library converts the coordinates of geometrical primitives among various geographical coordinates systems; it is based on the USGS cartographic library [5]. An individual mapplet may consist of several processes, usually 1-3, which

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	divide the typical mapplet tasks into 3 stages: (1) obtaining metadata and data from servers, (2) converting obtained data into an internal representation, and (3) drawing the data. If a mapplet also needs to obtain meta information from a server or data from multiple information servers, additional processes may have to be spawn. Much of this design depends on the number of simultaneous requests a mapplet will be making and the size and latency of the returned data. 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) **mapping into an image, (3) conversion to the local display format**. The mapplet may request compressed tiles from the tile server if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - **maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient** - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - **maps elevations into elevation colors** without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - **maps gradients into a**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - **maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image),** • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings**, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed**. 3.3.2 A GIF Applet This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. **The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents.** Optionally, in addition to the GIF transparency value, an alpha-blending value can be specified to make the image background partially visible. Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and (3) the multi-resolution Mars image server at the Los Alamos National Laboratory. Potmesil at 1340: 3.4 A 3D Geographical Browser

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	We have developed a preliminary version **of a three-dimensional browser which displays terrain data cached from the tile server** and geographical names cached from the name server. **The browser uses the OpenGL library to render 3D graphics.** To make the three-dimensional browser truly global, we represent the Earth as an ellipsoid or geodetic datum called World Geodetic System 1984 (WGS84) [4]. Hornbacker, Abstract: A computer network server using HTTP (Web) server software combined with foreground view composer software (50), background view composer software (80), view tile cache (60), view tile cache garbage collector (70) and image files (90) provides image view data to client workstations (20) using graphical web browsers to display the view of an image from the server. Problems with specialized client workstation image view software are eliminated by using the internet and industry standards based graphical web browsers for the client software. Network and system performance problems that previously existed when accessing large image files from a network file server are eliminated by **tiling the image view so that computation and transmission of the view data can be done in an incremental fashion**. The vied tiles are cached on the client workstation to further reduce network traffic. View tiles are cached on the server to reduce the amount to view tile computation and to increase responsiveness of the image view server.
1.D: processing, on the remote computer, source image data to obtain a series K_{1-N} of derivative images of progressively	Potmesil at 1329-30: 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections.** A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
lower image resolution and	the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1**]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. **The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it.**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

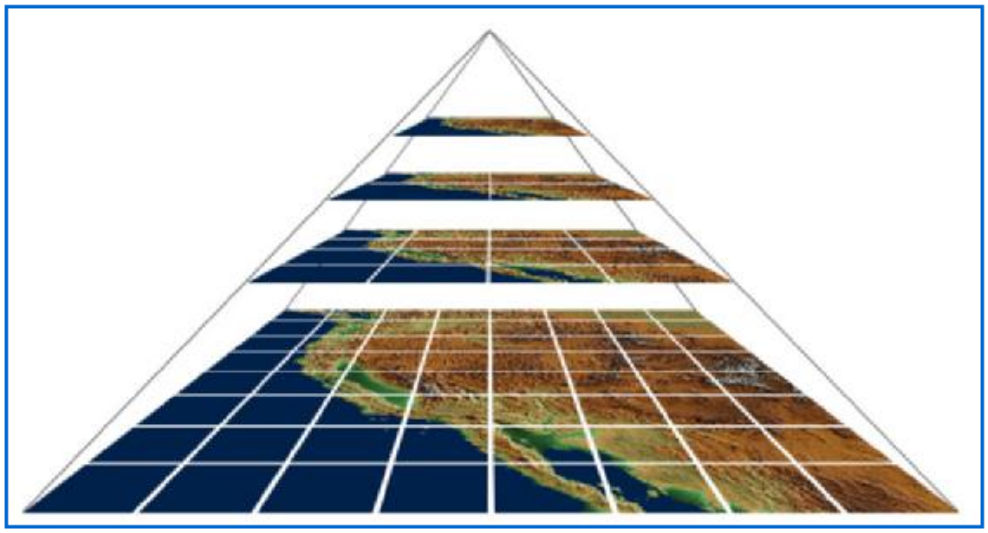


Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

Potmesil, 1332:

3. A Geographical Browser

In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally.

Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and **image pyramids for 2D lattice data such as images, elevations or gradients**. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. **This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid**. Time in this context is interpreted as discrete time slices. Potmesil, 1335: 3.3.2 A GIF Applet This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents. Optionally, in addition to the GIF

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	transparency value, an alpha-blending value can be specified to make the image background partially visible. Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and **(3) the multi-resolution Mars image server at the Los Alamos National Laboratory.** Hornbacker also discloses that view tiles are generated at the server by an image tiling routine that divides a given image into a grid of smaller images, which are further computed for distinct resolutions. The view tiles may either be pre-processed at the server (pre-cached) or newly computed in response to a request: Hornbacker, 3:22-27: A further aspect of the invention is the computer program recorded on magnetic or optical media **for use on a network server** comprising code which interprets HTTP requests from a workstation for a particular view of a digital document image file stored in memory, retrieves the digital document image file, **composes a grid of view tiles corresponding to the requested view of the image**, computes HTML code for the grid of view tiles in a form which can be transmitted from the server to the workstation. Hornbacker, 5:3-8: Referring first to FIG. 1, a network comprising client workstations 10 and 20 are connected through network connections to **a network image view server 100** comprising a network server interface, preferably a web server 30 which uses the Hypertext Transfer Protocol (HTTP), a request broker 40**, a**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	foreground view composer 50, a view tile cache 60, a background view composer 80, a garbage collector 70, and a document repository 90 having image files. Hornbacker, 5:16-6:19: The code is sent over the network to the network server where the web server software interprets the request 120, passes the view request URL to the foreground view composer software through a common gateway interface (CGI) that is designed to allow processing of HTTP requests external to the Web server software, and thereby instructs the request broker 130 to get the particular requested view, having the scale and region called for by the URL**. The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed for the view**, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser. In either case, t**he foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server**, which in turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with n**ewly computed view tiles 220 which are fetched from the server.** **The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation. The use**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	of tiling enables effective image data caching 60 at the image view server and by the browser 10 at the client workstation. Hornbacker further teaches that the image tiles created at the server are processed into a series of images of progressively lower resolutions based on the source data: Hornbacker, 6:20-7:25 T**he preferred view tile format is 128 pixel by 128 pixel GIF image file**s. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. The 128 pixel view tile size is a good compromise between view tile granularity and view tile overhead. The view tile granularity of 128 pixels determines the minimum view shift distance (pan distance) that can be achieved with standard graphical Web browser and level 2 HTML formatting. This allows the adjustment of the view position on a 0.64 inch grid when viewing a 200 pixel-per-inch image at 1 to 1 scale. Reducing the size of the view tiles allows finer grid for view positioning, but has the

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	problem that the view tile overhead becomes excessive. A view tile typically represents more or less than 128 x 128 pixels of the image file. **If the view being displayed is reduced 2 to 1 , then each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client.** For example, consider a view of 512 pixels by 512 pixels. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be computed. In addition only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example. Hornbacker, 8:30-9:28 To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	V < SCALE > < TILE NUMBER > .GIF **The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256.** The < TILE NUMBER > value is a 5 character string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view. A view tile name with these extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:3-10 The foreground view composer 50 interprets the view request command 140 to determine what view needs to be composed. The view request may be

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	absolute by defining scale and position, relative by defining scale and position as a delta to a previous view, or implied by relying on system defaults to select the view. View computation software routine 150 is illustrated in FIG 7 wherein **the command interpreter 151 takes the view request and determines 152 what scale view tile grid is needed for the view** and what view tiles within the grid are needed for the view 150 (FIG. 2), and generates the view tile 153, resulting in formatted view output 154. Hornbacker, 11:19-28 FIG 6 A illustrates how the background view composer algorithm works. Assuming that for a given view requested by the client, tiles C3, C4, D3 and D4 are delivered, after those tile are delivered to the Web browser, the background view composer routine within the server program creates the tiles around these tiles, starting at E4, by composing or computing such surrounding tiles. As long as the client continues to view this page at this scale factor, the server will compute view tiles expanding outward from the tiles requested last. FIG 6B illustrates another request made by a client, after the two rotations of tiles were generated. The request asked for tiles G3, G4, H3, and H4. When the tile pre-computation begins for this request it will create tiles G5, H5, 15, 14, 13, 12, H2, and G2 in the first rotation, but it will not attempt to create tiles in the F column. Hornbacker, 12:21-13:10 Performance and usability of document viewing can be increased by using progressive display of tiled images. By using an image file format that allows a rough view of the image to be displayed while the remainder of the image content is downloaded, a rough view of the document can be seen more quickly.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Since most Web browsers can only transfer 1 to 4 GIF images at a time, usually not all of the view tiles in the view array can be progressively displayed at the same time. Therefore, it is preferred that to implement progressive display, algorithms at the client are provided to accept an alternate data format that would allow the whole document viewing area screen to take advantage of the progressive display while still taking advantage of the benefits of tiling and caching at the client. This can be accomplished in a Web browser environment using algorithms written in Java, JavaScript, or ActiveX technologies. By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile. Hornbacker, 13:26-14:6 For example an E-size engineering drawing raster image file is 8 million bytes in size when imaged in monochrome at 200 pixels-per-inch. With commonly used data compression the image file can be reduced to 250 kilobytes. With a low bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput, it 83 seconds (250 KB / 3 KB/second) to transfer the image file to the workstation application for viewing. With the image view server only the image data to be displayed needs to be transmitted. **A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles**. The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second).

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
1.E: wherein series image K_0 being subdivided into a regular array	Potmesil, Fig. 1: **Figure 1** A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level. Potmesil at 1329-1330: 2.1 A Tile Server The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space**. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. Potmesil at 1332:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and **image pyramids for 2D lattice data such as images, elevations or gradients**. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. **The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions.** In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. **This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid**. Time in this context is interpreted as discrete time slices. Hornbacker teaches that view tiles are generated by an image tiling routine which divides a source image into an array of 128 X 128 pixel tiles at varying resolutions. Hornbacker, 6:13-19 The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation. The use of tiling enables effective image data caching 60 at the image view server and by the browser 10 at the client workstation. Hornbacker, 7:11-15 A view tile typically represents more or less than 128 x 128 pixels of the image file. If the view being displayed is reduced 2 to 1 , then each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 8:30-9:28 To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form: V < SCALE > < TILE NUMBER > .GIF The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256. The < TILE NUMBER > value is a 5 character string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view. A view tile name with these extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:7-10 View computation software routine 150 is illustrated in FIG 7 wherein the command interpreter 151 takes the view request and determines 152 what scale view tile grid is needed for the view and what view tiles within the grid are needed for the view 150 (FIG. 2), and generates the view tile 153, resulting in formatted view output 154.
1.F: wherein each resulting image parcel of the array has a predetermined pixel resolution	Potmesil, Fig. 1: **Figure 1** A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level. Potmesil at 1329-1330:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	2.1 A Tile Server The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. **All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. **Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats**. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. Potmesil at 1332: 3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and **image pyramids for 2D lattice data such as images, elevations or gradients**. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. **The concept of prefiltered power-of-two images for texture mapping was**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, **level-of-detail** and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. Hornbacker, 6:20-7:25 T**he preferred view tile format is 128 pixel by 128 pixel GIF image file**s. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. The 128 pixel view tile size is a good compromise between view tile granularity and view tile overhead. The view tile granularity of 128 pixels determines the minimum view shift distance (pan distance) that can be achieved with standard graphical Web browser and level 2 HTML formatting. This allows the adjustment of the view position on a 0.64 inch grid when viewing a 200 pixel-per-inch image at 1 to 1 scale. Reducing the size of the view tiles allows finer grid for view positioning, but has the problem that the view tile overhead becomes excessive. A view tile typically represents more or less than 128 x 128 pixels of the image file. **If the view being displayed is reduced 2 to 1 , then each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client.** For example, consider a view of 512 pixels by 512 pixels. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be

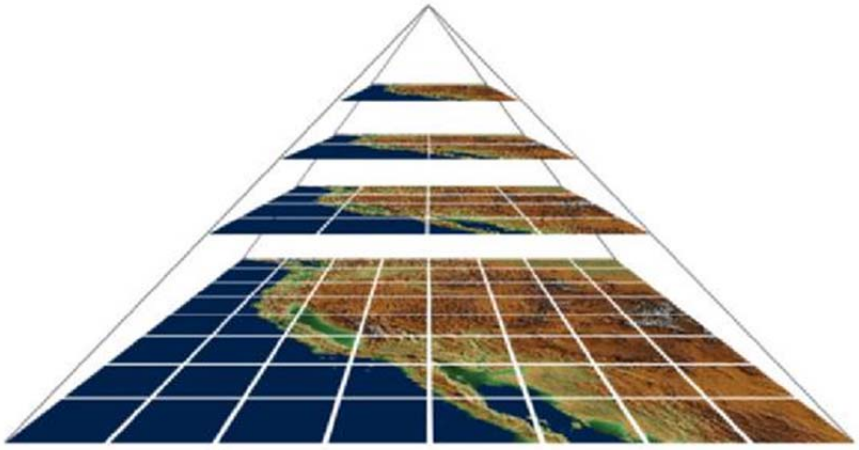
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	computed. In addition only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example. Hornbacker, 8:30-9:28 To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form: V < SCALE > < TILE NUMBER > .GIF **The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256.** The < TILE NUMBER > value is a 5 character string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view. A view tile name with these

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:3-10 The foreground view composer 50 interprets the view request command 140 to determine what view needs to be composed. The view request may be absolute by defining scale and position, relative by defining scale and position as a delta to a previous view, or implied by relying on system defaults to select the view. View computation software routine 150 is illustrated in FIG 7 wherein **the command interpreter 151 takes the view request and determines 152 what scale view tile grid is needed for the view** and what view tiles within the grid are needed for the view 150 (FIG. 2), and generates the view tile 153, resulting in formatted view output 154. Hornbacker, 11:19-28 FIG 6 A illustrates how the background view composer algorithm works. Assuming that for a given view requested by the client, tiles C3, C4, D3 and D4 are delivered, after those tile are delivered to the Web browser, the background view composer routine within the server program creates the tiles around these tiles, starting at E4, by composing or computing such

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	surrounding tiles. As long as the client continues to view this page at this scale factor, the server will compute view tiles expanding outward from the tiles requested last. FIG 6B illustrates another request made by a client, after the two rotations of tiles were generated. The request asked for tiles G3, G4, H3, and H4. When the tile pre-computation begins for this request it will create tiles G5, H5, 15, 14, 13, 12, H2, and G2 in the first rotation, but it will not attempt to create tiles in the F column. Hornbacker, 12:21-13:10 Performance and usability of document viewing can be increased by using progressive display of tiled images. By using an image file format that allows a rough view of the image to be displayed while the remainder of the image content is downloaded, a rough view of the document can be seen more quickly. Since most Web browsers can only transfer 1 to 4 GIF images at a time, usually not all of the view tiles in the view array can be progressively displayed at the same time. Therefore, it is preferred that to implement progressive display, algorithms at the client are provided to accept an alternate data format that would allow the whole document viewing area screen to take advantage of the progressive display while still taking advantage of the benefits of tiling and caching at the client. This can be accomplished in a Web browser environment using algorithms written in Java, JavaScript, or ActiveX technologies. By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile.

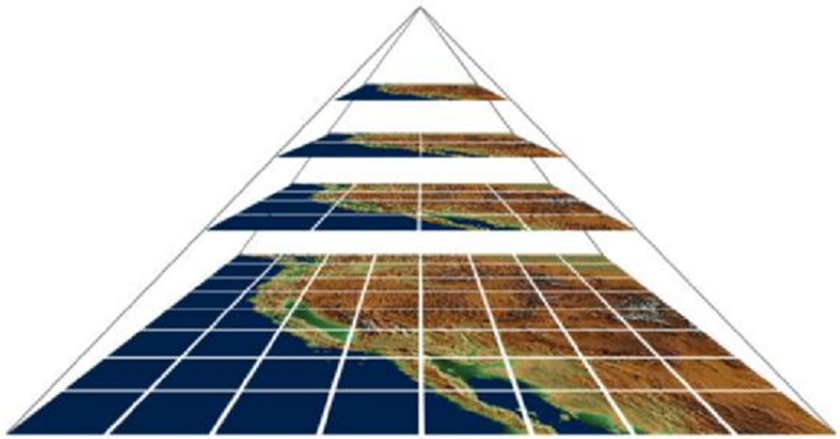
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 13:26-14:6 For example an E-size engineering drawing raster image file is 8 million bytes in size when imaged in monochrome at 200 pixels-per-inch. With commonly used data compression the image file can be reduced to 250 kilobytes. With a low bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput, it 83 seconds (250 KB / 3 KB/second) to transfer the image file to the workstation application for viewing. With the image view server only the image data to be displayed needs to be transmitted. **A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles**. The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second).
1.G: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. The mapplet may request compressed tiles from the tile server if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, **if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed**. Hornbacker at 6:20-7:3: The preferred view tile format is 128 pixel by 128 pixel GIF image files. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. **The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common.
1.H: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Potmesil and Hornbacker both teach storing image data tiles in fixed-size, power-of-two arrays: Potmesil, Fig. 1:  **Figure 1** A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level. Potmesil at 1329-1330: 2.1 A Tile Server The tile server stores data that was obtained by sampling on a 2D grid. This

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. **All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. **Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats**. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. Hornbacker, 6:13-7:25 The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). **These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation.** The use of tiling enables effective image data caching 60 at the image view server and by the browser 10 at the client workstation. The preferred view tile format is 128 pixel by 128 pixel GIF image files. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. The 128 pixel view tile size is a good compromise between view tile granularity and view tile overhead. The view tile granularity of 128 pixels determines the minimum view shift distance (pan distance) that can be achieved with standard graphical Web browser and level 2 HTML formatting. This allows the adjustment of the view position on a 0.64 inch grid when viewing a 200 pixel-per-inch image at 1 to 1 scale. Reducing the size of the view tiles allows finer grid for view positioning, but has the problem that the view tile overhead becomes excessive. A view tile typically represents more or less than 128 x 128 pixels of the image file. If the view being displayed is reduced 2 to 1 , then **each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels**. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client. For example, consider a view of 512 pixels by 512 pixels. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be computed. In addition only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example.
1.I: said array	Potmesil, Fig. 1:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al*., Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
subdivision being related by a factor of two	 **Figure 1** A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level. Potmesil at 1329-1330: 2.1 A Tile Server The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space**. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. Potmesil at 1332: 3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and image pyramids for 2D lattice data such as images, elevations or gradients. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. **The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps.** Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8].
1.J: such that each image parcel being of a fixed byte size,	Potmesil, Fig. 1:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

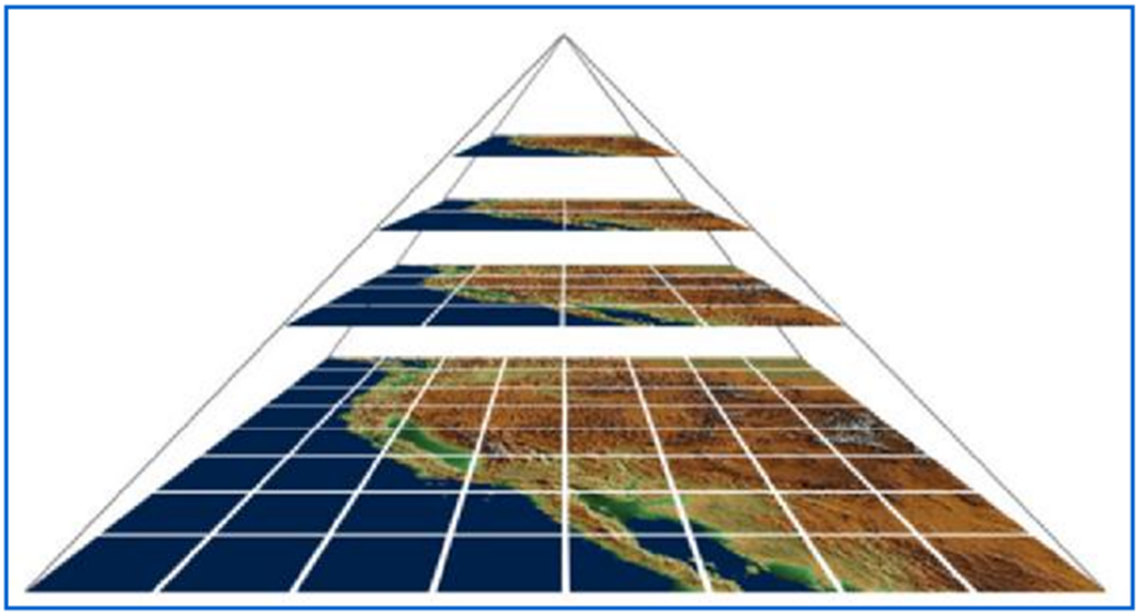


Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

Potmesil, p. 1329-30:

2.1 A Tile Server

The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image**. All tiles in a data set have usually the same size** with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. **A data set may also be stored on one or more compressed formats.** The index of each tile data set is read into the server at

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) **compress the tile if the network bandwidth requires it.** Potmesil, p. 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles. The tiles received from the server are processed in three pipelined steps: (1) **an optional decompression**, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	tile decompression. **The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG.** When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. **When all the tile components are decompressed**, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. Hornbacker at 6:20-7:3:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The preferred view tile format is 128 pixel by 128 pixel **GIF image files. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common**. Hornbacker at 7:14-15: **Fixed size view tiling** is beneficial because it allows more effective use of the caching mechanism at the server and at the client. Hornbacker at 14:2-16: A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles. **The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each** so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second). This method of image viewing provides better response to the user with much

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	lower demand on the network connection. A local-area-network typically utilizes a 10 megabit-per-second media so the savings from the efficiency of the image view server does not seem obvious. However, if the 10 megabit-per-second network is shared by 100 users, then the average bandwidth per user is only about 12.5 kilobytes-per-second so the efficiency of the image view server is still a benefit. **Another benefit of the image view server is that the data transfer size remains constant even if the size of the view image is increased.** If the image file size was 4 times larger than with the previous example as may be the case with a larger image, a higher resolution image or a less compressible image then the network load by the image view server would remain unchanged while network load of the traditional image viewer would quadruple.
1.K: receiving said update data parcel from the data parcel stored in the remote computer over a communications channel; and	Potmesil, Abstract: We describe **a WWW-based system - consisting of browsers, servers and connecting protocols** - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles.** The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. **On the server side, we have developed geographical and geometrical**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies.** The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1329-30: 2. Geospatial Servers The concept of a geography server system recognizes that **a digital map or a**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3D geographical model is held by many independent sources, distributed over a network. The objective of a browser is to gather all the necessary geographical layers, on as-needed basis, without having to store them locally and to display them. Our architecture of a geography server system has three major components: a directory scheme for finding servers, a common interface protocol for talking to the servers, and a strategy for implementing the servers themselves. We have developed four different types of servers, so far, in this project: the first three contain actual geographical geometry - (1) points sampled on grids, (2) random points with names, and (3) lines and polygons with names - while the last type stores metadata - information about where to find spatial and geographical information. 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections.** A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, **a second process is spawned to send tiles to the client while the first process receives tile requests from the client.** Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	in the same data as would happen in a networked game. **The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile**. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil at 1332-33: In this paper, we describe a system for **viewing geospatial models which reside in server hosts across the Internet network**. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. the servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally…. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. *3.1 Tile Caching* The cache process allocates a common tile memory that is shared by all

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Fig. 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user: least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). **The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there**. There are several implemented caching strategies: • Obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail (Fig. 2a-c), • Obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain (Fig.

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	2d), • Obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. *3.2 Tile Compositing* **The tile compositing process composites tile data from the off-screen cached tiles** into the on-screen window image… The two processes run independently and asynchronously. **The cache manager keeps rearranging the cache memory** even while the user has stopped and the image is not regenerated. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. 3.3.2 A GIF Applet **This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents.** Optionally, in addition to the GIF transparency value, an alpha-blending value can be specified to

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	make the image background partially visible. **Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and (3) the multi-resolution Mars image server at the Los Alamos National Laboratory.** Potmesil at 1340-41: *3.4. A 3D geographical browser* We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. Hornbacker teaches that tiles are received over the internet: Hornbacker, 3:10-27: These objects, and others which will become apparent from the following disclosure, are achieved by this invention which comprises in one aspect method of identifying and delivering a graphical image from a computer network file server comprising providing a network file server on which are stored digital document image files, said server adapted to receive **requests from a Web browser in Uniform Resource Locator (URL) code, to identify the image file and format selections being requested, to compose the requested view into a grid of view tiles, and to transmit HTML code for view tiles to the requesting Web browser**. Another aspect of the invention comprises apparatus comprising a computer network server adapted to store digital document image files, programmed to **receive requests from a client Web browser in URL code, the URL specifying a view which identifies an image file and format**, to compose

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	the requested view, and to transmit HTML code for the resultant view to the client Web browser to display. A further aspect of the invention is the computer program recorded on magnetic or optical media for use on a network server comprising code which interprets HTTP requests from a workstation for a particular view of a digital document image file stored in memory, retrieves the digital document image file, **composes a grid of view tiles corresponding to the requested view of the image**, computes HTML code for the grid of view tiles in a form which can be transmitted from the server to the workstation. Hornbacker, 5:3-6:19: Referring first to FIG. 1, **a network comprising client workstations 10 and 20 are connected through network connections to a network image view server 100** comprising a network server interface, preferably a web server 30 which uses the **Hypertext Transfer Protocol (HTTP),** a request broker 40, a foreground view composer 50, a view tile cache 60, a background view composer 80, a garbage collector 70, and a document repository 90 having image files. **The network image view server, i.e. , client workstation, or "workstation," 100 can be implemented on a computer, for example a personal computer configured with a processor, I/O, memory, disk storage, and a network interface. The network image view server 100 is configured with a network server operating system and Web server software 30 to provide the network HTTP protocol link with the client workstations 10 and 20.** Typical networks include many workstations served by one, and sometimes more than one, network server, **the server functioning as a library to maintain files which can be accessed by the workstations.** In operation according to an embodiment of the method of the invention, using the Web browser software on the client workstation, a user requests an

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	image view 110 (FIG. 2) having a scale and region specified by by means of a specially formatted Uniformed Resource Locator (URL) code using HTTP language which the Web server can decode as a request to be passed to the image view composition software and that identifies the image file to be viewed, the scale of the view and the region of the image to view. The network image server sends HTML data to the client with pre-computed hyperlinks, such that following a hyperlink by clicking on an area of an image will send a specific request to the server **to deliver a different area of the drawing or to change the resolution of the image**. The resultant HTML from this request will also contain pre-computed hyperlinks for other options the user may exercise. The code is sent over the network to the network server where the web server software interprets the request 120, passes the view request URL to the foreground view composer software through a common gateway interface (CGI) that is designed to allow processing of HTTP requests external to the Web server software, and thereby instructs the request broker 130 to get the particular requested view, having the scale and region called for by the URL. The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed for the view, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser. In either case, the foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server, which in **turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with newly computed view tiles 220 which are fetched from the server.**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 8:1-6 **For frequently accessed images there is a good chance that the view tiles for a view may already exist in the view tile cache since the view tile cache maintains the most recently accessed view tiles**. Since millions of view tiles may be created and eventually exceed the storage capacity of the image view server, the view tile cache garbage collector removes the least recently accessed view tiles in the case where the maximum storage allocation or minimum storage free space limits are reached. Hornbacker, 8:16-23 The HTML output file produced by the foreground view composer is passed to the Web server software to be transmitted to the Web browser. The graphical Web browser serves as the image viewer by utilizing the HTML output from the image view server to compose and display the array of view tiles that form a view of an image. The HTML page data list the size, position and the hyperlink for each view tile to be displayed. The view tiles are stored in the GIF image file format that can be displayed by all common graphical Web browsers. **The Web browser will retrieve each view tile to be displayed from a local cache if the view tile is present,** otherwise from the image view server. Hornbacker, 10:13-28 The view tile generator routine 160 performs the actual creation of the view tiles according to the preferred steps shown in FIG 8. The view tile generator receives information from the view computation as to what view tiles are needed for the view**. It has access to records in the cache 80 that determine which tiles have already been created and are resident in the cache**. If a needed view tile is in the cache then its last access time is updated

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	to prevent the cache garbage collector from deleting the view tile. If a needed view tile is not in the cache, then the view tile generator creates the view tile from the image file 90. The view tile generator uses a software imaging library that supports rendering many digital document file formats including monochrome raster images, grayscale raster images, color raster images as well as many content rich non-raster formats such as Adobe Portable Document Format (PDF), PostScript, HPGL, etc. When rendering monochrome image data the imaging library scale-to-gray scaling is used to provide a more visually appealing rendition of the reduced image. For example, a specific view request might include tiles B2, C2, B3, and C3 (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent. Hornbacker, 11:29-12:9 Preferably a view tile cache garbage collector algorithm 70 manages the use of the storage for view tiles (FIGS 10A, 10B, 10C). The garbage collector maintains the view tile cache 60 (FIG. 1) to keep the view tile cache storage usage below a storage limit and to keep the storage free space above a free space limit. The garbage collector constantly scans the cache to accumulate size and age statistics for the view tiles. When the cache size needs to be reduced, the garbage collector selects the least recently accessed view tiles for deletion until the cache size is within limits. The garbage collector runs at a low priority to minimize interference with more critical system tasks. The storage free space limit is designed as a failsafe limit to prevent the system from running out of storage. The free space limit is checked on a periodic basis and if it is exceeded the garbage collector becomes a critical system task and runs at a high priority until the storage free space is greater than the free space limit.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 12:17-23 The graphical Web browser on the client workstation 10 receives HTML data from the image view server 210 that contains **hyperlinks to the view tiles within the view tile cache 60 to be displayed and formatting that describes the layout of the of the tiles to form the image view**. The Web browser initially must fetch each view tile 220 for a view from the view server. **After the initial view, whenever a view overlaps with a previous view at the same scale, the Web browser preferably retrieves view tiles that have been previously displayed from the Web browser's local cache 210 rather than from the server**. Hornbacker, 13:19-23 **By using image tiling and caching according to the preferred method, relatively small amounts of data need to be transmitted when the user selects a new view of an image already received and viewed**. The server sends the requested image in the request format to the workstation and then allows viewing the image from the local copy of the image file.
1.L: displaying on the limited communication bandwidth computer device using the update data parcel that is a part of said predetermined image, an image wherein said update data	Potmesil, Abstract: We describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth. **Both**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
parcel uniquely forms a discrete portion of said predetermined image.	**browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles.** The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1327-28: Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. **In addition, very large data bases of geographical information itself, such as terrain elevation, satellite and aerial images, detailed street maps and geometrical models of buildings and similar man-made structures (present, past and future) are also becoming available**. We seek to build an integrated system which will allow its users to browse in such spatial data, make queries and post new data. Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles.** The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. The geographical system outlined in this paper is based on these assumptions: • **the amount of available geographical data by far exceeds the storage capacity of any one client machine: the system needs to be network based with data stored in server hosts (An extreme example is the USGS's 1-meter resolution monochrome image of the United States - when completed it will be available on 3,300 CD-ROM's!)**, • the system is model based: servers provide clients with models of spatial and other data and all image rendering is done locally by client hosts, • there is a large variety of data, relevant to this system, located on many servers in many formats: a directory system is needed to find such data, • some geography-related data (weather satellite and radar images, traffic reports, news, hotel reservations) need to be accessed in (almost) real time: the system must be network-based to obtain such data, • the system will used in traditional computers (PC's, workstations, NC's) as well as in many unforeseen or futuristic devices (ITV's, game boxes, exercise bicycles, multi-media kiosks, cellular phones,

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")

sunglasses, heads-up displays on car windshields),

- a user may need to write custom application programs to visualize some particular data while applications developed by others display related data.

Potmesil, Fig. 1:

Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

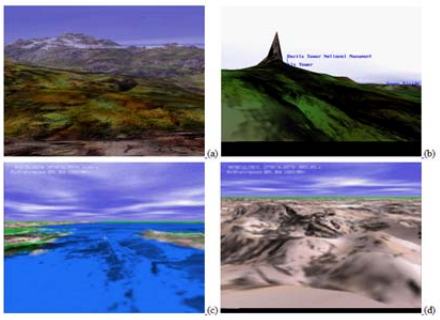
Potmesil at 1329-1330:

2.1 A Tile Server

The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. The mapplet may request compressed tiles from the tile server if the speed of the network connection justifies the additional time spent by the mapplet in tile

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	decompression. The elevation data are usually compressed using a wavelet compression [7], **while the gradient and image data are usually compressed using JPEG**. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • **rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image),** • **mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image).** Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. Potmesil at 1340-41: 3.4 A 3D Geographical Browser We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al*., Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	names cached from the name server. The browser uses the OpenGL library to render 3D graphics. To make the three-dimensional browser truly global, we represent the Earth as an ellipsoid or geodetic datum called World Geodetic System 1984 (WGS84) [4]. Potmesil, Fig. 8:  **Figure 8** Four views from the 3D browser: (a) Tuolumne Meadows, Yosemite National Park, California in fog, (b) Devils Tower, Wyoming with names, (c) east from Atlantic Ocean across Strait of Gibraltar, and (d) southwest from the top of Mount Everest towards India. Hornbacker, 6:7-19 In either case, the foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server, which in turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where **the requesting browser displays any view tiles already cached 210, combined with newly computed view tiles 220 which are fetched from the server.** **The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context).** These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation. The use of tiling enables effective image data caching 60 at the image view server and by the browser 10 at the client workstation.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[1] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Hornbacker, 7:11-25 A view tile typically represents more or less than 128 x 128 pixels of the image file. If the view being displayed is reduced 2 to 1 , then **each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view.** Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client. For example, consider a view of 512 pixels by 512 pixels. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be computed. In addition only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example. Hornbacker, 8:7-15 The number of view tiles needed to render a given view size increases in inverse proportion to the square of the view tile size. A 64 pixel view tile would require 4 times as many view tiles to render the same view area, and so is less preferred. The view tile overhead exists as quantity of data and as the number of network transactions. The data quantity overhead comes from the image file header size as a proportion of the total image file size as described above and as data needed to make the view tile references in the HTML text file. The network transaction overhead increases with smaller view tiles since each of the view tiles requires a network transaction. The increased number of network transactions required with a smaller view tile size would slow the response to render a view.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
2: The method of claim 1, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	Potmesil at 1329-30, 1332, 1335. Hornbacker at Fig. 1, 7:26-8:6, 8:30-9:4, 10:13-16, 11:9-12:23.
3: The method of claim 2, wherein receiving the update data parcel over a communications channel further	Potmesil at 1332-33, Fig. 2.

[2] For easier readability, color figures from Potmesil are copied from an online copy of the reference available at http://www.ra.ethz.ch/cdstore/www6/technical/paper130/paper130.html. The figures are identical to those in Ex. [XX].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	
4: The method of claim 1, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a	Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. The geographical system outlined in this paper is based on these

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
personal digital assistants and an internet-capable digital phone and a television.	assumptions: • **the amount of available geographical data by far exceeds the storage capacity of any one client machine**: the system needs to be network based with data stored in server hosts (An extreme example is the USGS's 1-meter resolution monochrome image of the United States - when completed it will be available on 3,300 CD-ROM's!), • the system is model based: servers provide clients with models of spatial and other data and all image rendering is done locally by client hosts, • there is a large variety of data, relevant to this system, located on many servers in many formats: a directory system is needed to find such data, • some geography-related data (weather satellite and radar images, **traffic reports**, news, hotel reservations) need to be accessed in (almost) real time: the system must be network-based to obtain such data, • the system will used in traditional computers (PC's, workstations, **NC's**) as well as in many unforeseen or futuristic devices (**ITV's**, game boxes, exercise bicycles, multi-media kiosks**, cellular phones**, sunglasses, **heads-up displays on car windshields),** a user may need to write custom application programs to visualize some particular data while applications developed by others display related data. Hornbacker, 14:26-28: The graphical Web browser is available on all common workstation types as well other devices such as **notebook computers, palm-top computers**, Network Computers and **Web television adapters** to provide a widely

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	available solution.
5: The method of claim 1, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	Potmesil at 1334-35. Hornbacker at 6:26-28.
6: The method of claim 1, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of	Potmesil at 1330. Hornbacker at 6:17-19, 7:26-8:6, 11:19-12:5. Lindstrom, Fig. 1, § 4.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
the update data parcel as determined by the remote computer.	
7.A: The method of claim 1, wherein the processing further comprises compressing each data parcel and	Potmesil, p. 1329-30: 2.1 A Tile Server The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image**. All tiles in a data set have usually the same size** with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. **A data set may also be stored on one or more compressed formats.** The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	the requested tile. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) **compress the tile if the network bandwidth requires it.** Potmesil, p. 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles. The tiles received from the server are processed in three pipelined steps: (1) **an optional decompression**, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. **The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG.** When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. **When all the tile components are decompressed**, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. Hornbacker at 6:20-7:3: The preferred view tile format is 128 pixel by 128 pixel **GIF image files. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. Hornbacker at 7:14-15: **Fixed size view tiling** is beneficial because it allows more effective use of the caching mechanism at the server and at the client. Hornbacker at 14:2-16: A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles. **The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each** so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second). This method of image viewing provides better response to the user with much lower demand on the network connection. A local-area-network typically utilizes a 10 megabit-per-second media so the savings from the efficiency of the image view server does not seem obvious. However, if the 10 megabit-per-second network is shared by 100 users, then the average bandwidth per user is only about 12.5 kilobytes-per-second so the efficiency of the image view server is still a benefit. **Another benefit of the image view server is that the data transfer size remains constant even if the size of the view image is increased.** If the image file size was 4 times larger than with the previous example as may be the case with a larger image, a higher resolution image or a less compressible image then the network load by the image view server would remain unchanged while network load of the traditional image viewer would quadruple.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
7B. storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Potmesil, Abstract: On the server side, we have developed geographical and geometrical servers which **contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees**. We have also developed a metadata server which contains, **in hierarchical layers**, URL pointers and **geographical coordinates** of various WWW documents, geographical information and geometrical models. Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position**, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies**. The browsers query servers only for relevant data around the user's current and predicted future locations** and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. **Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser**. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1329-30:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	2. Geospatial Servers The concept of a geography server system recognizes that a digital map or a 3D geographical model is held by many independent sources, distributed over a network. The objective of a browser is to gather all the necessary geographical layers, on as-needed basis, without having to store them locally and to display them. Our architecture of a geography server system has three major components: a directory scheme for finding servers, a common interface protocol for talking to the servers, and a strategy for implementing the servers themselves. We have developed four different types of servers, so far, in this project: the first three contain actual geographical geometry **- (1) points sampled on grids**, (2) random points with names, and (3) lines and polygons with names - while the last type stores metadata - information about where to find spatial and geographical information. 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections**. **A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image**. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil, Fig. 1:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

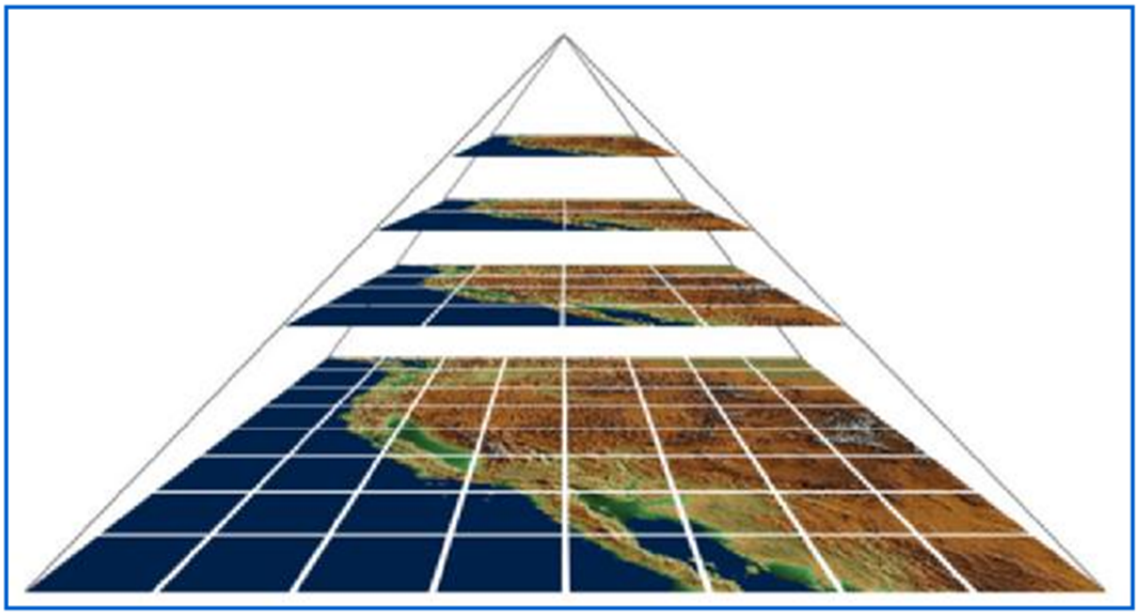


Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

Potmesil at 1332:

3.1 Tile Caching

The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. **This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time.** In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices.

The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the **tile coordinate system**. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)], • obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)], • obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. Potmesil, Fig. 2:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

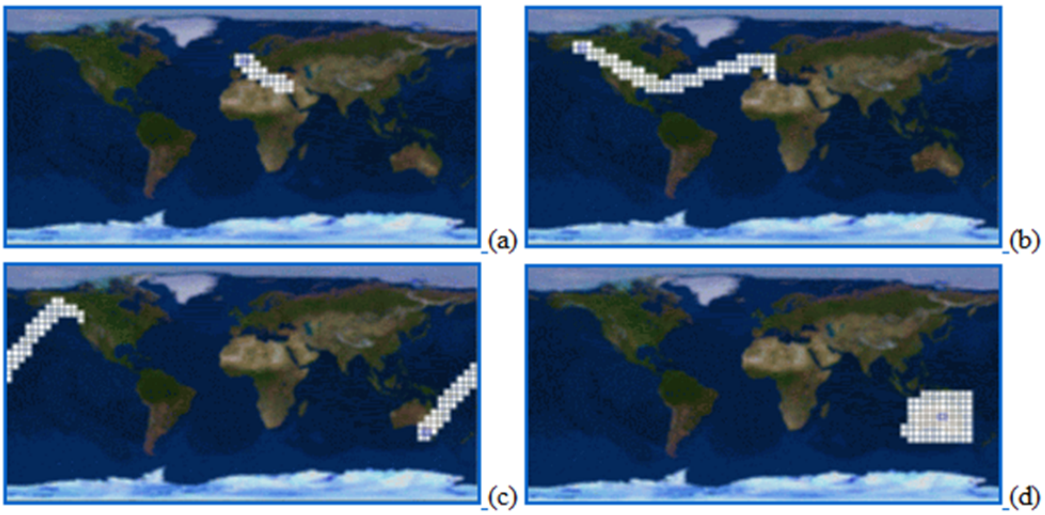


Figure 2 Contents of the browser's cache memory after (a) flying from Egypt to Britain, (b) to Alaska, (c) to Australia, and (d) hovering above Australia.

Hornbacker teaches that tiles may be located and requested by providing requests in URL format specifying the zoom level and x, y coordinates of the tile.

Hornbacker at 3:17-21:

Another aspect of the invention comprises apparatus comprising a computer network server adapted to store digital document image files, **programmed to recieve requests from a client Web browser in URL code, the URL specifying a view which identifies an image file and format, to compose the requested view**, and to transmit HTML code for the resultant view to the client Web browser to display.

Hornbacker at 5:16-24:

In operation according to an embodiment of the method of the invention, using the Web browser software on the client workstation, **a user requests an image view 110 (FIG. 2) having a scale and region specified by by**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	means of a specially formatted Uniformed Resource Locator (URL) code using HTTP language which the Web server can decode as a request to be passed to the image view composition software and that identifies the image file to be viewed, the scale of the view and the region of the image to view. The network image server sends HTML data to the client with pre-computed hyperlinks, such that following a hyperlink by clicking on an area of an image will send a specific request to the server to deliver a different area of the drawing or to change the resolution of the image. Hornbacker at 8:16-23: The HTML output file produced by the foreground view composer is passed to the Web server software to be transmitted to the Web browser. **The graphical Web browser serves as the image viewer by utilizing the HTML output from the image view server to compose and display the array of view tiles that form a view of an image. The HTML page data list the size, position and the hyperlink for each view tile to be displayed. The view tiles are stored in the GIF image file format that can be displayed by all common graphical Web browsers**. The Web browser will retrieve each view tile to be displayed from a local cache if the view tile is present, otherwise from the image view server. Hornbacker at 8:30-10:6: **To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL**. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	V < SCALE > < TILE NUMBER > .GIF The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256. The < TILE NUMBER > value is a 5 character string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF **The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view**. A view tile name with these extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. The foreground view composer 50 interprets the view request command 140 to determine what view needs to be composed. **The view request may be absolute by defining scale and position**, relative by defining scale and position as a delta to a previous view, or implied by relying on system

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	defaults to select the view. Hornbacker at 10:24-28: For example, **a specific view request might include tiles B2, C2, B3, and C3** (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent.
8 Preamble: A display system for displaying a large-scale image retrieved over a limited bandwidth communications channel, said display system comprising:	Potmesil, Abstract: We describe **a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth**. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. **We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets**. On the server side, we have developed geographical and geometrical servers

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1328: **The geographical system outlined in this paper is based on these assumptions**: • the amount of available geographical data by far exceeds the storage capacity of any one client machine: the system needs to be network based with data stored in server hosts (An extreme example is the USGS's 1-meter resolution monochrome image of the United States - when completed it will be available on 3,300 CD-ROM's!), • the system is model based: servers provide clients with models of spatial and other data and all image rendering is done locally by client hosts, • there is a large variety of data, relevant to this system, located on many servers in many formats: a directory system is needed to find such data, • some geography-related data (weather satellite and radar images, traffic reports, news, hotel reservations) need to be accessed in (almost) real time: the system must be network-based to obtain such data, • **the system will used in** traditional computers (PC's, workstations, NC's) as well as in many unforeseen or futuristic devices (ITV's, game boxes, exercise bicycles, multi-media kiosks, **cellular phones**,

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	sunglasses, heads-up displays on car windshields), • a user may need to write custom application programs to visualize some particular data while applications developed by others display related data. Hornbacker, Abstract: A computer network server using HTTP (Web) server software combined with foreground view composer software (50), background view composer software (80), view tile cache (60), view tile cache garbage collector (70) and image files (90) **provides image view data to client workstations (20) using graphical web browsers to display the view of an image from the server. Problems with specialized client workstation image view software are eliminated by using the internet and industry standards based graphical web browsers for the client software**. Network and system performance problems that previously existed when accessing large image files from a network file server are eliminated by tiling the image view so that computation and transmission of the view data can be done in an incremental fashion. The vied tiles are cached on the client workstation to further reduce network traffic. View tiles are cached on the server to reduce the amount to view tile computation and to increase responsiveness of the image view server. Hornbacker at 6:20-7:1: **The preferred view tile format is 128 pixel by 128 pixel GIF image files**. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. **The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. Hornbacker at 13:28-14:11: **With a low-bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput,** it 83 seconds (250 KB / 3KB/second) to transfer the image file to the workstation application for viewing. With the image view server only the image data to be displayed needs to be transmitted. A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles. **The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB /3 KB/second). This method of image viewing provides better response to the user with much lower demand on the network connection**. A local-area-network typically utilizes a 10 megabit-per-second media so the savings from the efficiency of the image view server does not seem obvious. However, if the 10 megabit-per-second network is shared by 100 users, then the average bandwidth per user is only about 12.5 kilobytes-per-second so the efficiency of the image view server is still a benefit. Hornbacker at 14:26-28:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The graphical Web browser is available on all common workstation types as well other devices such as **notebook computers, palm-top computers**, Network Computers and Web television adapters to provide a widely available solution.
8.A: a display of defined screen resolution for displaying a defined image;	Potmesil at 1332-33: 3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and **image pyramids for 2D lattice data such as images, elevations or gradients**. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14**]. The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice**. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Figure 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user, least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. **A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system**. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)], • obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)], • obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. Potmesil, Fig. 2:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

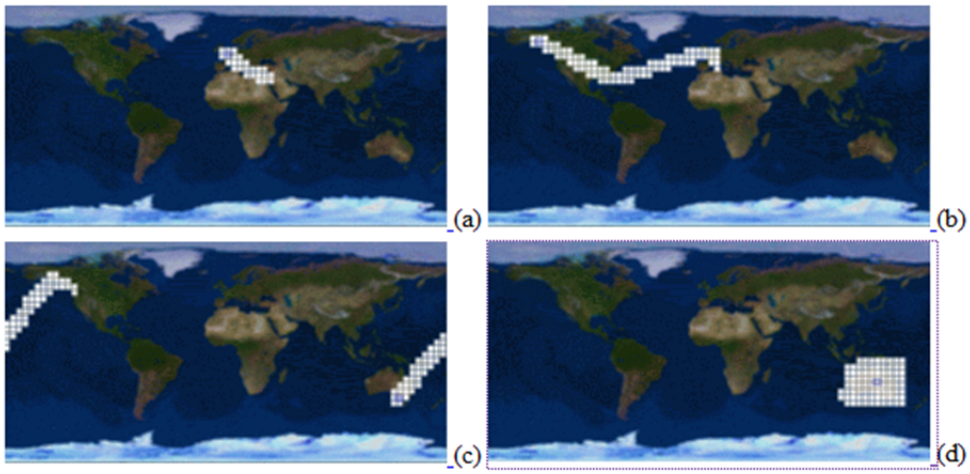


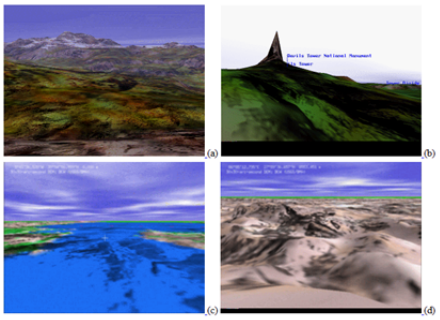
Figure 2 Contents of the browser's cache memory after (a) flying from Egypt to Britain, (b) to Alaska, (c) to Australia, and (d) hovering above Australia.

Potmesil at 1340-41:

3.4 A 3D Geographical Browser

We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. The browser uses the OpenGL library to render 3D graphics. To make the three-dimensional browser truly global, we represent the Earth as an ellipsoid or geodetic datum called World Geodetic System 1984 (WGS84) [4].

Potmesil, Fig. 8:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	 **Figure 8** Four views from the 3D browser: (a) Tuolumne Meadows, Yosemite National Park, California in fog, (b) Devils Tower, Wyoming with names, (c) east from Atlantic Ocean across Strait of Gibraltar, and (d) southwest from the top of Mount Everest towards India. Hornbacker, 7:4-25 The 128 pixel view tile size is a good compromise between view tile granularity and view tile overhead. The view tile granularity of 128 pixels determines the minimum view shift distance (pan distance) that can be achieved with standard graphical Web browser and level 2 HTML formatting. **This allows the adjustment of the view position on a 0.64 inch grid when viewing a 200 pixel-per-inch image at 1 to 1 scale**. Reducing the size of the view tiles allows finer grid for view positioning, but has the problem that the view tile overhead becomes excessive. A view tile typically represents more or less than 128 x 128 pixels of the image file. If the view being displayed is reduced 2 to 1 , then each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client. For example, **consider a view of 512 pixels by 512 pixels**. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be computed. In addition

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example. Hornbacker, 11:19-28 FIG 6 A illustrates how the background view composer algorithm works. Assuming that for a given view requested by the client, tiles C3, C4, D3 and D4 are delivered, after those tile are delivered to the Web browser, the background view composer routine within the server program creates the tiles around these tiles, starting at E4, by composing or computing such surrounding tiles. As long as the client continues to view this page at this scale factor, the server will compute view tiles expanding outward from the tiles requested last. FIG 6B illustrates another request made by a client, after the two rotations of tiles were generated. The request asked for tiles G3, G4, H3, and H4. When the tile pre-computation begins for this request it will create tiles G5, H5, 15, 14, 13, 12, H2, and G2 in the first rotation, but it will not attempt to create tiles in the F column. Hornbacker, 13:4-10 By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile. Hornbacker, 14:2-6 **A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles**. The monochrome view tiles are transmitted

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second).
8.B: a memory providing for the storage of a plurality of image parcels	Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies.** The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. **Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed**. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1332-33: In this paper, we describe a system for viewing geospatial models which reside in serer hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. the servers can generate the models in

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally…. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. *3.1 Tile Caching* **The cache process allocates a common tile memory that is shared by all mapplets**. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Fig. 2. **When the tile cache is full, some resident tiles need to be deleted**. These can be tiles furthermost from the user: least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). **The**

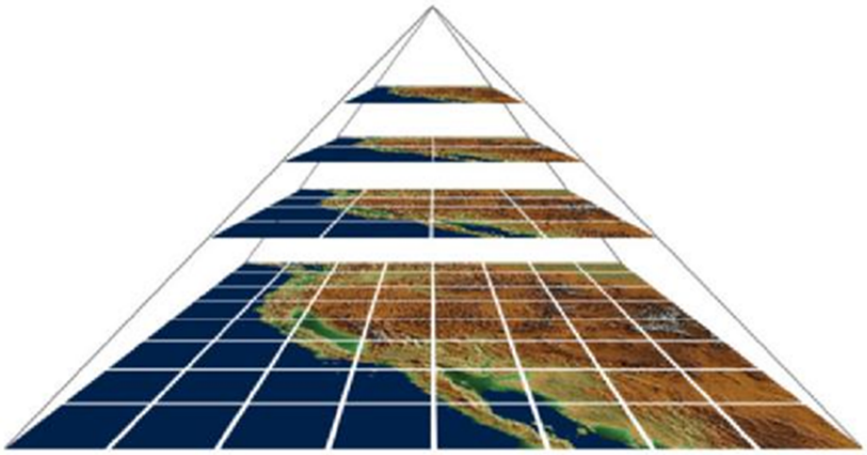
8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • Obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail (Fig. 2a-c), • Obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain (Fig. 2d), • Obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. **When the caching process has generated a new list of tiles to be cached**, each mapplet can start loading its data into each tile. **Mapplets also provide feedback to the cache process**: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. *3.2 Tile Compositing* **The tile compositing process composites tile data from the off-screen cached tiles** into the on-screen window image… The two processes run independently and asynchronously. **The cache manager keeps rearranging the cache memory** even while the user has

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	stopped and the image is not regenerated. Potmesil at 1334: There are several libraries that the core browser makes available to the user mapplets:… • A ***caching* library** allows HTTP documents to be cached on the local client machine in memory or on disc… *3.3.1 An image applet* This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. *3.4. A 3D geographical browser* We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. Hornbacker teaches that retrieved tiles may be stored in a local cache. Hornbacker, 6:1-19 The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed for the view, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, **unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser**. In either case, the

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server, which in turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with newly computed view tiles 220 which are fetched from the server. Hornbacker, 8:1-6 **For frequently accessed images there is a good chance that the view tiles for a view may already exist in the view tile cache since the view tile cache maintains the most recently accessed view tiles**. Since millions of view tiles may be created and eventually exceed the storage capacity of the image view server, the view tile cache garbage collector removes the least recently accessed view tiles in the case where the maximum storage allocation or minimum storage free space limits are reached. Hornbacker, 8:16-23 The HTML output file produced by the foreground view composer is passed to the Web server software to be transmitted to the Web browser. The graphical Web browser serves as the image viewer by utilizing the HTML output from the image view server to compose and display the array of view tiles that form a view of an image. The HTML page data list the size, position and the hyperlink for each view tile to be displayed. The view tiles are stored in the GIF image file format that can be displayed by all common graphical Web browsers. **The Web browser will retrieve each view tile to be displayed from a local cache if the view tile is present,** otherwise from the image view server. Hornbacker, 10:13-28 The view tile generator routine 160 performs the actual creation of the view

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	tiles according to the preferred steps shown in FIG 8. The view tile generator receives information from the view computation as to what view tiles are needed for the view**. It has access to records in the cache 80 that determine which tiles have already been created and are resident in the cache**. If a needed view tile is in the cache then its last access time is updated to prevent the cache garbage collector from deleting the view tile. If a needed view tile is not in the cache, then the view tile generator creates the view tile from the image file 90. The view tile generator uses a software imaging library that supports rendering many digital document file formats including monochrome raster images, grayscale raster images, color raster images as well as many content rich non-raster formats such as Adobe Portable Document Format (PDF), PostScript, HPGL, etc. When rendering monochrome image data the imaging library scale-to-gray scaling is used to provide a more visually appealing rendition of the reduced image. For example, a specific view request might include tiles B2, C2, B3, and C3 (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent. Hornbacker, 11:29-12:9 Preferably a view tile cache garbage collector algorithm 70 manages the use of the storage for view tiles (FIGS 10A, 10B, 10C). The garbage collector maintains the view tile cache 60 (FIG. 1) to keep the view tile cache storage usage below a storage limit and to keep the storage free space above a free space limit. The garbage collector constantly scans the cache to accumulate size and age statistics for the view tiles. When the cache size needs to be reduced, the garbage collector selects the least recently accessed view tiles for deletion until the cache size is within limits. The garbage collector runs at a low priority to minimize interference with more critical system tasks. The storage free space limit is designed as a failsafe limit to prevent the system

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	from running out of storage. The free space limit is checked on a periodic basis and if it is exceeded the garbage collector becomes a critical system task and runs at a high priority until the storage free space is greater than the free space limit. Hornbacker, 12:17-23 The graphical Web browser on the client workstation 10 receives HTML data from the image view server 210 that contains **hyperlinks to the view tiles within the view tile cache 60 to be displayed and formatting that describes the layout of the of the tiles to form the image view**. The Web browser initially must fetch each view tile 220 for a view from the view server. **After the initial view, whenever a view overlaps with a previous view at the same scale, the Web browser preferably retrieves view tiles that have been previously displayed from the Web browser's local cache 210 rather than from the server**. Hornbacker, 13:19-23 **By using image tiling and caching according to the preferred method, relatively small amounts of data need to be transmitted when the user selects a new view of an image already received and viewed**. The server sends the requested image in the request format to the workstation and then allows viewing the image from the local copy of the image file.
8.C: displayable over respective portions of a mesh corresponding to said defined image;	Potmesil, Fig. 1:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	 **Figure 1** A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level. Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles.** The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1329-30: 2. Geospatial Servers The concept of a geography server system recognizes that a digital map or a 3D geographical model is held by many independent sources, distributed over a network. The objective of a browser is to gather all the necessary geographical layers, on as-needed basis, without having to store them locally and to display them. Our architecture of a geography server system has three major components: a directory scheme for finding servers, a common interface protocol for talking to the servers, and a strategy for implementing the servers themselves. We have developed four different types of servers, so far, in this project: the first three contain actual geographical geometry - (1) **points sampled on grids**, (2) random points with names, and (3) lines and polygons with names - while the last type stores metadata - information about where to find spatial and geographical information. 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid**. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. **A data set is stored in a tile index.** A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats**. The index of each tile data set is read into the server**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. The output of the server has several pipelined stages which: (a) **reformat the tile if the requested tile is not aligned with tiles stored in the server**; (b) **resample the tile if the requested tile is not in the same coordinate system;** (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil at 1332-35: 3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	amount of such models far exceeds the ability to store these models locally. **Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and image pyramids for 2D lattice data such as images, elevations or gradients. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps**. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. **In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions**. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. **This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time.** In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Figure 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user, least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the **tile coordinate system**. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)], • obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)], • obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. 3.2 Tile Compositing **The tile compositing process composites tile data from the off-screen cached tiles into the on-screen window image**. While compositing tiles, it checks whether all mapplets have drawn their layer(s). If there are layers that have to be drawn before a tile can be shown, the process must wait. This process is also responsible for synchronizing all mapplets, obtaining the user's tracking data from a tracking device and obtaining real time or computing simulated time. This assures that all mapplets are in the same space and time. Directions where and how the browser should move in space can come from one of these sources: • a user can click on an anchor in an HTML document concurrently displayed by an HTML browser, • a user can use a mouse or some other tracking device (hand gestures, force-feedback joystick, GPS receiver), or • a mapplet can take control of the browser and compute directions procedurally (e.g., the great circle) or in any other way, perhaps even including the two above methods. The two processes run independently and asynchronously. The cache manager keeps rearranging the cache memory even while the user has stopped and the image is not regenerated. 3.3 Mapplets: Geographical Applets

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The core of the geographical browser, which consists of the display and caching processes, is programmable with small application programs called mapplets. They are preferably written in a platform-independent and down-loadable code such as Java. The programmability of the browser gives a user the ability to mix-and-match mapplets and to view data in novel ways - not foreseen by the authors of the browser. In this section, we describe some of the mapplets that we have developed. Mapplets obtain pertinent geographical and other data from Internet servers, convert them, if needed, from external representations, and render them via the browser's graphical and image-processing libraries. These are the basic rules that apply to mapplets: • After the core browser has been started, a user may launch additional mapplets - typically, from a mapplet HTML page. By default, the image mapplet, described in Section 3.3.1, is always started with the core browser. • Mapplets are ordered top to bottom in a stack, a mapplet can draw into one or more top-to-bottom ordered adjacent layers. • Before drawing a layer, a mapplet may have to wait for specified lower layers to be drawn first. • A mapplet draws static data (which changes infrequently) into the off-screen tiles, and dynamic data (which changes from frame to frame) into the on-screen window. • If a mapplet needs to redraw one of its layers in a cached tile, it invalidates the contents of the tile. All other running mapplets must then redraw their layers in that tile. This means that the mapplets may have to reload their server's data or must maintain their own independent tile cache.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	• Before compositing the cached tiles into the final window image, the browser may have to wait for specified layers to be drawn. By default it always waits for the image mapplet to draw its layer(s). • When a mapplet draws directly into the on-screen window, it likely requires a separate drawing process, in order to maintain the browser's interactive update rate. • A mapplet can register with the core browser to receive events from the user's tracking device. An event can be received by all the registered mapplets or can be passed from top to bottom mapplets until a mapplet acts on it. There are several libraries that the core browser makes available to the user mapplets: • a socket library provides a general client/server network connection functions, • an HTTP library provides an interface for the HTTP/1.0 protocol [3] on both the client and server sides. This library also implements an interface to an HTML browser (Netscape Navigator, Mosaic) running concurrently. Moreover, it provides a uniform interface for filling out URL templates. • a caching library allows HTTP documents to be cached in the local client machine in memory or on disc, • a graphical library draws geometrical primitives either to the off-screen tiles or directly to the on-screen window, as it clips geometrical primitives to within a tile, it puts any clipped parts on a waiting list and draws them later when the adjacent tiles become available, • an image processing library performs some elementary image processing functions; as in the graphical library case, if an image-

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	processing function, such as a filter, needs pixels from adjoining tiles, the library needs to preserve them and provide them to adjacent tiles, • a geographical library converts the coordinates of geometrical primitives among various geographical coordinates systems; it is based on the USGS cartographic library [5]. An individual mapplet may consist of several processes, usually 1-3, which divide the typical mapplet tasks into 3 stages**: (1) obtaining metadata and data from servers, (2) converting obtained data into an internal representation, and (3) drawing the data**. If a mapplet also needs to obtain meta information from a server or data from multiple information servers, additional processes may have to be spawn. Much of this design depends on the number of simultaneous requests a mapplet will be making and the size and latency of the returned data. 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2**) mapping into an image**, (3) conversion to the local display format. The mapplet may request compressed tiles from the tile server if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. 3.3.2 A GIF Applet This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. **The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents**. Optionally, in addition to the GIF transparency value, an alpha-blending value can be specified to make the image background partially visible. **Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	TIGER street map server, and (3) the multi-resolution Mars image server at the Los Alamos National Laboratory. Hornbacker at 6:13-19: The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation.
8.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel;	Potmesil and Hornbacker both teach a communications channel interface for receiving data over a limited bandwidth communications channel, as discussed in regard to the preamble of claim 1.
8.E: a processor coupled between said display, memory and communications channel interface,	Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Hornbacker at 5:3: Referring first to FIG. 1, a network comprising **client workstations 10 and 20** are connected through network connections to a network image view server 100 comprising a network server interface, preferably a web server 30 which uses the Hypertext Transfer Protocol (HTTP), a request broker 40, a foreground view composer 50, a view tile cache 60, a background view composer 80, a garbage collector 70, and a document repository 90 having image files. The network image view server, i.e. , client workstation, or "workstation," 100 can be implemented on a computer, for example **a personal computer configured with a processor,** I/O, memory, disk storage, and a network interface. The network image view server 100 is configured with a network server operating system and Web server software 30 to provide the network HTTP protocol link with the client workstations 10 and 20. Typical networks include many workstations served by one, and sometimes more than one, network server, the server functioning as a library to maintain files which can

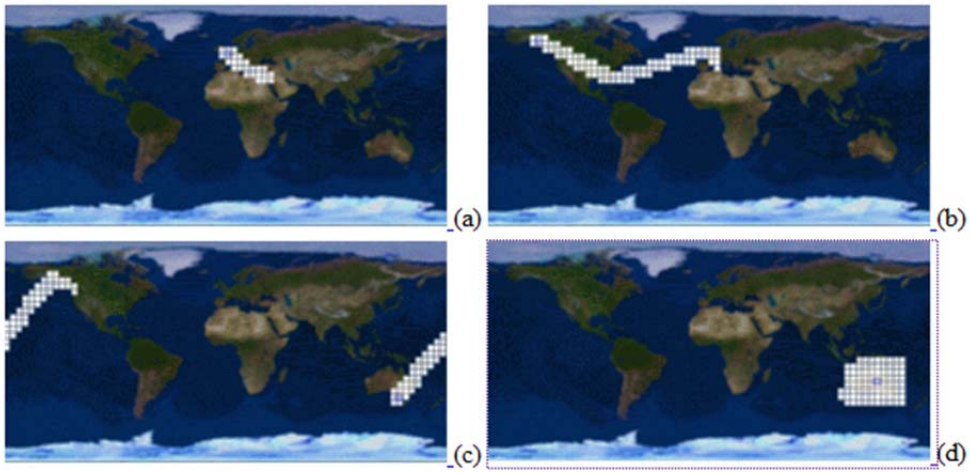
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	be accessed by the workstations. In operation according to an embodiment of the method of the invention**, using the Web browser software on the client workstation**, a user requests an image view 110 (FIG. 2) having a scale and region specified by by means of a specially formatted Uniformed Resource Locator (URL) code using HTTP language which the Web server can decode as a request to be passed to the image view composition software and that identifies the image file to be viewed, the scale of the view and the region of the image to view.
8.F: said processor operative to select said defined data parcel,	Potmesil, Abstract: We describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or **in full 3D environments**. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and **a 3D flight-simulator browser capable of continuous flight around the Earth**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. **The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space** as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees.

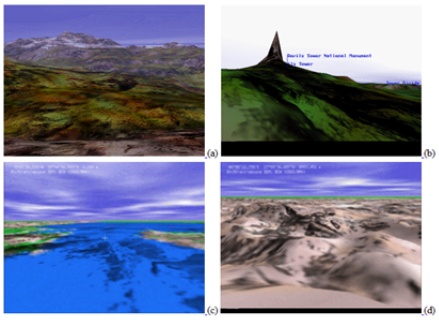
8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1327-29: Internet-based computers and communications can be very effective in enhancing our ability to visualize and to search 3D environments in the great outdoors where we move, work, play and learn. In this paper, we describe a WWW-based system - consisting of browsers, servers and connecting protocols - which allows users to view, search and post geographically-indexed information. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. In addition, very large data bases of geographical information itself, such as terrain elevation, satellite and aerial images, detailed street maps and geometrical models of buildings and similar man-made structures (present, past and future) are also becoming available. We seek to build an integrated system which will allow its users to browse in such spatial data, make queries and post new data. Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and **a 3D flight-simulator browser**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. **The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space** as well as on the latency of server replies. The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1332: 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. **The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there**. This process is shown in Figure 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user, least-recently visible tiles, or least-recently arrived tiles. **The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there. There are several implemented caching strategies: • obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail [Figure 2(a,b,c)], • obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain [Figure 2(d)], **obtain as many tiles ahead as possible, in a widening wedge**, and delete any tiles already visited; this may be used during high-speed flight. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. Potmesil at 1340-41:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3.4 A 3D Geographical Browser We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. The browser uses the OpenGL library to render 3D graphics. To make the three-dimensional browser truly global, we represent the Earth as an ellipsoid or geodetic datum called World Geodetic System 1984 (WGS84) [4]. Potmesil, Fig. 2:  **Figure 2** Contents of the browser's cache memory after (a) flying from Egypt to Britain, (b) to Alaska, (c) to Australia, and (d) hovering above Australia. Potmesil, Fig. 8:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	 **Figure 8** Four views from the 3D browser: (a) Tuolumne Meadows, Yosemite National Park, California in fog, (b) Devils Tower, Wyoming with names, (c) east from Atlantic Ocean across Strait of Gibraltar, and (d) southwest from the top of Mount Everest towards India.
8.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies.** The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. **Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed**. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Potmesil at 1332-33: In this paper, we describe a system for viewing geospatial models which reside in serer hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. the servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally…. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. *3.1 Tile Caching* **The cache process allocates a common tile memory that is shared by all mapplets**. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Fig. 2. **When the tile cache is full, some resident tiles need to be deleted**. These can be tiles furthermost from the user: least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). **The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there**. There are several implemented caching strategies: • Obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail (Fig. 2a-c), • Obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain (Fig. 2d), • Obtain as many tiles ahead as possible, in a widening wedge, and delete any tiles already visited; this may be used during high-speed flight. **When the caching process has generated a new list of tiles to be cached**, each mapplet can start loading its data into each tile. **Mapplets also provide feedback to the cache process**: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	3.2 Tile Compositing **The tile compositing process composites tile data from the off-screen cached tiles** into the on-screen window image… The two processes run independently and asynchronously. **The cache manager keeps rearranging the cache memory** even while the user has stopped and the image is not regenerated. Potmesil at 1334: There are several libraries that the core browser makes available to the user mapplets:… • A ***caching* library** allows HTTP documents to be cached on the local client machine in memory or on disc… *3.3.1 An image applet* This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. *3.4. A 3D geographical browser* We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. Hornbacker, 6:1-19 The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	for the view, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, **unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser**. In either case, the foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server, which in turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with newly computed view tiles 220 which are fetched from the server. Hornbacker, 8:1-6 **For frequently accessed images there is a good chance that the view tiles for a view may already exist in the view tile cache since the view tile cache maintains the most recently accessed view tiles**. Since millions of view tiles may be created and eventually exceed the storage capacity of the image view server, the view tile cache garbage collector removes the least recently accessed view tiles in the case where the maximum storage allocation or minimum storage free space limits are reached. Hornbacker, 8:16-23 The HTML output file produced by the foreground view composer is passed to the Web server software to be transmitted to the Web browser. The graphical Web browser serves as the image viewer by utilizing the HTML output from the image view server to compose and display the array of view tiles that form a view of an image. The HTML page data list the size, position and the hyperlink for each view tile to be displayed. The view tiles are stored in the GIF image file format that can be displayed by all common graphical Web browsers. **The Web browser will retrieve each view tile to be displayed from a local cache if the view tile is present,** otherwise from the

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	image view server. Hornbacker, 10:13-28 The view tile generator routine 160 performs the actual creation of the view tiles according to the preferred steps shown in FIG 8. The view tile generator receives information from the view computation as to what view tiles are needed for the view**. It has access to records in the cache 80 that determine which tiles have already been created and are resident in the cache**. If a needed view tile is in the cache then its last access time is updated to prevent the cache garbage collector from deleting the view tile. If a needed view tile is not in the cache, then the view tile generator creates the view tile from the image file 90. The view tile generator uses a software imaging library that supports rendering many digital document file formats including monochrome raster images, grayscale raster images, color raster images as well as many content rich non-raster formats such as Adobe Portable Document Format (PDF), PostScript, HPGL, etc. When rendering monochrome image data the imaging library scale-to-gray scaling is used to provide a more visually appealing rendition of the reduced image. For example, a specific view request might include tiles B2, C2, B3, and C3 (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent. Hornbacker, 11:29-12:9 Preferably a view tile cache garbage collector algorithm 70 manages the use of the storage for view tiles (FIGS 10A, 10B, 10C). The garbage collector maintains the view tile cache 60 (FIG. 1) to keep the view tile cache storage usage below a storage limit and to keep the storage free space above a free space limit. The garbage collector constantly scans the cache to accumulate

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	size and age statistics for the view tiles. When the cache size needs to be reduced, the garbage collector selects the least recently accessed view tiles for deletion until the cache size is within limits. The garbage collector runs at a low priority to minimize interference with more critical system tasks. The storage free space limit is designed as a failsafe limit to prevent the system from running out of storage. The free space limit is checked on a periodic basis and if it is exceeded the garbage collector becomes a critical system task and runs at a high priority until the storage free space is greater than the free space limit. Hornbacker, 12:17-23 The graphical Web browser on the client workstation 10 receives HTML data from the image view server 210 that contains **hyperlinks to the view tiles within the view tile cache 60 to be displayed and formatting that describes the layout of the of the tiles to form the image view**. The Web browser initially must fetch each view tile 220 for a view from the view server. **After the initial view, whenever a view overlaps with a previous view at the same scale, the Web browser preferably retrieves view tiles that have been previously displayed from the Web browser's local cache 210 rather than from the server**. Hornbacker, 13:19-23 **By using image tiling and caching according to the preferred method, relatively small amounts of data need to be transmitted when the user selects a new view of an image already received and viewed**. The server sends the requested image in the request format to the workstation and then allows viewing the image from the local copy of the image file.
8.H: render said defined data parcel over a	Potmesil, Abstract: We describe a WWW-based system - consisting of browsers, servers and

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display; and	connecting protocols - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. On the server side, we have developed geographical and geometrical servers which contain **very large data bases of images, elevations, lines, points and polygons stored in tiles structured into hierarchical pyramids or quadtrees**. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil, Fig. 1:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

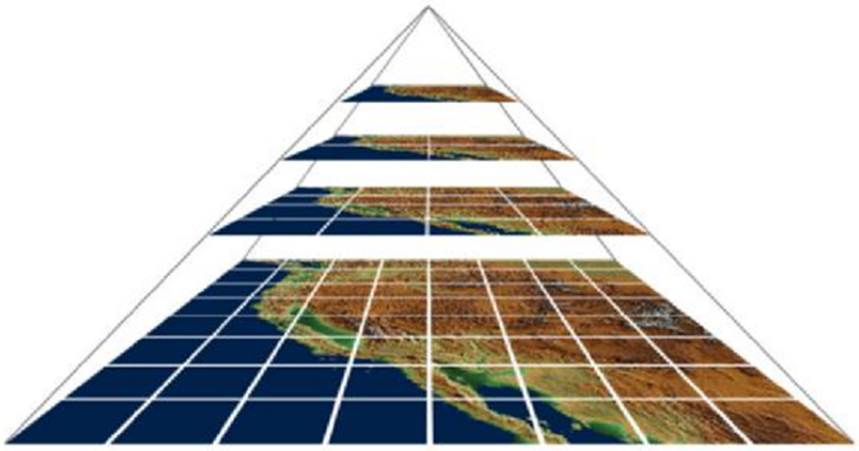


Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

Potmesil at 1329-1330:

2.1 A Tile Server

The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections. A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. **All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. **Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats**. The index of each tile data set is read into the server at startup time and stored in a quadtree [14].

3.1 Tile Caching

The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Fig. 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user: least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). **The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there**. There are several implemented caching strategies: • Obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail (Fig. 2a-c), • Obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain (Fig. 2d),

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	• **Obtain as many tiles ahead as possible, in a widening wedge**, and delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. Potmesil at 1332: 3. A Geographical Browser In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally. Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and image pyramids for 2D lattice data such as images, elevations or gradients. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14]. **The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]**. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, **level-of-detail** and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. Hornbacker, 12:24-13:10 **Performance and usability of document viewing can be increased by using progressive display of tiled images. By using an image file format that allows a rough view of the image to be displayed while the remainder of the image content is downloaded, a rough view of the document can be seen more quickly**. Since most Web browsers can only transfer 1 to 4 GIF images at a time, usually not all of the view tiles in the view array can be progressively displayed at the same time. Therefore, it is preferred that to implement progressive display, algorithms at the client are provided to accept an alternate data format that would allow the whole document viewing area screen to take advantage of the progressive display while still taking

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	advantage of the benefits of tiling and caching at the client. This can be accomplished in a Web browser environment using algorithms written in Java, JavaScript, or ActiveX technologies. By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile.
8.I: a remote computer, coupled to the limited bandwidth communications channel, that delivers the defined data parcel	Potmesil, Abstract: We describe **a WWW-based system - consisting of browsers, servers and connecting protocols** - which allows users to view, search and post geographically-indexed information of the Earth. Much information available on the WWW, such as weather reports, home pages of National Parks, VRML models of cities, home pages of Holiday Inn hotels, Yellow and White Page directory listings or traffic and news reports, is better located and visualized when displayed directly or via clickable anchors on top of 2D maps or in full 3D environments. We have developed two geographical browsers: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet and a 3D flight-simulator browser capable of continuous flight around the Earth**. Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles.** The tile caching process is based on the user's current position, velocity, and acceleration in the 2D/3D space as well as on the latency of server replies. A user can program these browsers by adding small application programs - mapplets. **On the server side, we have developed geographical and geometrical servers which contain very large data bases of images, elevations, lines,**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	points and polygons stored in tiles structured into hierarchical pyramids or quadtrees. We have also developed a metadata server which contains, in hierarchical layers, URL pointers and geographical coordinates of various WWW documents, geographical information and geometrical models. Potmesil at 1328: Two geographical browsers have been developed for this system: a 2D map browser capable of continuous scroll and zoom of an arbitrarily large sheet of 2D information and a 3D flight-simulator browser. **Both browsers download and cache geographical information, geometrical models, and URL anchors in small regions called tiles. The tile caching process is based on the user's current position, velocity, and acceleration in a 2D/3D space as well as on the latency of server replies.** The browsers query servers only for relevant data around the user's current and predicted future locations and expect to receive such data and to prepare them for display before the user reaches it. Around this core concept of tile caching, various specialized visualization applets - written in C, C++ or Java - are developed. Such applets run simultaneously on top of the browser and convert all their respective data into a common coordinate system specified by the browser. Examples of such applets are weather and traffic reports, bird migrations, and a spatial bulletin board applet which displays an anchor of any WWW document at any geographical location. Each applet typically queries two servers: a spatial meta server, which knows what information is available at what geographical location and where on the WWW to find it, and the server which contains the information itself. Potmesil at 1329-30: 2. Geospatial Servers The concept of a geography server system recognizes that **a digital map or a 3D geographical model is held by many independent sources, distributed**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	over a network. The objective of a browser is to gather all the necessary geographical layers, on as-needed basis, without having to store them locally and to display them. Our architecture of a geography server system has three major components: a directory scheme for finding servers, a common interface protocol for talking to the servers, and a strategy for implementing the servers themselves. We have developed four different types of servers, so far, in this project: the first three contain actual geographical geometry - (1) points sampled on grids, (2) random points with names, and (3) lines and polygons with names - while the last type stores metadata - information about where to find spatial and geographical information. 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid. This may be satellite and aerial images, terrain elevations and gradients or geoid corrections.** A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations** [Figure 1]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, **a second process is spawned to send tiles to the client while the first process receives tile requests from the client.** Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game.

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it. Potmesil at 1332-33: In this paper, we describe a system for **viewing geospatial models which reside in server hosts across the Internet network**. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. the servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally…. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. *3.1 Tile Caching* The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. This cache allocation is currently based on five parameters: x, y, z, level-of-

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid. Time in this context is interpreted as discrete time slices. The caching algorithm uses the user's current position, velocity, and acceleration to estimate where the user is moving and allocates new tiles there. This process is shown in Fig. 2. When the tile cache is full, some resident tiles need to be deleted. These can be tiles furthermost from the user: least-recently visible tiles, or least-recently arrived tiles. The caching process receives information about the current view from the compositing process. A 2D browser may have multiple windows opened, each with an orthographic projection of a different location and scale of a map. A 3D browser may have also multiple windows opened, each with a different perspective projection. Each window can be moving completely independently of all the others, or they may be different views from one user (e.g., left and right views from a cockpit, or the view of a tail gunner). **The caching process computes one or more estimated positions of each view and intersects their bounding volume with the tile coordinate system. Any intersected tiles not present in the cache are sorted by distance from the user, and the caching algorithm determines how many of them can be loaded into the cache. This depends on the total number of allocated tiles for we need to prevent tile thrashing. The more disc and memory space the host machine has available, the more tiles can be brought into the cache and remain there**. There are several implemented caching strategies: • Obtain only tiles in a narrow corridor along the user's path; the cached tiles look like a snake with a growing head and disappearing tail (Fig. 2a-c), • Obtain as many tiles as close to the view as possible; this may be used in low speeds (hovering) when the direction of flight is uncertain (Fig. 2d), • Obtain as many tiles ahead as possible, in a widening wedge, and

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	delete any tiles already visited; this may be used during high-speed flight. When the caching process has generated a new list of tiles to be cached, each mapplet can start loading its data into each tile. Mapplets also provide feedback to the cache process: each tile is marked by each mapplet when it has been drawn, and each mapplet saves the average time it takes to receive and draw the tile data. *3.2 Tile Compositing* **The tile compositing process composites tile data from the off-screen cached tiles** into the on-screen window image… The two processes run independently and asynchronously. **The cache manager keeps rearranging the cache memory** even while the user has stopped and the image is not regenerated. Potmesil at 1334-35: 3.3.1 An Image Applet This is the fundamental mapplet, by default always enabled by the browser. **It obtains tile data from the tile server described in Section 2.1 and converts them into images in the cached tiles**. The tiles received from the server are processed in three pipelined steps: (1) an optional decompression, (2) mapping into an image, (3) conversion to the local display format. **The mapplet may request compressed tiles from the tile server** if the speed of the network connection justifies the additional time spent by the mapplet in tile decompression. The elevation data are usually compressed using a wavelet compression [7], while the gradient and image data are usually compressed using JPEG. When using a slower network, the gradient data may be computed locally by the mapplet rather than downloaded from the server. When all the tile components are decompressed, they are converted

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	into an image using one of these mappings: • elevation + gradient -> color shaded relief - maps elevations into ambient colors - via an elevation lookup table - and adds relief shading based on the local surface gradient - via a second lookup table; Horn [10] describes many relief shading techniques, • elevation -> elevation-mapped color - maps elevations into elevation colors without gradient shading using only the first lookup table above, • gradient -> grey-tone shaded relief - maps gradients into a monochrome shaded relief image using only the second lookup table above, • rgb + gradient -> shaded-relief image - maps gradients into a monochrome shaded-relief image which is added to a color image (typically a LANDSAT image), • mono + gradient -> shaded-relief image - same as above but the shaded-relief is added to a monochrome image (typically a DOQ image). Finally, following the above mappings, if the local display buffer is 24/32-bits deep, a true-color image is displayed. However, if the frame buffer is only 8-bits deep, a dithered image - using ordered dither algorithm - or a monochrome image is displayed. 3.3.2 A GIF Applet **This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents.** Optionally, in addition to the GIF transparency value, an alpha-blending value can be specified to make the image background partially visible.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and (3) the multi-resolution Mars image server at the Los Alamos National Laboratory. Potmesil at 1340-41: *3.4. A 3D geographical browser* We have developed a preliminary version of a three-dimensional browser which displays terrain data cached from the tile server and geographical names cached from the name server. Hornbacker teaches that tiles are received over the internet: Hornbacker, 3:10-27: These objects, and others which will become apparent from the following disclosure, are achieved by this invention which comprises in one aspect method of identifying and delivering a graphical image from a computer network file server comprising providing a network file server on which are stored digital document image files, said server adapted to receive **requests from a Web browser in Uniform Resource Locator (URL) code, to identify the image file and format selections being requested, to compose the requested view into a grid of view tiles, and to transmit HTML code for view tiles to the requesting Web browser**. Another aspect of the invention comprises apparatus comprising a computer network server adapted to store digital document image files, programmed to **receive requests from a client Web browser in URL code, the URL specifying a view which identifies an image file and format**, to compose the requested view, and to transmit HTML code for the resultant view to the

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	client Web browser to display. A further aspect of the invention is the computer program recorded on magnetic or optical media for use on a network server comprising code which interprets HTTP requests from a workstation for a particular view of a digital document image file stored in memory, retrieves the digital document image file, **composes a grid of view tiles corresponding to the requested view of the image**, computes HTML code for the grid of view tiles in a form which can be transmitted from the server to the workstation. Hornbacker, 4:26-31: The preferred embodiment is a server PC consisting of an Intel Pentium Pro 200MHz processor, with at least 128MB of RAM, an Ultra-wide Fast SCSI disk controller with at least 4GB of hard disk space, and LAN/ WAN/Internet network interface controllers. The server runs the Windows NT Server Version 4 operating system with NT File System, Microsoft Internet Information Server Version 3, and the network image server software. The server and client are configured with TCP/IP network protocols to support the HTTP (Web) protocol. Hornbacker, 5:3-6:19: Referring first to FIG. 1, **a network comprising client workstations 10 and 20 are connected through network connections to a network image view server 100** comprising a network server interface, preferably a web server 30 which uses the **Hypertext Transfer Protocol (HTTP),** a request broker 40, a foreground view composer 50, a view tile cache 60, a background view composer 80, a garbage collector 70, and a document repository 90 having image files. **The network image view server, i.e. , client workstation, or "workstation," 100 can be implemented on a computer, for example a**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	personal computer configured with a processor, I/O, memory, disk storage, and a network interface. The network image view server 100 is configured with a network server operating system and Web server software 30 to provide the network HTTP protocol link with the client workstations 10 and 20. Typical networks include many workstations served by one, and sometimes more than one, network server, **the server functioning as a library to maintain files which can be accessed by the workstations.** In operation according to an embodiment of the method of the invention, using the Web browser software on the client workstation, a user requests an image view 110 (FIG. 2) having a scale and region specified by by means of a specially formatted Uniformed Resource Locator (URL) code using HTTP language which the Web server can decode as a request to be passed to the image view composition software and that identifies the image file to be viewed, the scale of the view and the region of the image to view. The network image server sends HTML data to the client with pre-computed hyperlinks, such that following a hyperlink by clicking on an area of an image will send a specific request to the server **to deliver a different area of the drawing or to change the resolution of the image**. The resultant HTML from this request will also contain pre-computed hyperlinks for other options the user may exercise. The code is sent over the network to the network server where the web server software interprets the request 120, passes the view request URL to the foreground view composer software through a common gateway interface (CGI) that is designed to allow processing of HTTP requests external to the Web server software, and thereby instructs the request broker 130 to get the particular requested view, having the scale and region called for by the URL. The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	for the view, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser. In either case, the foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server, which in **turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with newly computed view tiles 220 which are fetched from the server.** Hornbacker, 7:26-8:6: **The use of view tiling also allows the image view server to effectively pre-compute view tiles that may be required by the next view request. The image view server background view composer computes view tiles that surround the most recent view request in anticipation a request for a shifted view. When the shifted view is requested, the foreground view composer can use the pre-computed view tiles and eliminate the time to compute new view tiles for the view. For frequently accessed images there is a good chance that the view tiles for a view may already exist in the view tile cache since the view tile cache maintains the most recently accessed view tiles**. Since millions of view tiles may be created and eventually exceed the storage capacity of the image view server, the view tile cache garbage collector removes the least recently accessed view tiles in the case where the maximum storage allocation or minimum storage free space limits are reached. Hornbacker, 8:16-23 The HTML output file produced by the foreground view composer is passed

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	to the Web server software to be transmitted to the Web browser. The graphical Web browser serves as the image viewer by utilizing the HTML output from the image view server to compose and display the array of view tiles that form a view of an image. The HTML page data list the size, position and the hyperlink for each view tile to be displayed. The view tiles are stored in the GIF image file format that can be displayed by all common graphical Web browsers. **The Web browser will retrieve each view tile to be displayed from a local cache if the view tile is present,** otherwise from the image view server. Hornbacker, 9:29-10:2: The **Web server 30** is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:13-28 The view tile generator routine 160 performs the actual creation of the view tiles according to the preferred steps shown in FIG 8. The view tile generator receives information from the view computation as to what view tiles are needed for the view**. It has access to records in the cache 80 that determine which tiles have already been created and are resident in the cache**. If a needed view tile is in the cache then its last access time is updated to prevent the cache garbage collector from deleting the view tile. If a needed view tile is not in the cache, then the view tile generator creates the view tile from the image file 90. The view tile generator uses a software imaging library that supports rendering many digital document file formats including monochrome raster images, grayscale raster images, color raster images as well as many content rich non-raster formats such as Adobe Portable Document Format (PDF), PostScript, HPGL, etc. When rendering

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	monochrome image data the imaging library scale-to-gray scaling is used to provide a more visually appealing rendition of the reduced image. For example, a specific view request might include tiles B2, C2, B3, and C3 (FIG 4A and FIG 5A). If, after viewing those tiles, the client decides that the view to the immediate left is desired, then the server would send tiles A2 and A3 (FIG 4B and FIG 5B). This assumes that the client retains in a cache the other tiles. If the client does not cache then tiles A2, A3, B2, and B3 are sent. Hornbacker, 12:17-23 The graphical Web browser on the client workstation 10 receives HTML data from the image view server 210 that contains hyperlinks to the view tiles within the view tile cache 60 to be displayed and formatting that describes the layout of the of the tiles to form the image view. **The Web browser initially must fetch each view tile 220 for a view from the view server**. After the initial view, whenever a view overlaps with a previous view at the same scale, the Web browser preferably retrieves view tiles that have been previously displayed from the Web browser's local cache 210 rather than from the server. Hornbacker, 13:17-14:16: The method, apparatus, and software article of the invention provide **an improved client-server** architecture using a graphical Web browser to access the **image view server** which makes efficient use of the network. By using image tiling and caching according to the preferred method, relatively small amounts of data need to be transmitted when the user selects a new view of an image already received and viewed. **The server sends the requested image in the request format to the workstation** and then allows viewing the image from the local copy of the image file. **The image view server** provides a better solution by utilizing lower cost graphical Web browsers for the workstations to access a network image view server that provides views

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	of the image that can be displayed on the workstation by the graphical Web browser. For example an E-size engineering drawing raster image file is 8 million bytes in size when imaged in monochrome at 200 pixels-per-inch. With commonly used data compression the image file can be reduced to 250 kilobytes. With a low bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput, it 83 seconds (250 KB / 3 KB/second) to transfer the image file to the workstation application for viewing. **With the image view server** only the image data to be displayed needs to be transmitted. A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles. The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second). This method of image viewing provides better response to the user with much lower demand on the network connection. A local-area-network typically utilizes a 10 megabit-per-second media so the savings from the efficiency of the image view server does not seem obvious. However, if the 10 megabit-per-second network is shared by 100 users, then the average bandwidth per user is only about 12.5 kilobytes-per-second so the efficiency of the image view server is still a benefit. Another benefit of **the image view server** is that the data transfer size remains constant even if the size of the view image is increased. If the image file size was 4 times larger than with the previous example as may be the case with a larger image, a higher resolution image or a less compressible image then the network load by **the image view server** would remain unchanged while network load of the traditional image viewer would quadruple.
8.J: wherein delivering the defined data parcel further comprises	Potmesil at 1329-30: 2.1 A Tile Server **The tile server stores data that was obtained by sampling on a 2D grid.**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
processing source image data to obtain a series K1-N of derivative images of progressively lower image resolution and	**This may be satellite and aerial images, terrain elevations and gradients or geoid corrections.** A data set is stored in a tile index. A data set may have several components such as: elevation, gradient, and rgb image. All tiles in a data set have usually the same size with the possible exception of tiles along the edges of the data set. **Tiles in an index are stored in a power-of-two pyramid to allow fast access and scroll and zoom operations [Figure 1**]. Storing data in a power-of-two pyramid requires 1/4 + 1/8 + 1/16 + 1/32 + ... = 1/3 additional storage space. A data set may also be stored on one or more compressed formats. The index of each tile data set is read into the server at startup time and stored in a quadtree [14]. The server was designed to maintain maximum tile output to a large number of clients which are connected by fast networks. The server is multi-threaded: it serves multiple clients simultaneously. For each opened connection it spawns a separate process. If a connection is permanently kept opened, a second process is spawned to send tiles to the client while the first process receives tile requests from the client. Tiles, which have been recently read from discs, are saved in shared memory so that other clients can obtain them more quickly. This is useful when multiple clients are browsing in the same data as would happen in a networked game. The server, using the HTTP/1.0 protocol [3], accepts two types of query: (a) send me a description of the requested tile index, (b) send me the contents of the requested tile. **The output of the server has several pipelined stages which: (a) reformat the tile if the requested tile is not aligned with tiles stored in the server; (b) resample the tile if the requested tile is not in the same coordinate system; (c) dither the tile if the requesting client has only a limited number of colors; (d) add a digital watermark [2] if the tile data is copyrighted or encrypt the tile if it is to be seen only by the client; (e) compress the tile if the network bandwidth requires it.**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

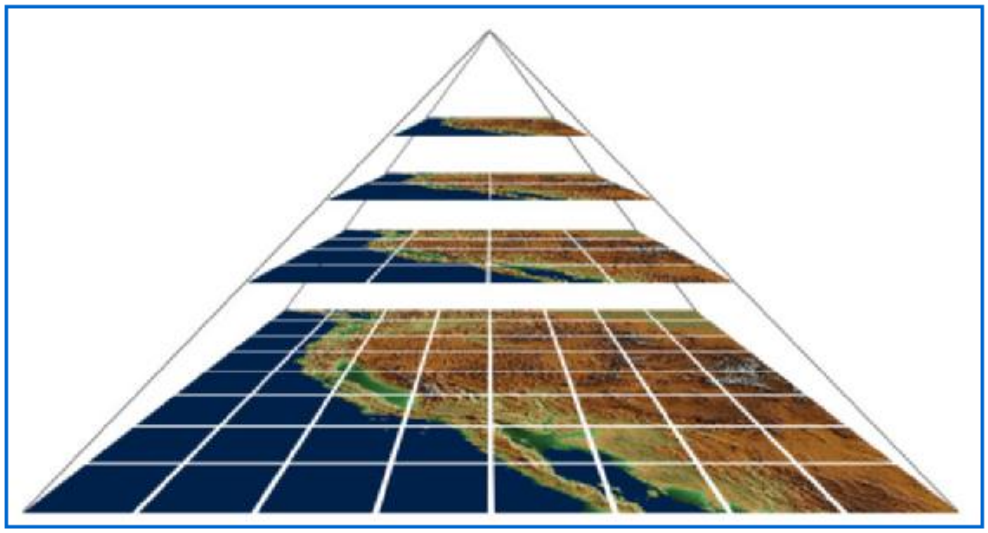


Figure 1 A hierarchical representation of tiles in a pyramid: 2×2 tiles are filtered [11] into a single tile in the next higher level.

Potmesil, 1332:

3. A Geographical Browser

In this paper, we describe a system for viewing geospatial models which reside in server hosts across the Internet network. The client browsers, which are described here, have the ability to cache parts of the geospatial models before a user needs to display them. The servers can generate the models in small sections - called tiles - because they store them in hierarchical representations or have the ability to clip all parts of a model outside the requested area (or volume). We base this approach on the assumption that the amount of such models far exceeds the ability to store these models locally.

Much work in this paper is centered around two data representations: quadtrees for geometrical elements such as points, lines and polygons and **image pyramids for 2D lattice data such as images, elevations or gradients**. Quadtree data structures and algorithms, many of them for geographical applications, are described in books and papers by Samet [14].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	The concept of prefiltered power-of-two images for texture mapping was introduced by Williams [17] who named them mip maps. Since his seminal paper, it has become a rendering standard implemented, for example, in OpenGL software [12] and hardware [1]. In the 2D browser we use any type of data sampled on a 2D lattice. However, our techniques are applicable to any other model representations such as TIN's (Triangulated Irregular Networks) of terrain or VRML models which are clipped to rectangular regions. In more complex environments, such as furnished interiors of buildings, one must use more sophisticated data structures and display algorithms to maintain interactive display rates [8]. The browser consists of two processes: caching and compositing. The former process is responsible for managing the local cache while the latter process reads tracking data, synchronizes all application mapplets, and composites the final image. It also makes space (data and user) and time (either real or simulated) consistent among all the mapplets. 3.1 Tile Caching The cache process allocates a common tile memory that is shared by all mapplets. It controls how the cached tiles are allocated in space and time. **This cache allocation is currently based on five parameters: x, y, z, level-of-detail and time. In a 2D mode, the level-of-detail parameter is used as a discrete z level in a 3D pyramid**. Time in this context is interpreted as discrete time slices. Potmesil, 1335: 3.3.2 A GIF Applet This mapplet obtains image tiles, stored or generated as GIF images, from several WWW servers. The mapplet obtains a GIF image, decodes it and draws it on top of the current tile contents. Optionally, in addition to the GIF

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	transparency value, an alpha-blending value can be specified to make the image background partially visible. Currently, the mapplet can obtain maps and images from three outside sources: (1) the well-known Xerox PARC map server which contains data from the DMA's Digital Chart of the World and the USGS's 1:2,000,000 Digital Line Graph, (2) the U.S. Bureau of the Census TIGER street map server, and **(3) the multi-resolution Mars image server at the Los Alamos National Laboratory.** Hornbacker also discloses that view tiles are generated at the server by an image tiling routine that divides a given image into a grid of smaller images, which are further computed for distinct resolutions. The view tiles may either be pre-processed at the server (pre-cached) or newly computed in response to a request: Hornbacker, 3:22-27: A further aspect of the invention is the computer program recorded on magnetic or optical media **for use on a network server** comprising code which interprets HTTP requests from a workstation for a particular view of a digital document image file stored in memory, retrieves the digital document image file, **composes a grid of view tiles corresponding to the requested view of the image**, computes HTML code for the grid of view tiles in a form which can be transmitted from the server to the workstation. Hornbacker, 5:3-8: Referring first to FIG. 1, a network comprising client workstations 10 and 20 are connected through network connections to **a network image view server 100** comprising a network server interface, preferably a web server 30 which uses the Hypertext Transfer Protocol (HTTP), a request broker 40**, a**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	foreground view composer 50, a view tile cache 60, a background view composer 80, a garbage collector 70, and a document repository 90 having image files. Hornbacker, 5:16-6:19: The code is sent over the network to the network server where the web server software interprets the request 120, passes the view request URL to the foreground view composer software through a common gateway interface (CGI) that is designed to allow processing of HTTP requests external to the Web server software, and thereby instructs the request broker 130 to get the particular requested view, having the scale and region called for by the URL**. The foreground view composer is initialized 140 and composes the requested view 150 after recovering it from memory on the network server. The foreground view composer software interprets the view request, computes which view tiles are needed for the view, creates the view tiles 160 needed for the view**, and then creates Hypertext Markup Language (HTML) output file to describe the view composition to the Web browser, unless the necessary view tiles to fulfill the request are already computed and stored in cache memory of the workstation, in which case the already-computed tiles are recovered by the Web browser. In either case, t**he foreground view composer formats the output 170 and then intitializes backgound view composer 180 which passes the formatted output to the Web server**, which in turn transmits the formatted output over the network to the Web browser 200 on the requesting workstation 10, where the requesting browser displays any view tiles already cached 210, combined with n**ewly computed view tiles 220 which are fetched from the server.** **The generation of the view tiles 160 is handled by an image tiling routine which divides a given page, rendered as an image, into a grid of smaller images (FIG 3 A) called view tiles Al , A2, Bl, etc. (or just tiles in the image view server context). These tiles are computed for distinct resolutions (FIG 3B) of a given image at the server according to the URL request received from the browser software on the workstation. The use**

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	of tiling enables effective image data caching 60 at the image view server and by the browser 10 at the client workstation. Hornbacker further teaches that the image tiles created at the server are processed into a series of images of progressively lower resolutions based on the source data: Hornbacker, 6:20-7:25 T**he preferred view tile format is 128 pixel by 128 pixel GIF image file**s. The GIF image file format is preferred because of Web browser compatibility and image file size. The GIF image file format is the most widely supported format for graphical Web browsers and therefore gives the maximum client compatibility for the image view server. The GIF image format has the desirable properties of loss-less image data compression, reasonable data compression ratios, color and grayscale support, and a relatively small image file header, which relates to the selection of view tile size. With a raw image data size for monochrome view tiles of 2,048 bytes and a typical GIF compression of 4 to 1, the compressed data for a view tile is approximately 512 bytes. With many image file formats, such as TIFF and JPEG, the image file header (and other overhead information such as data indexes) can be as large or larger than the image data itself for small images such as the view tiles; whereas a GIF header for a monochrome image adds as little as 31 bytes to the GIF image file. Alternate view tile formats such as Portable Network Graphics (PNG) may be used, especially as native browser support for the format becomes common. The 128 pixel view tile size is a good compromise between view tile granularity and view tile overhead. The view tile granularity of 128 pixels determines the minimum view shift distance (pan distance) that can be achieved with standard graphical Web browser and level 2 HTML formatting. This allows the adjustment of the view position on a 0.64 inch grid when viewing a 200 pixel-per-inch image at 1 to 1 scale. Reducing the size of the view tiles allows finer grid for view positioning, but has the

8,924,506 Patent Claim Language	Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")
	problem that the view tile overhead becomes excessive. A view tile typically represents more or less than 128 x 128 pixels of the image file. **If the view being displayed is reduced 2 to 1 , then each view tile will represent a 256 x 256 pixel area of the image file that has been scaled down to 128 x 128 pixels. For each possible scale factor there is an array of tiles to represent the view. Fixed size view tiling is beneficial because it allows more effective use of the caching mechanism at the server and at the client.** For example, consider a view of 512 pixels by 512 pixels. Without tiling, this view is composed of a single GIF file that is displayed by the Web browser, and so if the user asks for the view to be shifted by 256 pixels, then a new GIF image of 512 x 512 pixels needs to be created and transmitted to the Web browser. With tiling, the first view would cause 16 view tiles to be computed and transmitted for display by the Web browser. When the request for the view to be shifted by 256 pixels is made, only 8 view tiles representing an area of 256 by 512 pixels need to be computed. In addition only the 8 new view tiles need to be transmitted to the Web browser since the shifted view will reuse 8 view tiles that are available from the Web browser cache. The use of tiling cuts the computation and data transmission in half for this example. Hornbacker, 8:30-9:28 To support the tiling and caching of many images on the same image view server, each view tile must be uniquely identified for reference by the Web browser with a view tile URL. This uniqueness is accomplished through a combination of storage location and view tile naming. Uniqueness between images is accomplished by having a separate storage subdirectory in the view tile cache for each image. Uniqueness of view tiles for each scale of view is accomplished through the file name for each view tile. The view tile name is preferably of the following form:

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	V < SCALE > < TILE NUMBER > .GIF **The < SCALE > value is a 2 character string formed from the base 36 encoding of the view scale number as expressed in parts per 256.** The < TILE NUMBER > value is a 5 character string formed from the base 36 encoding of the tile number as determined by the formula: TILE NUMBER = TILE ROW * IMAGE TILE WIDTH + TILE COLUMN The TILE ROW and TILE COLUMN values start at 0 for this computation. For example the second tile of the first row for a view scaled 2: 1 would be named under the preferred protocol: V3J00001.GIF The full URL reference for the second tile of the first row for image number 22 on the image view server would be: http : //hostname/view-tile-cache-path/000022/ V3 J00001. GIF In addition to the view tile position and view scale, other view attributes that may be encoded in the view tile storage location or in the view tile name. These attributes are view rotation angle, view x-mirror, view y-mirror, invert view. A view tile name with these extra view attributes can be encoded as: V < SCALE > < TILE_NUMBER> <VIEW_ANGLE> <X_MIRROR> < Y_ MIRROR > < INVERT > . GIF VIEW ANGLE is of the form A < ANGLE > . X MIRROR, Y MIRROR, and INVERT are encoded by the single characters X, Y, and I respectively. An example is: V3J00001A90XYI.GIF The Web server 30 is configured to recognize the above-described specially formatted request Uniform Resource Locators (URL) to be handled by the image view server request broker 40. This is done by association of the request broker 40 with the URL path or with the document filename extension. Hornbacker, 10:3-10 The foreground view composer 50 interprets the view request command 140 to determine what view needs to be composed. The view request may be

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	absolute by defining scale and position, relative by defining scale and position as a delta to a previous view, or implied by relying on system defaults to select the view. View computation software routine 150 is illustrated in FIG 7 wherein **the command interpreter 151 takes the view request and determines 152 what scale view tile grid is needed for the view** and what view tiles within the grid are needed for the view 150 (FIG. 2), and generates the view tile 153, resulting in formatted view output 154. Hornbacker, 11:19-28 FIG 6 A illustrates how the background view composer algorithm works. Assuming that for a given view requested by the client, tiles C3, C4, D3 and D4 are delivered, after those tile are delivered to the Web browser, the background view composer routine within the server program creates the tiles around these tiles, starting at E4, by composing or computing such surrounding tiles. As long as the client continues to view this page at this scale factor, the server will compute view tiles expanding outward from the tiles requested last. FIG 6B illustrates another request made by a client, after the two rotations of tiles were generated. The request asked for tiles G3, G4, H3, and H4. When the tile pre-computation begins for this request it will create tiles G5, H5, 15, 14, 13, 12, H2, and G2 in the first rotation, but it will not attempt to create tiles in the F column. Hornbacker, 12:21-13:10 Performance and usability of document viewing can be increased by using progressive display of tiled images. By using an image file format that allows a rough view of the image to be displayed while the remainder of the image content is downloaded, a rough view of the document can be seen more quickly.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
	Since most Web browsers can only transfer 1 to 4 GIF images at a time, usually not all of the view tiles in the view array can be progressively displayed at the same time. Therefore, it is preferred that to implement progressive display, algorithms at the client are provided to accept an alternate data format that would allow the whole document viewing area screen to take advantage of the progressive display while still taking advantage of the benefits of tiling and caching at the client. This can be accomplished in a Web browser environment using algorithms written in Java, JavaScript, or ActiveX technologies. By using client software to enhance the client viewer, additional enhancements to performance can be made by using alternate view tile image formats and image compression algorithms. A significant example would be to use the Portable Network Graphics (PNG) format with the optimization of having the image view server and client transfer only one image header common to be shared by all view tiles and then sending the low-resolution compressed image data for each view tile followed by the full-resolution image data for each view tile. Hornbacker, 13:26-14:6 For example an E-size engineering drawing raster image file is 8 million bytes in size when imaged in monochrome at 200 pixels-per-inch. With commonly used data compression the image file can be reduced to 250 kilobytes. With a low bandwidth 28.8 kilobaud modem network connection with approximately 3 kilobytes-per-second throughput, it 83 seconds (250 KB / 3 KB/second) to transfer the image file to the workstation application for viewing. With the image view server only the image data to be displayed needs to be transmitted. **A typical view size of 896 by 512 pixels is made up of a 7 by 4 array of 128 pixel x 128 pixel view tiles**. The monochrome view tiles are transmitted in a compressed format that typically yields tiles that are 512 bytes each so the entire view is approximately 14 kilobytes (0.5 KB x 28 tiles) and the transfer takes approximately 4.8 seconds (14 KB / 3 KB/second).

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
8.K: wherein series image K_0 being subdivided into a regular array	See teachings for claim element 1.E.
8.L: wherein each resulting image parcel of the array has a predetermined pixel resolution	See teachings for claim element 1.F.
8.M: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See teachings for claim element 1.G.
8.N: resolution of the series K1-N of derivative images being related to that of the source image data or predecessor	See teachings for claim element 1.H.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
image in the series by a factor of two, and	
8.O: said array subdivision being related by a factor of two	See teachings for claim element 1.I.
8.P: such that each image parcel being of a fixed byte size.	See teachings for claim element 1.J.
9: The display system of claim 8, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand	See teachings for claim 2.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
based on the request for the updated image parcel.	
10: The display system of claim 9, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	See teachings for claim 2.
11: The display system of claim 8, wherein the limited communication bandwidth computer device further comprises one	See teachings for claim 4.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	
12: The display system of claim 8, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See teachings for claim 5.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[2] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
13: The display system of claim 8, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	See teachings for claim 6.
14.A: The display system of claim 8, wherein the processing may	See teachings for claim element 7A.

[3] For easier readability, color figures from Potmesil are copied from an online copy of the reference available at http://www.ra.ethz.ch/cdstore/www6/technical/paper130/paper130.html. The figures are identical to those in Ex. [XX].

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
further comprises compressing each data parcel and	
14.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a KD, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	See teachings for claim element 7B.
15: Preamble. A remote computer for delivering large-scale images over network communications channels for display on a limited communication	See teachings for claim element 8.I (server), claim 8 preamble (client).

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
bandwidth computer device that has a display system for displaying a large-scale image retrieved over a limited bandwidth communications channel,	
15.A: a display of defined screen resolution for displaying a defined image,	See teachings for claim element 8.A.
15.B: a memory providing for the storage of a plurality of image parcels	See teachings for claim element 8.B.
15.C: displayable over respective portions of a mesh corresponding to said defined	See teachings for claim element 8.C.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
image,	
15.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel and	See teachings for claim element 8D.
15.E: a processor coupled between said display, memory and communications channel interface,	See teachings for claim element 8E.
15.F: said processor operative to select said defined data parcel,	See teachings for claim element 8F.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
15.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	See teachings for claim element 8G.
15.H: render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display, the remote computer comprises:	See teachings for claim element 8H.
15.I: a parcel processing unit that processes a piece of source image data and	See teachings of Potmesil for claims 2, 6, 7. Hornbacker, Fig. 1, 5:3-8, 6:1-19, 10:3-23.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
15.J: delivers the defined data parcel to the limited communication bandwidth computer device; and	See teachings for claim element 8.I.
15.K: wherein the parcel processing unit further comprises a parcel processing control that processes source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	See teachings for claim element 1.D.
15.L: wherein series image K_0 being subdivided into a regular array	See teachings for claim element 1.E.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
15.M: wherein each resulting image parcel of the array has a predetermined pixel resolution	See teachings for claim element 1.F.
15.N: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See teachings for claim element 1.G.
15.O: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	See teachings for claim element 1.H.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
15.P: said array subdivision being related by a factor of two	See teachings for claim element 1.I.
15.Q: such that each image parcel being of a fixed byte size.	See teachings for claim element 1.J.
16: The remote computer of claim 15, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See teachings for claim 2.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
17: The remote computer of claim 16, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	See teachings for claim 3.
18: The remote computer of claim 15, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular	See teachings for claim 4.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	
19: The remote computer of claim 15, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See teachings for claim 5.
20: The remote computer of claim 15, wherein processing the source image data further comprises	See teachings for claim 6.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	
21.A: The remote computer of claim 15, wherein processing further comprises compressing each data parcel and	See teachings for claim element 7.A.
21.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and	See teachings for claim element 7.B.

8,924,506 Patent Claim Language	**Prior Art: M. Potmesil, "Maps alive: viewing geospatial information on the WWW," Computer Networks and ISDN Systems 29 (1997) 1327-1342[3] in view of PCT Publication No. WO 99/41675, Pub. Aug. 19, 1999 ("Hornbacker") and *An Integrated Global GIS and Visual Simulation System* by P. Lindstrom *et al.*, Tech. Rep. GIT-GVU-97-07, March 1997 ("Lindstrom")**
corresponding image array coordinate.	

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
1.Preamble. A method of retrieving large-scale images over network communications channels for display on a limited communication bandwidth computer device, said method comprising:	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
1.A issuing, from a limited communication bandwidth computer device to a remote computer, a request for an update data parcel	Rutledge: 2:25-30, 3:17-20, 5:13-17 ,8:37-42,10:1-7,7:48-62,10:10-19 Ligtenberg, 8:36-43. Cooper: Abstract, 3:49-53; 4:61-5:8; 6:5-10; 6:16-20, FIG. 2, 4:34-37, 4:48-51, 6:27-32
1.B wherein the update data parcel is selected based on an operator controlled image viewpoint on the computer device relative to a predetermined image and	Rutledge: 1:21-24, 2:62-64, 3:12-15 , 4:29-31, 4:41-47, 7:48-62, :52-56, 7:38-47, 7:48-62, 7:63 - 7:8, 3:5-10, 5:14-23, 4:41-47, 7:63 - 8:6, 9:11-17 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	Cooper: 1:13-16, 1:16-21, 4:39-41, 4:48-51, 5:34-36, 3:44-49, Abstract, 1:6-10, 1:16-21, 1:23-29, 1:34-38, 3:16-54, 3:59-65, 4:14-19, 4:27-29, 4:39-47, 4:48-51, 5:26-36, 5:35-57
1.C the update data parcel contains data that is used to generate a display on the limited communication bandwidth computer device;	Rutledge: 6:38-40, 6:45-50, 7:48-62, 10:10-17 Ligtenberg: 1:16-19, 1:34-42, 4:28-34, 5:1-8, 5:13-17 Cooper: 1:13-22, 1:31-35, 4;14-18, 5:26-36, 5:58-67
1.D processing, on the remote computer, source image data to obtain a series K.sub.1-N of derivative images of progressively lower image resolution	Rutledge: 6:38-40, 6:45-50, 7:48-62, 10:10-17 Ligtenberg: 1:16-19, 1:34-42, 4:28-34, 5:1-8, 5:13-17 Cooper: 1:13-22, 1:31-35, 4;14-18, 5:26-36, 5:58-67
1.E and wherein series image K.sub.0 being subdivided into a regular array	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
1.F wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66
1.G wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.H resolution of the series K.sub.1-N of derivative images being related to that of the source image data or predecessor image in the series by a factor of two,	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.I and said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.J such that each image parcel being of a fixed byte size;	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
1.K receiving said update data parcel from the data parcel stored in the remote computer over a communications channel; and	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
1.L displaying on the limited communication bandwidth computer device using the update data parcel that is a part of said predetermined image, an image wherein said update data parcel uniquely forms a discrete portion of said predetermined image.	Rutledge: 6:38-40, 6:45-50, 4:41-47, 5:14-64, 7:48-62 Ligtenberg: 1:16-19, 10:1-7, Appendix A, 10:9-21
2. The method of claim 1, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
3. The method of claim 2, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	Ligtenberg: 11:61-12:6 Cooper: 1:7-10, 3:9-13, 3:28-31, 4:22-27, 4:61-65, 12:22-33
4. The method of claim 1, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Ligtenberg, Col. 4:25-54 Hassan: Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5
5. The method of claim 1, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	Rutledge: 1:21-24, 2:62-64, 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, 1:16-19, 1:34-56, 5:13-17, Appendix A
6. The method of claim 1, wherein processing the source image data further comprises queuing the	Rutledge: col. 7:48-62, col. 10:19-33, 7:11-29, 7:31-34, 7:51-52

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	Cooper: Abstract, FIG. 2, col. 4:34-37, col. 4:48-51, col. 4:61 - 5:6, col. 6:16-21, col. 6:27-32, 4:55-60, 5:1-8, 6:16-20 Ligtenberg: col. 8:36-43.
7.A The method of claim 1, wherein the processing further comprises compressing each data parcel and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
7.B storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K.sub.D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Rutledge: Fig. 5, 7:63-8:44 Ligtenberg: Appendix A, 12:60:66; 13:29-38
8. A display system for displaying a large-scale image retrieved over a limited bandwidth communications channel, said display system comprising:	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
8.A a display of defined screen resolution for displaying a defined image;	Rutledge: 5:24-47, 7:38-47 Ligtenberg: Fig. 1, 1:63-67, 5:1-8
8.B a memory providing for the storage of a plurality of image parcels	Rutledge: 6:38-40, 6:45-50: Ligtenberg: 1:16-19, 10:1-7 Cooper: 6:11-10:45, FIG. 5, 6:28-38, 5:59-65, 1:54-65, 5:6-8, 6:28-32, 5:1-8, 10:2-6
8.C displayable over respective portions of a mesh corresponding to said defined image;	Rutledge: 7:48-62 Ligtenberg,: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 4:16-17, 5:39-43, 5:54-65, 6:7-12, 7:18-21
8.D a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel;	Rutledge: 1:21-24, 2:62-64 , 3:12-15, 4:29-31 , 4:41-47 Ligtenberg: 1:16-19, 1:34-42, 5:13-17, 1:22-23,. 3:50-

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	55, 4:41-47, 6:38-43, Fig. 4D, 7:41-45, Appendix A
8.E a processor coupled between said display, memory and communications channel interface,	Ligtenberg: 11:66-12:2, 4:28-34
8.F said processor operative to select said defined data parcel,	Rutledge: 7:48-62 Ligtenberg: 4:28-34
8.G retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	Ligtenberg: 1:16-19, 1:32-42, 10:1-7
8.H render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display; and	Rutledge: 7:48-62, 10:10-17 Ligtenberg: 7:7-21 Cooper: Abstract, 4:34-38, 4:48-58, 4:61-5:6, 6:16-24, 6:27-32, 7:12-16, 9:65-10:5
8.I a remote computer, coupled to the limited bandwidth communications channel, that delivers the defined data parcel	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
8.J wherein delivering the defined data parcel further comprises processing source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
8.K wherein series image K_0 being subdivided into a regular array	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.L wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
8.M wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.N resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.O said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.P such that each image parcel being of a fixed byte size.	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
9. The display system of claim 8, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
the request for the updated image parcel.	Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
10. The display system of claim 9, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	Ligtenberg: 11:61-12:6 Cooper: 1:7-10, 3:9-13, 3:28-31, 4:22-27, 4:61-65, 12:22-33
11. The display system of claim 8, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Ligtenberg, Col. 4:25-54 Hassan: Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5
12. The display system of claim 8, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	Rutledge: 1:21-24, 2:62-64, 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, 1:16-19,

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	1:34-56, 5:13-17, Appendix A
13. The display system of claim 8, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	Rutledge: col. 7:48-62, col. 10:10-17 Cooper: Abstract, FIG. 2, col. 4:34-37, col. 4:48-51, col. 4:61 - 5:6, col. 6:16-21, col. 6:27-32, 4:55-60, 5:1-8, 6:16-20 Ligtenberg: col. 8:36-43.
14.A The display system of claim 8, wherein the processing may further comprises compressing each data parcel and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
14.B storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Rutledge: Fig. 5, 7:63-8:44 Ligtenberg: Appendix A, 12:60:66; 13:29-38
15.Preamble A remote computer for delivering large-scale images over network communications channels for display on a limited	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
communication bandwidth computer device that has a display system for displaying a large-scale image retrieved over a limited bandwidth communications channel,	Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
15.A a display of defined screen resolution for displaying a defined image,	Rutledge: 5:24-47, 7:38-47 Ligtenberg: Fig. 1, 1:63-67, 5:1-8
15.B a memory providing for the storage of a plurality of image parcels	Rutledge: 6:38-40, 6:45-50: Ligtenberg: 1:16-19, 10:1-7 Cooper: 6:11-10:45, FIG. 5, 6:28-38, 5:59-65, 1:54-65, 5:6-8, 6:28-32, 5:1-8, 10:2-6
15.C displayable over respective portions of a mesh corresponding to said defined image,	Rutledge: 7:48-62 Ligtenberg,: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 4:16-17, 5:39-43, 5:54-65, 6:7-12, 7:18-21
15.D a communications channel interface supporting the retrieval	Rutledge: 1:21-24, 2:62-64 , 3:12-15, 4:29-31 , 4:41-

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
of a defined data parcel over a limited bandwidth communications channel and	47 Ligtenberg: 1:16-19, 1:34-42, 5:13-17, 1:22-23,. 3:50-55, 4:41-47, 6:38-43, Fig. 4D, 7:41-45, Appendix A
15.E a processor coupled between said display, memory and communications channel interface,	Ligtenberg: 11:66-12:2, 4:28-34
15.F said processor operative to select said defined data parcel,	Rutledge: 7:48-62 Ligtenberg: 4:28-34
15.G retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	Ligtenberg: 1:16-19, 1:32-42, 10:1-7
15.H render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display, the remote computer comprises:	Rutledge: 7:48-62, 10:10-17 Ligtenberg: 7:7-21 Cooper: Abstract, 4:34-38, 4:48-58, 4:61-5:6, 6:16-24, 6:27-32, 7:12-16, 9:65-10:5
15.I a parcel processing unit that processes a piece of source image data and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
	4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
15.J delivers the defined data parcel to the limited communication bandwidth computer device; and	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
15.K wherein the parcel processing unit further comprises a parcel processing control that processes source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
15.L wherein series image K_0 being subdivided into a regular	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
array	Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.M wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66
15.N wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.O resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.P said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.Q such that each image parcel	Rutledge:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
being of a fixed byte size.	2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
16. The remote computer of claim 15, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
17. The remote computer of claim 16, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	Ligtenberg: 11:61-12:6 Cooper: 1:7-10, 3:9-13, 3:28-31, 4:22-27, 4:61-65, 12:22-33
18. The remote computer of claim 15, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an	Ligtenberg, Col. 4:25-54 Hassan: Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	
19. The remote computer of claim 15, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	Rutledge: 1:21-24, 2:62-64, 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, 1:16-19, 1:34-56, 5:13-17, Appendix A
20. The remote computer of claim 15, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	Rutledge: col. 7:48-62, col. 10:10-17 Cooper: Abstract, FIG. 2, col. 4:34-37, col. 4:48-51, col. 4:61 - 5:6, col. 6:16-21, col. 6:27-32, 4:55-60, 5:1-8, 6:16-20 Ligtenberg: col. 8:36-43.
21.A The remote computer of claim 15, wherein processing further comprises compressing each data parcel and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
21.B storing each data parcel on	Rutledge:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") and U. S. 6,118,456 ("Cooper")**
the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Fig. 5, 7:63-8:44 Ligtenberg: Appendix A, 12:60:66; 13:29-38

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
1.Preamble. A method of retrieving large-scale images over network communications channels for display on a limited communication bandwidth computer device, said method comprising:	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
1.A issuing, from a limited communication bandwidth computer device to a remote computer, a request for an update data parcel	Rutledge: 2:25-30, 3:17-20, 5:13-17 ,8:37-42,10:1-7,7:48-62,10:10-19 Ligtenberg, 8:36-43. Cooper: Abstract, 3:49-53; 4:61-5:8; 6:5-10; 6:16-20, FIG. 2, 4:34-37, 4:48-51, 6:27-32
1.B wherein the update data parcel is selected based on an operator controlled image viewpoint on the computer device relative to a predetermined image and	Rutledge: 1:21-24, 2:62-64, 3:12-15 , 4:29-31, 4:41-47, 7:48-62, :52-56, 7:38-47, 7:48-62, 7:63 - 7:8, 3:5-10, 5:14-23, 4:41-47, 7:63 - 8:6, 9:11-17 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
	Cooper: 1:13-16, 1:16-21, 4:39-41, 4:48-51, 5:34-36, 3:44-49, Abstract, 1:6-10, 1:16-21, 1:23-29, 1:34-38, 3:16-54, 3:59-65, 4:14-19, 4:27-29, 4:39-47, 4:48-51, 5:26-36, 5:35-57
1.C the update data parcel contains data that is used to generate a display on the limited communication bandwidth computer device;	Rutledge: 6:38-40, 6:45-50, 7:48-62, 10:10-17 Ligtenberg: 1:16-19, 1:34-42, 4:28-34, 5:1-8, 5:13-17 Cooper: 1:13-22, 1:31-35, 4;14-18, 5:26-36, 5:58-67
1.D processing, on the remote computer, source image data to obtain a series K.sub.1-N of derivative images of progressively lower image resolution	Rutledge: 6:38-40, 6:45-50, 7:48-62, 10:10-17 Ligtenberg: 1:16-19, 1:34-42, 4:28-34, 5:1-8, 5:13-17 Cooper: 1:13-22, 1:31-35, 4;14-18, 5:26-36, 5:58-67

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
1.E and wherein series image K.sub.0 being subdivided into a regular array	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.F wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66
1.G wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.H resolution of the series K.sub.1-N of derivative images being related to that of the source image data or predecessor image in the series by a factor of two,	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.I and said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
	27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.J such that each image parcel being of a fixed byte size;	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
1.K receiving said update data parcel from the data parcel stored in the remote computer over a communications channel; and	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
1.L displaying on the limited communication bandwidth computer device using the update data parcel that is a part of said predetermined image, an image wherein said update data parcel uniquely forms a discrete portion of said predetermined image.	Rutledge: 6:38-40, 6:45-50, 4:41-47, 5:14-64, 7:48-62 Ligtenberg: 1:16-19, 10:1-7, Appendix A, 10:9-21
4. The method of claim 1, wherein the limited communication bandwidth computer device	Ligtenberg, Col. 4:25-54 Hassan:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5
8. A display system for displaying a large-scale image retrieved over a limited bandwidth communications channel, said display system comprising:	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
8.A a display of defined screen resolution for displaying a defined image;	Rutledge: 5:24-47, 7:38-47 Ligtenberg: Fig. 1, 1:63-67, 5:1-8
8.B a memory providing for the storage of a plurality of image parcels	Rutledge: 6:38-40, 6:45-50: Ligtenberg: 1:16-19, 10:1-7

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
	Cooper: 6:11-10:45, FIG. 5, 6:28-38, 5:59-65, 1:54-65, 5:6-8, 6:28-32, 5:1-8, 10:2-6
8.C displayable over respective portions of a mesh corresponding to said defined image;	Rutledge: 7:48-62 Ligtenberg,: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 4:16-17, 5:39-43, 5:54-65, 6:7-12, 7:18-21
8.D a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel;	Rutledge: 1:21-24, 2:62-64 , 3:12-15, 4:29-31 , 4:41-47 Ligtenberg: 1:16-19, 1:34-42, 5:13-17, 1:22-23,. 3:50-55, 4:41-47, 6:38-43, Fig. 4D, 7:41-45, Appendix A
8.E a processor coupled between said display, memory and communications channel interface,	Ligtenberg: 11:66-12:2, 4:28-34
8.F said processor operative to select said defined data parcel,	Rutledge: 7:48-62 Ligtenberg: 4:28-34
8.G retrieve said defined data parcel via said limited bandwidth	Ligtenberg: 1:16-19, 1:32-42, 10:1-7

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
communications channel interface for storage in said memory, and	
8.H render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display; and	Rutledge: 7:48-62, 10:10-17 Ligtenberg: 7:7-21 Cooper: Abstract, 4:34-38, 4:48-58, 4:61-5:6, 6:16-24, 6:27-32, 7:12-16, 9:65-10:5
8.I a remote computer, coupled to the limited bandwidth communications channel, that delivers the defined data parcel	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
8.J wherein delivering the defined data parcel further comprises processing source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
	Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
8.K wherein series image K_0 being subdivided into a regular array	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.L wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66
8.M wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.N resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
8.O said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
8.P such that each image parcel being of a fixed byte size.	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
11. The display system of claim 8, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Ligtenberg, Col. 4:25-54 Hassan: Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5
15.Preamble A remote computer for delivering large-scale images over network communications channels for display on a limited communication bandwidth computer device that has a display system for displaying a large-scale	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg:

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
image retrieved over a limited bandwidth communications channel,	1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
15.A a display of defined screen resolution for displaying a defined image,	Rutledge: 5:24-47, 7:38-47 Ligtenberg: Fig. 1, 1:63-67, 5:1-8
15.B a memory providing for the storage of a plurality of image parcels	Rutledge: 6:38-40, 6:45-50: Ligtenberg: 1:16-19, 10:1-7 Cooper: 6:11-10:45, FIG. 5, 6:28-38, 5:59-65, 1:54-65, 5:6-8, 6:28-32, 5:1-8, 10:2-6
15.C displayable over respective portions of a mesh corresponding to said defined image,	Rutledge: 7:48-62 Ligtenberg,: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 4:16-17, 5:39-43, 5:54-65, 6:7-12, 7:18-21
15.D a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth	Rutledge: 1:21-24, 2:62-64 , 3:12-15, 4:29-31 , 4:41-47

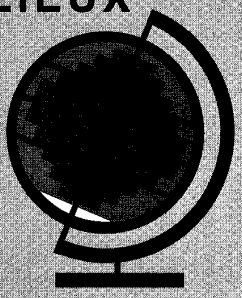
8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
communications channel and	Ligtenberg: 1:16-19, 1:34-42, 5:13-17, 1:22-23,. 3:50-55, 4:41-47, 6:38-43, Fig. 4D, 7:41-45, Appendix A
15.E a processor coupled between said display, memory and communications channel interface,	Ligtenberg: 11:66-12:2, 4:28-34
15.F said processor operative to select said defined data parcel,	Rutledge: 7:48-62 Ligtenberg: 4:28-34
15.G retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	Ligtenberg: 1:16-19, 1:32-42, 10:1-7
15.H render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display, the remote computer comprises:	Rutledge: 7:48-62, 10:10-17 Ligtenberg: 7:7-21 Cooper: Abstract, 4:34-38, 4:48-58, 4:61-5:6, 6:16-24, 6:27-32, 7:12-16, 9:65-10:5
15.I a parcel processing unit that processes a piece of source image data and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
	12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
15.J delivers the defined data parcel to the limited communication bandwidth computer device; and	Rutledge: 1:21-24, 2:62-64, 3:12-15, 4:29-31, 4:41-47, 1:21-23, 3:2-10, 3:54-57; 4:43-47; 6:11-13, 2:62-65; 3:21-23, 4:51-54; 12:4-6 Ligtenberg: 1:16-19, 1:34-42, 2:62-64, 5:13-17, 11:1-2, 1:39-41; 4:40-42; 5:14-17, 11:67-12:2 Cooper: 2:50-53, 4:9-11, 2:27-30; 3:19-22, 4:21-23, 2:50-52; 4:61-5:6
15.K wherein the parcel processing unit further comprises a parcel processing control that processes source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Rutledge: 4:41-47, 5:15-24, 5:50-64; 7:48-62 Ligtenberg: 4:16-17, 2:31-38, 5:39-43, 5:54-65, 6:7-12, 7:18-21, 7:48-62, FIG. 3, 2:31-38, 5:39-43, 5:54-65; 6:7-12 Cooper: 1:29-33, 1:49-64, 2:9-12, 2:19-26, 2:45-50, 5;1-8, 5:65-6:4
15.L wherein series image K_0 being subdivided into a regular	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
array	Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.M wherein each resulting image parcel of the array has a predetermined pixel resolution	Rutledge: Abstract, 5:9-13, 5:14-22; 5:52-64, 8:27-30, 9:15-17 Ligtenberg: Abstract, 2:56-62, 6:52-57, 7:19-21, 7:57-8:11, 11:62-66
15.N wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Rutledge: 4:41-47, 5:15-24, 5:50-64, FIG. 3 Ligtenberg: Abstract, 2:9-22, 2:25-30, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.O resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
15.P said array subdivision being related by a factor of two	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A

8,924,506 patent	**Teachings of U.S. 6,650,998 ("Rutledge") in view of U. S. 5,682,441 ("Ligtenberg") U. S. 6,118,456 ("Cooper") and U.S. 5,940,117 ("Hassan")**
15.Q such that each image parcel being of a fixed byte size.	Rutledge: 2:31-38; 5:15-24, 4:41-47, 5:50-64, FIG. 3 Ligtenberg: 2:56-62; 5:39-43, 2:9-22, 2:25-38, 5:21-27, 5:34-53, 7:19-21, 9:6-20, Appendix A
18. The remote computer of claim 15, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Ligtenberg, Col. 4:25-54 Hassan: Abstract, 1:20-25, 1:44-48, 1:55-57, 2:26-32, Fig. 5

COMPUTING MILIEUX



The Challenges of Mobile Computing

George H. Forman and John Zahorjan
University of Washington

Computers are typically configured for use in a single location. The shift toward mobility and wireless communication is testing the abilities of designers to adapt traditional system structures.

Recent advances in technology have provided portable computers with wireless interfaces that allow networked communication even while a user is mobile. Whereas today's first-generation notebook computers and personal digital assistants (PDAs) are self-contained, networked mobile computers are part of a greater computing infrastructure. Mobile computing — the use of a portable computer capable of wireless networking — will very likely revolutionize the way we use computers.

Wireless networking greatly enhances the utility of a portable computing device. It allows mobile users versatile communication with other people and expedient notification of important events, yet with much more flexibility than with cellular phones or pagers. It also permits continuous access to the services and resources of land-based networks. The combination of networking and mobility will engender new applications and services, such as collaborative software to support impromptu meetings, electronic bulletin boards whose contents adapt to the current viewers, lighting and heating that adjust to the needs of those present, and navigation software to guide users in unfamiliar places and on tours.[1]

The technical challenges that mobile computing must surmount to achieve this potential are hardly trivial, however. Some of the challenges in designing software for mobile computing systems are quite different from those involved in the design of software for today's stationary networked systems. In this article we focus on the issues pertinent to software designers without delving into the lower level details of the hardware realization of mobile computers. We look at some promising approaches under investigation and also consider their limitations.

The many issues to be dealt with stem from three essential properties of mobile computing: communication, mobility, and portability. Of course, special-purpose systems may avoid some design pressures by doing without certain desirable properties. For instance, portability would be less of a concern for mobile computers installed in the dashboards of cars than with hand-held mobile computers. However, we concentrate on the goal of large-scale, hand-held mobile computing as a way to reveal a wide assortment of issues.

0018-9162/ $4.00 © 1994 IEEE

Microsoft Corp. Exhibit 1009

Wireless communication

Mobile computers require wireless network access, although sometimes — when in meeting rooms or at a user's desk — they may remain stationary long enough to be physically attached to the network for a better or cheaper connection.

Wireless networks communicate by modulating radio waves or pulsing infrared light. Wireless communication is linked to the wired network infrastructure by stationary transceivers. The area covered by an individual transceiver's signal is known as a cell. Cell sizes vary widely; for example, an infrared transceiver can cover a small meeting room, a cellular telephone transceiver has a range of a few miles, and a satellite beam can cover an area more than 400 miles in diameter.

Wireless communication faces more obstacles than wired communication because the surrounding environment interacts with the signal, blocking signal paths and introducing noise and echoes. As a result, wireless communication is characterized by lower bandwidths, higher error rates, and more frequent spurious disconnections. These factors can in turn increase communication latency resulting from retransmissions, retransmission time-out delays, error-control protocol processing, and short disconnections.

Mobility can also cause wireless connections to be lost or degraded. Users may travel beyond the coverage of network transceivers or enter areas of high interference. Unlike typical wired networks, the number of devices in a network cell varies dynamically, and large concentrations of mobile users, say, at conventions and public events, may overload network capacity.

The need for wireless communication leads to design challenges in several areas.

Disconnection. Today's computer systems often depend heavily on a network and may cease to function during network failures. For example, distributed file systems may lock up waiting for other servers, and application processes may fail altogether if the network stays down too long.

Network failure is a greater concern in mobile computing than in traditional computing because wireless communication is so susceptible to disconnection. Designers must decide whether to spend available resources on the network, trying to prevent disconnections, or to spend them trying to enable systems to cope with disconnections more gracefully and work around them where possible.

The more autonomous a mobile computer, the better it can tolerate network disconnection. For example, certain applications can reduce communication by running entirely locally on the mobile unit rather than by splitting the application and the user interface across the network. In environments with frequent disconnections, it is better for a mobile device to operate as a stand-alone computer than as a portable terminal.

In some cases, both round-trip latency and short disconnections can be hidden by asynchronous operation. The X11 Window System uses this technique to achieve good performance. With the synchronous remote procedure call paradigm, the client waits for a reply after each request; in asynchronous operation, a client sends multiple requests before asking for acknowledgment. Similarly, prefetching and delayed write-back also decouple the act of communication from the actual time a program consumes or produces data, allowing it to proceed during network disconnections. These techniques, therefore, have the potential to mask some network failures.

The Coda file system provides a good example of how to handle network disconnections, although it is designed for today's notebook computers in which disconnections may be less frequent, more predictable, and longer lasting than in mobile computing.[2] Information from the user's profile helps in keeping the best selection of files in an on-board cache. It is important to cache whole files rather than blocks of files so that entire files can be read during a disconnection. When the network reconnects, Coda attempts to reconcile the cache with the replicated master repository.

With Coda, files can be modified even during disconnections. More conservative file systems disallow this to prevent multiple users from making inconsistent changes. Coda's optimism is justified by studies showing that only rarely are files actually shared in a distributed system; fewer than 1 percent of all writes are followed by a write from a different user.[2] If strong consistency guarantees are needed, clients can ask for them explicitly. Hence, providing flexible consistency semantics can allow greater autonomy.

Of course, not all network disconnec-

Portable terminal versus stand-alone computer

The role of a portable device has considerable latitude within the notion of mobile computing. Is it a terminal or an independent, stand-alone computer? How many purposes should the device serve? Should it incorporate a telephone like the AT&T EO? Should its work environment be that of a general-purpose workstation or of something more restrictive, say, the Apple Newton MessagePad? These design choices greatly affect the way we approach the important issues of communication, mobility, and portability. For example, a portable terminal such as Xerox PARC's Tab[1] is more dependent on a network but less prone to loss of storage media than a stand-alone computer. Such questions require careful consideration when designing software for mobile computing.

Reference

1. M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Comm. ACM*, Vol. 36, No. 7, July 1993, pp. 75-84.

Microsoft Corp. Exhibit 1009

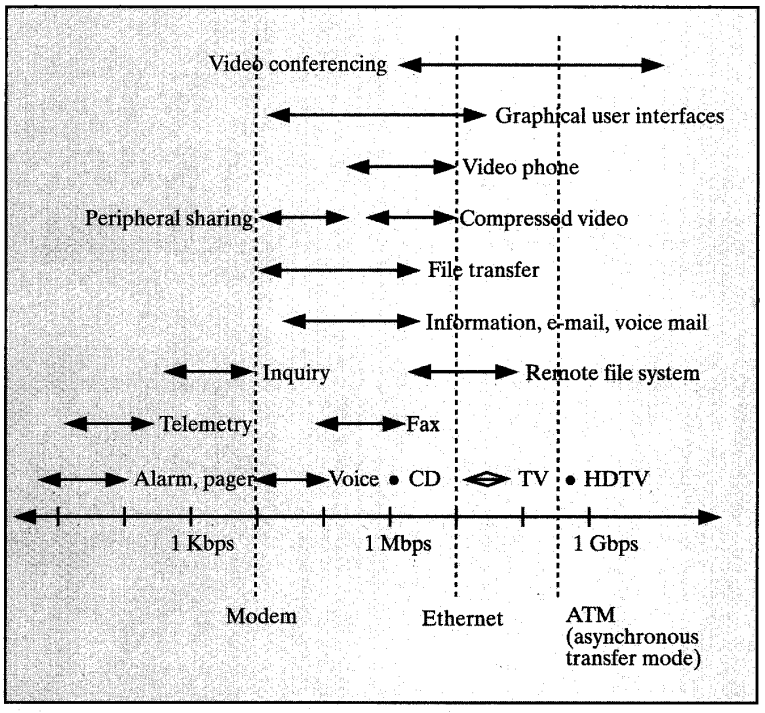


Figure 1. Application bandwidth requirements in bits per second. The vertical dashed lines show the bandwidth capability of certain network technologies. Cellular modems are becoming fast enough for mobile users' everyday information needs, such as e-mail, and someday may be able to support remote file systems.

tions can be masked. In these cases, good user interfaces can help by providing feedback about which operations are unavailable because of network disconnections.

Low bandwidth. Mobile computing designs need to reflect a greater concern for bandwidth consumption and constraints than do designs for stationary computing. Wireless networks deliver lower bandwidth than wired networks: Cutting-edge products for portable wireless communications achieve only 1 megabit per second for infrared communication, 2 Mbps for radio communication, and 9-14 kilobits per second for cellular telephony, while Ethernet provides 10 Mbps, fast Ethernet and FDDI 100 Mbps, and ATM (asynchronous transfer mode) 155 Mbps (see Figure 1). Even nonportable wireless networks, such as the Motorola Altair, barely achieve 5.7 Mbps.

Network bandwidth is divided among the users sharing a cell. The deliverable bandwidth *per user*, therefore, is an important measure of network capacity in addition to the raw transmission bandwidth. But because this measure depends on the size and distribution of a user population, Weiser and others recommend measuring a wireless network's capacity by its bandwidth per cubic meter.[1]

Improving network capacity means installing more wireless cells to service a user population. There are two ways to do this: Overlap cells on different wavelengths, or reduce transmission ranges so that more cells fit in a given area (see Figure 2).

The scalability of the first technique is limited because the electromagnetic spectrum available for public consumption is scarce. This technique is more flexible, however, because it allows (in fact, requires) software to allocate bandwidth among users.

The second technique is generally preferred. It is arguably simpler, reduces power requirements, and may decrease signal corruption because there are fewer objects in the environment to interact with. Also, it involves a hardware tradeoff between bandwidth and coverage area: Transceivers covering less area can achieve higher bandwidths.

Certain software techniques can also help cope with the low bandwidth of wireless links. Modems typically use compression to increase their effective bandwidth, sometimes almost doubling throughput. Because bulk operations are usually more efficient than many short transfers, buffering can improve bandwidth usage by making large requests out of many short ones. Buffering in conjunction with compression can further improve throughput because larger blocks compress better.

Certain software techniques for coping with disconnection can also help cope with low bandwidth. Network usage typically occurs in bursts, and disconnections are similar to bursts in that demand temporarily exceeds available bandwidth. For example, delayed write-back and prefetching use the periods of low network activity to reduce demand at the peaks. Delayed write-back can even reduce overall communication if the data to be transmitted is further mutated or deleted before it is transmitted. Prefetching involves knowing or guessing which files will be needed soon and downloading them over the network before they are demanded.[3] Bad guesses can waste network bandwidth, however.

System performance can be improved by scheduling communication intelligently. When available bandwidth does not satisfy the demand, processes the user is waiting for should be given priority. Backups should be performed only with "leftover" bandwidth. Mail can be trickle fed onto the mobile computer slowly before the user is notified. Although these techniques do not increase effective bandwidth, they improve user satisfaction just the same.

High bandwidth variability. Mobile computing designs also contend with much greater variation in network bandwidth than do traditional designs. Band-

Microsoft Corp. Exhibit 1009

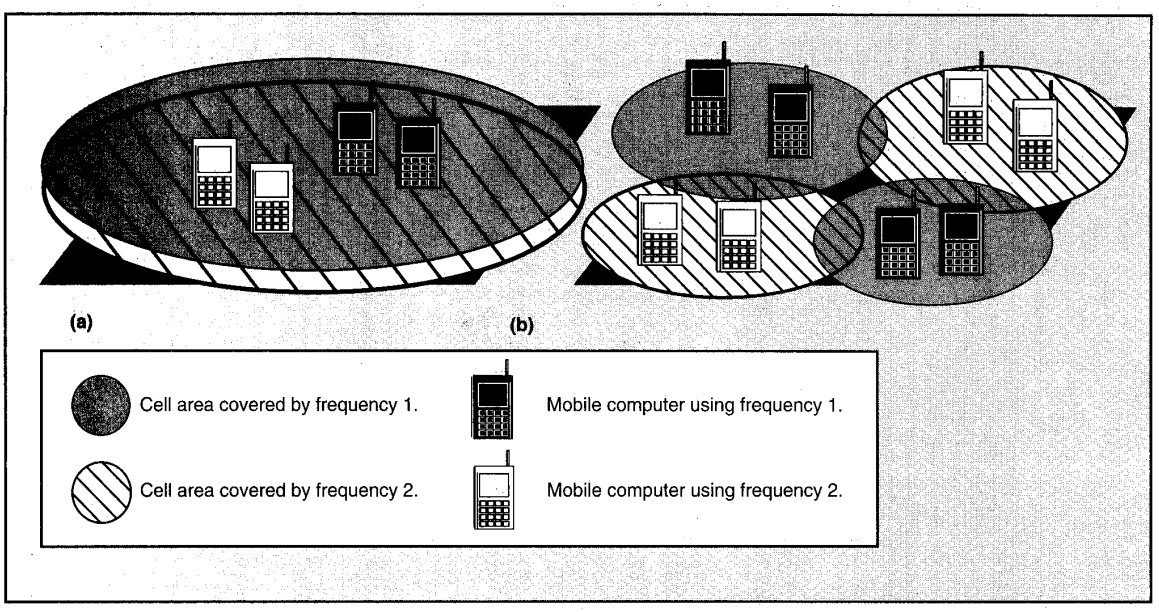


Figure 2. Suppose that a single frequency provides only enough wireless bandwidth for two users. Then two frequencies can support (a) four users with two large coincident cells or (b) eight users with four small noninterfering cells that use the same frequency in nonadjacent cells. The latter scheme requires more transceivers and installation effort but is more scalable and allows higher bandwidth technology and lower transmission power.

width can shift one to four orders of magnitude, depending on whether the system is plugged in or using wireless access. Fluctuant traffic load seldom causes this much variation in available bandwidth on today's networks.

An application can approach this variability in one of three ways: It can assume high-bandwidth connections and operate only while plugged in, it can assume low-bandwidth connections and not take advantage of higher bandwidth when it is available, or it can adapt to currently available resources, providing the user with a variable level of detail or quality. For example, a video-conferencing application could display only the current speaker or all the participants, depending on the available bandwidth. Different choices make sense for different applications.

Heterogeneous network. In contrast to most stationary computers, which stay connected to a single network, mobile computers encounter more heterogeneous network connections in several ways. First, as they leave the range of one network transceiver and switch to another, they may also need to change transmission speeds and protocols. Second, in some situations a mobile computer may have access to several network connections at once, for example, where adjacent cells overlap or where it can be plugged in for concurrent wired access.

Also, mobile computers may need to switch interfaces, for example, when going between indoors and outdoors. Infrared interfaces cannot be used outside because sunlight drowns out the signal. Even with radio frequency transmission, the interface may still need to change access protocols for different networks, for example, when switching from cellular coverage in a city to satellite coverage in the country. This heterogeneity makes mobile networking more complex than traditional networking.

Security risks. Precisely because connection to a wireless link is so easy, the security of wireless communication can be compromised much more easily than that of wired communication, especially if transmission extends over a large area. This increases pressure on mobile computing software designers to include security measures.

Security is further complicated if users are allowed to cross security domains. For example, a hospital may allow patients with mobile computers to use nearby printers but prohibit access to distant printers and resources designated for hospital personnel only.

Secure communication over insecure channels is accomplished by encryption, which can be done in software or, more quickly, by specialized hardware such as the recently proposed Clipper chip. Security depends on a secret encryption key known only to the authorized parties. Managing these keys securely is difficult, but it can be automated by software such as the Massachusetts Institute of Technology's Kerberos.[4]

Kerberos provides secure authentication services, as long as the Kerberos server itself is trusted. It authenticates users without exposing their passwords on the network and generates secret en-

Microsoft Corp. Exhibit 1009

cryption keys that can be selectively shared between mutually suspicious parties. It also allows mobile units to authenticate themselves in domains where they are unknown, thus enhancing the scale of mobility. Methods have also been devised to use Kerberos for authorization control and accounting. Its security is limited, however. For example, the current version is susceptible to off-line password-guessing attacks and to replay attacks for a limited time window.

Mobility

The ability to change locations while connected to the network increases the volatility of some information. Certain data considered static for stationary computing becomes dynamic for mobile computing. For example, a stationary computer can be configured statically to prefer the nearest server, but a mobile computer needs a mechanism for determining which server to use.

As volatility increases, cost-benefit trade-off points shift, calling for appropriate modifications in the design. For example, a highly volatile data object has fewer uses per modification. For such objects it makes little sense to cache the data. As another example, consider static information, which is often managed by hand; to handle higher rates of change, automated methods are required. However, even where such methods exist, they may be ill-suited for the dynamism of mobile computing.

Mobility introduces several problems: A mobile computer's network address changes dynamically, its current location affects configuration parameters as well as answers to user queries, and the communication path grows as it wanders away from a nearby server.

Address migration. As people move, their mobile computers will use different network access points, or "addresses." Today's networking is not designed for dynamically changing addresses. Active network connections usually cannot be moved to a new address. Once an address for a host name is known to a system, it is typically cached with a long expiration time and with no way to invalidate out-of-date entries. In the Internet Protocol, for example, a host IP name is inextricably bound with its network address; moving to a new location means acquiring a new IP name. Human intervention is commonly required to coordinate the use of addresses.

To communicate with a mobile computer, messages must be sent to its most recent address. Four basic mechanisms determine a mobile computer's current address: broadcast,[5,6] central services,[7] home base,[8] and forwarding pointers.[5]

As people move, their mobile computers will use different network access points, or "addresses."

These are the building blocks of the current proposals for "mobile-IP" schemes.

Selective broadcast. With the broadcast method, a message is sent to all network cells asking the mobile computer sought to reply with its current address. This becomes too expensive for frequent use in a large network, but if the mobile computer is known to be in some small set of cells, selectively broadcasting in just those cells is workable. Hence, the methods described below can use selective broadcast to obtain the current address when only approximate location information is known. For example, a slightly out-of-date cell address may suffice if adjacent cells are known.

Central services. With the central service method, the current address for each mobile computer is maintained in a logically centralized database. Each time a mobile computer changes its address, it sends a message to update the database. Even with the database's centralized location, the common techniques of distribution, replication, and caching can be used to improve availability and response time.

Home bases. The home base method is essentially the limiting case of distributing a central service; that is, the location of a given mobile computer is known by a single server. This aggressive distribution without replication can lead to low availability of information. For example, if a home base is down or inaccessible, the mobile computers it tracks cannot be contacted. If users sometimes change home bases, the address migration problem arises again, albeit with much lower volatility.

Forwarding pointers. With the forwarding pointer method, each time a mobile computer changes its address, a copy of the new address is deposited at the old location. Each message is forwarded along the chain of pointers until it reaches the mobile computer. To avoid the inefficient routing that can result from long chains, pointers at message forwarders can be updated gradually to reflect more recent addresses.

Although the forwarding pointer method is among the fastest, it is prone to failures anywhere along the trail of pointers, and in its simplest form it does not allow forwarding pointers to be deleted unless all possible message sources have been updated. Hence, forwarding pointers are often used only to speed the common case, and another method is used to fall back on for failures and to allow reclamation of old pointers.

The forwarding pointer method requires an active entity at the old address to receive and forward messages. This does not fit standard networking models, where either a network address is a passive entity, such as an Ethernet cable, or it's specific to the mobile computer, which cannot remain to forward its own messages. This mismatch introduces subtle difficulties in implementing forwarding efficiently (for example, with intracell traffic or when multiple gateways are attached to a network address).

Location-dependent information. Because traditional computers do not move,

Microsoft Corp. Exhibit 1009

information that depends on location, such as the local name server, available printers, and the time zone, is typically configured statically. One challenge for mobile computing is to factor out this information intelligently and provide mechanisms for obtaining configuration data appropriate to each location. Additionally, a mobile computer carried with a user is likely to be used in a wide variety of administrative domains. Dealing with the multitude of conventions that current computing systems rely on is another challenge to building mobile systems.

Besides this dynamic configuration problem, mobile computers need access to more location-sensitive information than stationary computers do. If they are to serve as guides in places unfamiliar to their users, mobile computers may need to answer queries such as "where is the fiction section (in this particular library)?" or "where is the nearest open gas station heading north?"

Queries of this sort require static location information about the world. Other information needs can be even more complex: Badrinath and Imielinski are studying a related class of queries that depend on the dynamic locations of other mobile objects, for example, the location of the nearest taxi.[6]

Privacy. Answering dynamic location queries requires knowing the location of another mobile user. In some cases this may be sensitive information, more so if given at a fine resolution. Even where it is not particularly sensitive, such information should be protected against misuse; for example, we do not want a burglar to be able to determine when the inhabitants of a house are far away.

Privacy can be ensured by denying users the ability to know another's location. The challenge for mobile computing is to allow more flexible access to this information without violating privacy. Legitimate uses of location information include contacting colleagues, routing telephone calls, logging meetings in personal diaries, and tailoring the content of electronic announcement displays to the current viewers.[1]

Migrating locality. Mobile computing engenders a new kind of locality that migrates as users move. Even if a mobile computer is equipped to find the nearest server for a given service, over time migration may alter this condition. Because the physical distance between two points does not necessarily reflect the network distance, the communication path can grow disproportionately to actual movement. For example, a small movement can result in a much longer path when crossing network administrative boundaries, and a longer network path means communication traverses more intermediaries, resulting in longer latency and greater risk of disconnection. A longer communication path also consumes more network capacity, even though the bandwidth between the mobile unit and the server may not degrade.

Mobile computers need access to more location-sensitive information than do stationary computers.

To avoid these disadvantages, service connections may be dynamically transferred to servers that are closer.[9] When many mobile units converge, during meetings, for example, load-balancing concerns may outweigh the importance of communication locality.

Portability

Today's desktop computers are not meant to be carried, so designers take a liberal approach to space, power, cabling, and heat dissipation. In contrast, designers of hand-held mobile computers should strive for the properties of a wristwatch: small, light, durable, operational under wide environmental conditions, and requiring minimal power usage for long battery life. Concessions can be made in each of these areas to enhance functionality, but ultimately the user must receive value that exceeds the trouble of carrying the device. Similarly, any specialized hardware to offload such tasks as data compression or encryption from the CPU should justify its consumption of power and space.

Below, we describe the design pressures caused by portability constraints. These pressures are evident in the designs of the recent PDA products listed in Table 1.

Low power. Batteries are the largest single source of weight in a portable computer. While reducing battery weight is important, too small a battery can undermine the value of portability by causing users to recharge frequently, carry spare batteries, or use their mobile computers less. Minimizing power consumption can improve portability by reducing battery weight and lengthening the life of a charge.

Power consumption of dynamic components is proportional to CV^2F, where C is the capacitance of the circuit, V is the voltage swing, and F is the clock frequency. This function suggests three ways to save power:

(1) Capacitance can be reduced by greater levels of VLSI integration and multichip module technology.
(2) Voltage can be reduced by redesigning chips to operate at lower voltages. Historically, chips operate at 5 volts, but some, like those in the Apple MessagePad, save power by operating at 3 volts. Manufacturers are rapidly developing a line of low-power chip sets for 2.5V and 3.3V operation.
(3) Clock frequency can be reduced, thereby trading computational speed for power savings. PDA products have adopted this concession, as shown in Table 1. In some notebook computers, the clock frequency can be changed dynamically, providing a flexible trade-off; for example, the Sharp PC 6785 can save power by dynamically shifting its clock from 25 MHz to 10 MHz or even 5 MHz, as shown in Table 2. To

Microsoft Corp. Exhibit 1009

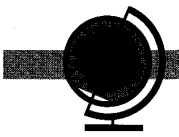


Table 1. Characteristics of personal digital assistant products and the AT&T EO tablet computer. Each has a pen interface and a black-and-white reflective LCD screen. The portable PC is included for comparison.

Product	RAM	MHz	CPU	Batteries (No. hours)	Batteries (type)	Weight (lbs.)	Display (pixels)	Display (sq. in.)
Amstrad Pen Pad PDA600	128 Kbytes	20	Z-80	40	3 AA's	0.9	240 × 320	10.4
Apple Newton MessagePad	640 Kbytes	20	ARM*	6-8	4 AAA's	0.9	240 × 336	11.2
Apple Newton MessagePad 110	1 Mbyte	20	ARM*	50	4 AA's	1.25	240 × 320	11.8
Casio Z-7000 PDA	1 Mbyte	7.4	8086	100	3 AA's	1.0	320 × 256	12.4
Sharp Expert Pad	640 Kbytes	20	ARM*	20	4 AAA's	0.9	240 × 336	11.2
Tandy Z-550 Zoomer PDA	1 Mbyte	8	8086	100	3 AA's	1.0	320 × 256	12.4
AT&T EO 440 Personal Communicator	4-12 Mbytes	20	Hobbit	1-6	NiCad	2.2	640 × 480	25.7
Portable PC	4-16 Mbytes	33-66	486	1-6	NiCad	5-10	640 × 480 to 1,024 × 768 (color)	40

*Advanced RISC microprocessor

retain more computational power at lower frequencies, processors are being designed that perform more work on each clock cycle.[10]

Power can be conserved not only by the design but also by efficient operation. Power management software can power down individual components when they are idle, for example, spinning down the internal disk or turning off screen lighting. Recently, Li et al. determined that for today's notebook computing it is worthwhile to spin down the internal disk drive after it has been idle for just a few seconds.[11]

Applications can conserve power by reducing their appetite for computation, communication, and memory, and by performing their periodic operations infrequently to amortize the start-up overhead. Since radio modem transmission typically requires about 10 times as much power as reception, power can be saved by trading transmission for reception. For example, base stations might periodically broadcast information that otherwise would have to be requested frequently. In this way, mobile computers can obtain this information without expending power to transmit a request.

The potential savings of these techniques can be evaluated using Tables 2 and 3, which break down power consumption in notebook computers by component and subsystem, respectively. Although screen lighting consumes a large amount of power, it has been found to greatly improve readability; for example, on EO models it enhances contrast from 6:1 to 13:1. Nevertheless, PDA products have elected to omit screen lighting in favor of longer battery life.

Risks to data. Making computers portable increases the risk of physical damage, unauthorized access, loss, and theft. Breaches of privacy or total loss of data become more likely. These risks can be reduced by minimizing the essential data kept on board. Obviously, a mobile device that serves only as a portable terminal is less prone to data loss than a stand-alone computer. This is the approach taken for Xerox PARC's Tabs and the portable multimedia terminal project at the University of California, Berkeley.[10]

To help prevent unauthorized disclosure of information, data stored on disks and removable memory cards can be encrypted. For this to be effective, users

Microsoft Corp. Exhibit 1009

must not leave authenticated sessions (logins) unattended.

Keeping a copy that does not reside on the portable unit can safeguard against data loss. However, users often neglect to make backup copies, and even when they do, data modified between backups is not protected. With the addition of wireless networks to portable computers, new or modified data can be copied immediately to secure, remote media. This can be accomplished with replicated file systems such as Echo and Coda.[2]

Small user interface. Size constraints on a portable computer require a small user interface. Desktop windowing environments may be sufficient for today's notebook computers, but for smaller, more portable devices, current windowing technology is inadequate. On small displays it is impractical to have several windows open at a time regardless of screen resolution, and it can be difficult to locate windows or icons deeply stacked one on another. Also, window title bars and borders either consume significant portions of screen space or, if reduced, become difficult to operate with the pointing device.

Duchamp, Feiner, and Maguire investigated the use of head-mounted virtual reality displays for portable computers.[9] As the user's head turns, the image displayed to the eye shifts to give the sensation of a surrounding screen. This effectively increases the screen area available for windowing systems; however, wearing head gear is cumbersome, resolution is low (one-tenth that of conventional displays), eyes succumb to fatigue, and dim lighting is required.

Buttons versus analog input. The shortage of surface area on a small computer leads designers to sacrifice buttons in favor of analog input devices for communicating user commands. These communication mechanisms include handwriting recognition, gesture recognition, and voice recognition. Although on average handwriting is about three times slower than typing, it allows the keyboard to be eliminated, thus reducing size and improving durability. This approach has been adopted by all the PDA products listed in Table 1.

Handwriting recognition rates for high-end systems are typically 96-98 percent accurate when trained to a specific user.[12] With context information, recognition rates can be enhanced effectively to 100 percent, but context constraints do not help for all kinds of input, for example, when entering words that are not in the on-line dictionary. The Apple Newton's handwriting recognition, while among the best of the PDAs, is nevertheless reputedly a source of frustration. Recognizing a user's intention in a general setting is inherently hard because the interpretation of pen strokes is ambiguous. For example, a user drawing a circle may intend to select an object or an area, or write a zero, a degree sign, or the letter *o*.

Speech generation and rec-

Table 2. Power consumption of portable-computer components and accessories.*

Device	Power (watts)
Base system (2 Mbytes, 25-MHz CPU)	3.650
Base system (2 Mbytes, 10-MHz CPU)	3.150
Base system (2 Mbytes, 5-MHz CPU)	2.800
Screen backlight	1.425
Hard drive motor	1.100
Math coprocessor	0.650
Floppy drive	0.500
External keyboard	0.490
LCD screen	0.315
Hard drive active (head seeks)	0.125
IC card slot	0.100
Additional memory (per Mbyte)	0.050
Parallel port	0.035
Serial port	0.030
Accessories	
1.8-inch PCMCIA hard drive	0.7-3.0
Cellular telephone (active)	5.400
Cellular telephone (standby)	0.300
Infrared network, 1 Mbit per second**	0.250
PCMCIA modem, 14,400 bits per second	1.365
PCMCIA modem, 9,600 bits per second	0.625
PCMCIA modem, 2,400 bits per second	0.565
Global positioning receiver**	0.670

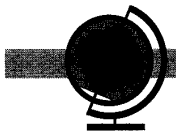
*Data for computer components was derived from the Sharp PC 6785 manual; data for accessories was obtained from the manufacturers.
**Estimate for soon-to-be-released product.

Table 3. Power consumption of the major components in a portable computer.*

System	Power (percent)
Display edge-light	35
CPU/memory	31
Hard disk	10
Floppy disk	8
Display	5
Keyboard	1

*Data was obtained from the Compaq LTE 386/s20 manual.

Microsoft Corp. Exhibit 1009



ognition seem an ideal user interface for a mobile computer in that they require no surface area and allow hands-free and even eye-free operation. The voice-commanded VCR programmer by Voice Powered Technology demonstrates the feasibility of speech input to a hand-held device for a narrow domain. The Sphinx research project at Carnegie Mellon University has reported speaker-independent recognition rates of nearly 96 percent, and 98 percent for speaker-trained recognition. However, general-purpose speech input and output places substantial storage and processing demands on a mobile device. Also, speech may often be inappropriate: It disturbs others in quiet environments, it cannot be recognized clearly in noisy environments, and it can compromise privacy. Finally, because of its sequential nature, speech is ill-suited for skimming data.

Pointing devices. The mouse is the standard pointing device for desktop computers, but it doesn't suit mobile computers. Pens have become the standard input device for PDAs because of their ease of use while mobile, their versatility, and their ability to supplant the keyboard.

Switching to pens requires changing both the user interface and the software interface because a mouse and a pen are really quite different.[9] Users can jump to absolute screen positions and enter path information more easily with a pen than with a mouse, and it is nearly impossible to write with a mouse. Pen-positioning resolution on current tablet computers is several times more accurate than screen resolution; for example, pen resolution on the EO is 0.10 mm, while screen resolution is 0.23-0.30 mm. Also, parallax between the pen tip and the screen image can mislead when pointing; with a mouse there is no parallax because the mouse cursor provides feedback in the image plane. Finally, the mouse cursor obscures much less of the screen than the user's hand does when writing with a pen.

Small storage capacity. Storage space on a portable computer is limited by physical size and power requirements. Traditionally, disks provide large amounts of nonvolatile storage. In a mobile computer, however, disk drives are a liability. They consume more power than memory chips, except when off line, and they may not really be nonvolatile when subject to the indelicate treatment a portable device receives. Hence, none of the PDA products have disk drives.

Coping with limited storage is not a new problem. Solutions include compressing files automatically, accessing remote storage over the network, sharing code libraries, and compressing virtual memory pages. Although today's networked computers have had great success with distributed file systems and remote paging, mobile computers that regularly encounter network disconnections are less capable of relying on a network.

A novel approach to reducing the size of program code is to interpret script languages instead of executing compiled object codes, which are typically many times larger than the source code. This approach is embodied by General Magic's Telescript and Apple Technology Group's Dylan and NewtonScript. An equally important goal of such languages is to enhance portability by supporting a common programming model across different machines.

Mobile computing is a technology that enables access to digital resources at any time, from any location. From a narrow viewpoint, mobile computing represents a convenient addition to wire-based local area distributed systems. Taken more broadly, mobile computing represents the elimination of time-and-place restrictions imposed by desktop computers and wired networks.

In forecasting the impact of mobile technology, we would do well to observe recent trends in the use of the wired infrastructure, in particular, the Internet. In the past year, the advent of convenient mechanisms for browsing Internet resources has engendered an explosive growth in the use of those resources. The ability to access them at all times through mobile computing will allow their use to be integrated into all aspects of life and will accelerate the demand for network services. The challenge for computing designers is to adapt the system structures that have worked well for traditional computing so that mobile computing can be integrated as well. ■

Acknowledgments

Support for this work was provided in part by the National Science Foundation (grants CCR-9123308 and CCR-9200832), Tektronix Inc. (a graduate fellowship), the Washington Technology Center, and Digital Equipment Corp. (Systems Research Center and External Research Program). We thank Robert Bedichek, Brian Bershad, Blake Hannaford, Marc Fiuczynski, Brian Pinkerton, and Stefan Savage for helpful pointers and clarifying discussions that significantly improved this article.

References

1. M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Comm. ACM*, Vol. 36, No. 7, July 1993, pp. 75-84.

2. J.J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Trans. Computer Systems*, Vol. 10, No. 1, Feb. 1992, pp. 3-25.

3. C.D. Tait and D. Duchamp, "Detection and Exploitation of File Working Sets," *Proc. 11th Int'l Conf. Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., Order No. 2144, 1991, pp. 2-9.

4. B.C. Neuman, "Protection and Security Issues for Future Systems," in *Workshop on Operating Systems of the 90s and Beyond*, Lecture Notes in Computer Science #563, Springer-Verlag, New York, 1991, pp. 184-201.

5. J. Ioannisdis, D. Duchamp, and G.Q. Maguire Jr., "IP-Based Protocols for Mobile Internetworking," *Proc. SIGComm 91 Symp.*, ACM, New York, 1991, pp. 235-245.

6. T. Imielinski and B.R. Badrinath, "Data Management for Mobile Computing," *SIGMOD Record*, Vol. 22, No. 1, Mar. 1993, pp. 34-39.

7. C. Ma, "On Building Very Large Naming Systems," *Proc. Fifth ACM SIGOPS Workshop Models and Paradigms for Distributed Systems Structuring*, ACM, New York, 1992, 5 pp.

8. F. Teraoka and M. Tokoro, "Host Migration Transparency in IP Networks: The VIP Approach," *Computer Comm. Rev.*, Vol. 23, No. 1, Jan. 1993, pp. 45-65.

Microsoft Corp. Exhibit 1009

9. D. Duchamp, S.K. Feiner, and G.Q. Maguire Jr., "Software Technology for Wireless Mobile Computing," *IEEE Network Magazine*, Vol. 5, No. 6, Nov. 1991, pp. 12-18.

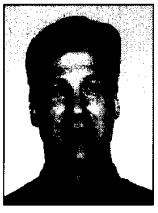
10. A. Chandrakasan, S. Sheng, and R.W. Brodersen, "Design Considerations for a Future Portable Multimedia Terminal," in *Third-Generation Wireless Information Networks*, S. Nanda and D.J. Goodman, eds., Kluwer Academic Publishers, Hingham, Mass., 1992, pp. 75-97.

11. K. Li et al., "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," tech. report, Computer Science Division, University of California, Berkeley, Calif., 1993.

12. C.C. Tappert, C.Y. Suen, and T. Wakahara, "On-Line Handwriting Recognition — A Survey," *Proc. Ninth Int'l Conf. Pattern Recognition*, Vol. 2, IEEE CS Press, Los Alamitos, Calif., Order No. 878, 1988, pp. 1,123-1,132.

George Forman is a PhD candidate in the Department of Computer Science and Engineering at the University of Washington. His research interests include mobile computing and compilers for parallel computers.

Forman received his BA in mathematics from Pomona College, California, in 1988. The following year he received a Fulbright fellowship for study at the Swiss Federal Institute of Technology, Zurich. He is a member of Sigma Xi and Phi Beta Kappa.



John Zahorjan is a professor of computer science and engineering at the University of Washington. His research interests include performance modeling and experimental evaluations, as well as issues in mobile computing, runtime support for parallel computing, and resource scheduling for continuous-media applications.

Zahorjan received an ScB in applied mathematics from Brown University in 1975 and MSc and PhD degrees in computer science from the University of Toronto in 1976 and 1980, respectively. In 1984 he received a Presidential Young Investigator Award from the National Science Foundation. He is a member of the IEEE Computer Society.

The authors can be contacted at the Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195, e-mail {forman, zahorjan}@cs.washington.edu.

April 1994

RIDT'94

RASTER IMAGING & DIGITAL TYPOGRAPHY

Darmstadt, Germany
11-14 April, 1994

Sessions

Font modeling, parametrisation of fonts
Variables width splines, digital halftoning
Readibility, symbols for displays
Intelligent outline-fonts
Optical font recognition
Constraints, string matching techniques

Tutorials

Desktop colour reproduction, colorimetry
Modeling human vision
Typography and Multimedia

Guest speakers

Chuck Bigelow

Hermann Zapf

Information
Jacques Andre
INRIA-Rennes, campus de Beaulieu
F-35042 Rennes cedex, France
Fax: +33 99 38 38 32
email: ridt94@irisa.fr

A Network Architecture for Mobile Computing*

Kevin Brown
Dept. of Computer Science
Univ. of South Carolina
Columbia, SC 29208
kbrown@cs.sc.edu

Suresh Singh
Dept. of Computer Science
Univ. of South Carolina
Columbia, SC 29208
singh@cs.sc.edu

September 7, 1995

Submitted for publication

Abstract

In this paper we report on an ongoing project to design and build the network and transport layers for mobile networking. The network architecture used is unique in that it separates the mobile network(s) from fixed networks and provides connectivity between the two via special gateways. These gateways provide QOS guarantees to mobile users for all their open connections. We provide summaries of all the protocols we are implementing (or have implemented) and discuss possible improvements.

Technical area : Wireless Networks and Protocols

Corresponding author: Suresh Singh

Telephone: (803)777-2596
Fax: (803)777-3767

1 Introduction

Mobile computing is an emerging new computing environment incorporating both wireless and high-speed networking technologies. Users equipped with *personal digital assistants* or PDAs

*Funding for this work was provided by the NSF under grant number NCR-9410357.

Microsoft Corp. Exhibit 1009

(palm-top computers) will have access to a wide variety of services that will be made available over national and international communication networks. Mobile users will be able to access their data and other services such as electronic mail, electronic news, yellow pages, map services, electronic banking and videotelephony services while on the move. To receive these services, mobile users will be connected to fixed networks via wireless networks (or mobile networks).

The **goal** of this paper is to present a comprehensive solution to the problem of wireless networking for mobile computing. We propose a mobile network architecture, network layer design and transport protocols that, we believe, will make it possible to offer all of the above services in an integrated manner. Such a system is currently being built at University of S. Carolina and, in this paper, we discuss the design in detail.

1.1 Challenges in Mobile Networking

Providing the type of services discussed above to mobile users requires high data rates on the wireless link and several authors (see for example Goodman[7, 8] Joseph[11]) have proposed an average data rate of between 1–2Mbps per mobile user. The third generation cellular system being developed in Europe (UMTS – Universal Mobile Communication System), for instance, also propose bandwidth in the same range (see DaSilva[4]). In order to support such high data rates, a *microcellular* network architecture has been proposed, see Goodman[7] and Duchamp[5]. Here, a geographical region such as a campus is divided into *microcells* with a diameter of the order of hundreds of meters. All mobile users within a microcell communicate with a central host machine within that cell who serves as a gateway to the wired networks; this machine is called a *mobile support station* (MSS).

What are some of the *networking issues* we need to address in order to provide the different types of service discussed above? Two broad issues that need to be considered are the following:

- Design of an efficient network architecture to support mobility and corresponding network layer protocols. The problems here include,
  - Tracking mobile users as they roam,
  - Routing messages and forwarding them to the current location of the mobile user,
  - Providing flow-control and buffering for open connections.
- Developing transport layer protocols that mesh easily with protocols that will be made available over high-speed networks. Some requirements here include,
  - Developing mobile network analogues of commonly used protocols such as TCP and UDP,
  - Developing protocols that support real-time applications such as voice and video or on-line data services,

Microsoft Corp. Exhibit 1009

– Maintaining quality of service guarantees for applications even in the presence of mobility.

Some of the problems mentioned above have been addressed by other researchers as we discuss below.

Routing in Mobile Networks: In mobile networks, since the hosts are mobile, routing is a problem. Ioanidis[9] proposes a solution called the **IPIP** ("IP-within-IP") protocol. Here every MH has a unique IP address called its 'home address'. To deliver a packet to a remote MH, the source MSS first broadcasts an ARP to all other MSS nodes to locate the MH. Eventually some MSS responds. The source MSS then encapsulates each packet from the source MH within another packet containing the IP address of the MSS in whose cell the destination MH is located. Upon receiving this packet the destination MSS extracts the original packet and attempts to deliver it to the destination MH. If the MH has moved away, the destination MSS attempts to locate it by broadcasting an ARP request. As discussed in Teraoka[15] this method is not easily scalable.

Teraoka[15] proposes a more flexible solution to the problem called – the Virtual Internet Protocol or **VIP**. Here every host has a *virtual network address* (VIP address) that is unchanging. In addition, hosts have associated *physical network addresses* (traditional IP addresses) that may change as the host moves around. At the transport layer, the target node is always specified by its VIP address only. The address resolution from the VIP address to the IP address takes place at either the network layer of the same machine or at a gateway. Both, the host machines and gateways, maintain a cache of VIP to IP mappings with associated time stamps. This information is in the form of a table and is called AMT (or *address mapping table*). Routing is achieved by referring to these AMT tables.

Transport Protocols for Mobile Networks: Since mobile hosts will expect the same services that are offered to fixed hosts, it is necessary to implement transport services in the mobile domain that are similar to those offered in the fixed networks. TCP is one such protocol. If we use TCP without any modification in mobile networks we have a serious problem of efficiency. This is because in TCP the sender begins retransmission of packets if they are not acknowledged within a short amount of time (hundreds of milliseconds). In a mobile environment, as a user moves between cells, there is a brief blackout period while the mobile unit performs a 'handshake' with the new MSS. These blackout periods may also be caused by physical obstacles in the environment that interfere with radio signals. These periods can be of the order of 1 second thus delaying the transmission of acknowledgements for packets received. This results in the TCP sender timing out and retransmitting the unacknowledged packets thus greatly reducing the efficiency of the connection. A solution to this problem is the I-TCP protocol (Indirect-TCP), implemented at Rutgers University as part of the DataMan project (see Barke[1]), that provides efficient reliable communication for the wireless environment. A benefit of the above implementation is that it allows mobile hosts to be connected over the Internet. We examine the I-TCP protocol in more

Microsoft Corp. Exhibit 1009

detail in section 3.2.1 where we compare it against our own proposal.

1.2 Summary of Paper

We are currently building a mobile network from the ground up and this paper discusses our design and initial experience. Specifically, we propose a design for the network layer and transport layers for 3rd generation wireless systems and provide arguments in support of this design. The implementation of the protocol stack is been done under Unix (we use NetBSD) running on Pentium PCs.

- In section 2 we discuss our architecture for the mobile subnetwork that, we feel, best addresses the various issues raised in section 1.1.
- Our network layer design is presented in section 3.1. The sketch of our transport layer is presented in section 3.2. We also address management and control questions specific to the mobile environment (e.g., feedback of dynamic bandwidth changes to applications, etc.). Special protocols for notification applications (e.g., pager service) and continuous media are also incorporated.

2 Overview of our Proposed Architecture

The solutions discussed in section 1.1 use a microcellular architecture where the base station for each cell is a node on the internet. The solutions proposed for routing and TCP implementations thus assume that the underlying subnetwork is a datagram network. We believe that this assumption has only one justification – compatibility with existing technology – which, in our view, is insufficient. Some of the problems with this approach are,

- The base stations (or MSSs) are responsible for tracking mobile users and forwarding packets to their new locations. This adds to the *cost* and *complexity* of the base stations and since we expect cell sizes to be small (100m), in order to accommodate high-bandwidth applications, the total cost of a mobile network will be very high.
- As a user roams between cells, the bandwidth available in each cell may also vary. If the user has open connections, it will have to renegotiate QOS parameters frequently. This is clearly an undesirable situation.
- Cell latency times (staying time in a cell) are typically small (several seconds). This exacerbates both of the above problems.

Our architecture is discussed in detail in Singh[14]. We summarize the main points in this section. Our architecture may be viewed as a three-level hierarchy (see Figure 1). At the lowest level are the mobile hosts (MH) who communicate with MSS nodes in each cell. Several MSSs are

Microsoft Corp. Exhibit 1009

controlled by a machine called the Supervisor Host (SH). The SH is connected to the wired network and it handles most of the routing and other protocol details for the mobile users. In addition it maintains connections for mobile users, handles flow-control and is responsible for maintaining the negotiated quality of service. A single SH may thus control all MSS nodes within a small building. Our architecture separates the mobile network from the high-speed wired network and provides connectivity between the two via supervisor hosts (SHs) who serve the function of a **gateway**.

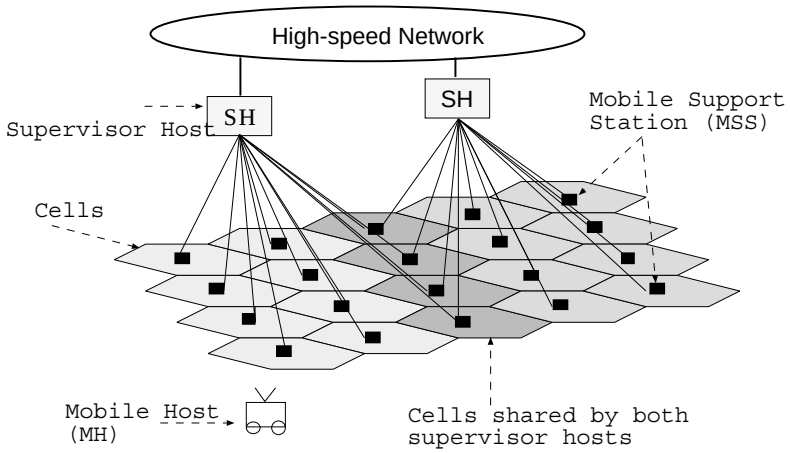


Figure 1: Proposed Architecture.

A mobile user may set up connections where the other end-point is either another mobile user or a fixed host (e.g., a service-provider) in the fixed network. In either case the connection is managed by the current SH of the mobile host(s) (see Figure 2). The reason for splitting the connection between the MH and the service-provider is to shield fixed nodes from the idiosyncrasies of the mobile environment. Thus, the service-provider sets up a connection with the SH assuming the SH is the other end-point of the connection. The SH sets up another connection to the MH. Thus for every **MH – service-provider** connection the QOS parameters are defined *separately* for the **MH – SH** part and for the **SH – service-provider** part of the connection. Connections between two MHs are broken in three (if the two MHs are controlled by different SHs) – **MH1 – SH1**, **SH1 – SH2** and **SH2 – MH2**. Note that the SH–SH part of any connection is established over the fixed network. The benefits of our architecture, thus, are:

- The MSSs are simple and cheap devices because they merely serve as a point of attachment for MHs.
- Since several MSSs are controlled by a single SH, the roaming MH remains within the domain of the same SH for long time periods[1]. This makes it easier to guarantee QOS

[1] e.g., a MH may roam frequently between rooms in an office building but remain for many hours within the building – each room is a cell and all cells in the building are controlled by one SH.

Microsoft Corp. Exhibit 1009

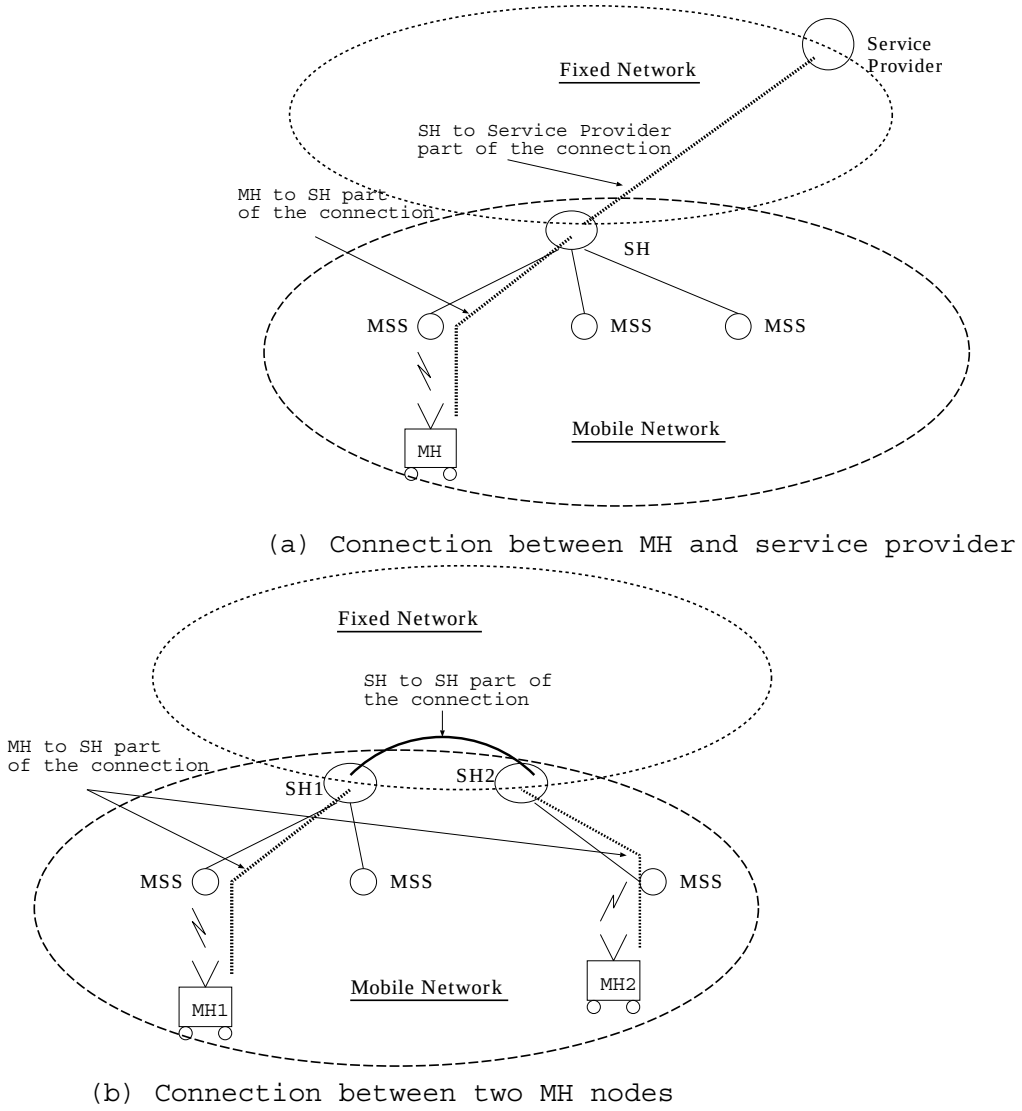


Figure 2: Connections for MHs are managed by SHs.

parameters for MH connections.

3 Network and Transport Layer Protocols

Our view of the mobile network in relation to fixed wired networks is illustrated in Figure 3. The mobile network is actually composed of many sub-networks each of which is connected to the fixed networks via a SH node. SH nodes communicate with one another over the fixed network. Each SH controls several MSS nodes. Physical communication between a SH node and its MSS nodes is accomplished either over a dedicated network consisting of dedicated wiring (perhaps several MSSs are connected via twisted-pair to a hub and several hubs are connected directly to the SH) or over the fixed network itself – i.e., the MSS nodes are regular hosts on the fixed network and

Microsoft Corp. Exhibit 1009

can be addressed with IP addresses. The latter is a cheaper solution and provides a *migration* path to having dedicated mobile networks.

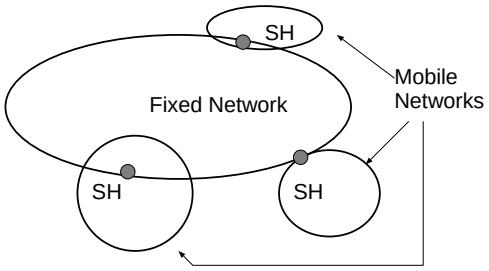


Figure 3: Relationship of mobile networks to fixed networks.

3.1 Network Layer Design

3.1.1 Routing and Tracking

Recall that all connections set up by a MH pass through its SH. For instance, a connection between MH M_1 and M_2 located within the same mobile network (i.e., same SH S) is set up as M_1–S–M_2. If the two MHs are in different mobile networks S_1 and S_2 the connection is M_1–S_1–S_2–M_2. The S_1–S_2 portion of the connection passes over the fixed network. A connection from M_1 to a fixed host F is set up as, M_1–S–F [2]. In all of these cases, notice that routing consists of two components – routing within a given mobile network from the SH to the MH and routing over the fixed network between SHs or between SHs and fixed hosts. Let us consider each of these two components separately.

In our design we implement *virtual circuits* at the network layer *in each mobile sub-network*. This means that the network layer will deliver all packets *in order* to the current MSS of the MH. Thus even if the MH moves between cells, so long as the MH remains within the same mobile sub-network, all packets will be delivered to its MSS in order. It is important to observe that the network layer *does not* guarantee delivery of packets *to the* MH; it only guarantees delivery to the MSS. This is because the wireless link is very unreliable and error recovery is best left to the transport layer which is responsible for implementing service guarantees. We discuss this point in greater detail in section 3.2. Moreover, if the MH moves into the domain of another SH, there are no guarantees made regarding the delivery of packets in transit.

To route within one mobile sub-network (i.e., within the domain of one SH), the network layer at the SH maintains a location table consisting of entries for each MH currently within its domain and the location of the MH (i.e., the identity of its MSS). This table is updated via control messages passed between the MSS and SH every time a MH moves. If the MSS nodes are

[2]The reason for routing all connections through a SH is that the SH can provide network level buffering and flow-control. If we set up a connection directly between two MHs, for instance, flow-control and retransmission of lost packets will require an exchange of control messages between the MHs consuming scarce wireless bandwidth.

Microsoft Corp. Exhibit 1009

connected via dedicated links to the SH, there is no need for them to have IP numbers. The SH simply transmits on the appropriate port. If the MSS nodes are nodes on the fixed network, then the SH needs to route messages to the MSS nodes on the fixed network. In our implementation all MH nodes have unique IP addresses (with some 'home' network as part of the address). Here the SH nodes route messages to the MH by using the IP *loose source routing option* (following the work of Johnson[10]). The destination address in the header is set to the IP address of the MSS and the MH IP address is contained as the first IP address in the option part of the header. The MSS examines the datagram and delivers it to the MH (if it is present in its cell). If it has moved away, the MSS discards the datagram.

To implement reliable delivery (in the sense discussed earlier), every datagram is given a sequence number. A MSS sends an ACK for each datagram transmitted to the MH (note that the datagram may not be received by the MH because of fading or other interference). Until a datagram is ACKed, it is buffered at the network layer of the SH. If a MH moves away from its current MSS to a new MSS, the old MSS discards all messages but simultaneously informs the SH of the sequence numbers of these discarded messages. These messages are then retransmitted to the new location of the MH. A detailed protocol is presented in Gahi[6] (though our current implementation contains many changes).

For routing over the fixed network (e.g., between SHs or between an SH and a service provider), the existing routing protocol provided over the fixed network is used. Thus the network layer shown in Figure 4 is local to the mobile sub-networks only. The network layer shown in Figure 4 consists of two sub-layers – a tracking and VC maintenance sub-layer sitting on top of IP. The tracking sub-layer is responsible for maintaining location information for each MH currently in that mobile sub-network. VC maintenance refers to the task of guaranteeing reliable delivery of datagrams. It is noteworthy that we currently use IP for routing purposes. This choice is dictated more by budgetary constraints than scientific ones.

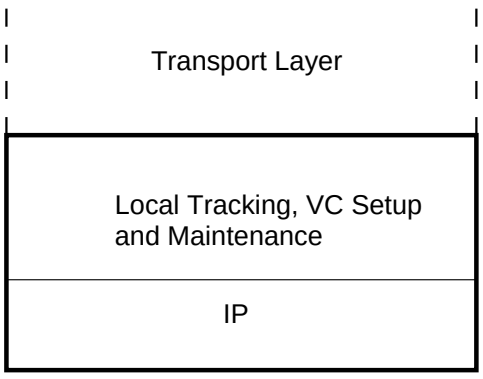


Figure 4: Network Layer.

Microsoft Corp. Exhibit 1009

3.2 Transport Layer Design

When developing transport layer protocols for the mobile environment, it is important to keep the following constraints in mind:

- The wireless link is very fragile and error-prone. This means that any reliable protocol must perform a significant amount of error-recovery.
- The bandwidth of the wireless link will always be a limited resource. Thus, all protocols need to be 'lean'. For instance, in the case of TCP-like protocols, it is not a good idea to have end-to-end flow control (where one end is mobile) because of the high number of control messages that will be sent.
- Available bandwidth within a cell may change dynamically (because it is impossible to control the number of users per cell). This leads to all kinds of problems in guaranteeing QOS parameters such as delay bounds and bandwidth guarantees.
- Mobile hosts frequently encounter extended periods of disconnection (caused due to hand-shake or due to physical interference with the signal) and this will result in significant losses of data for UDP-like protocols. We need to redefine best-effort service for such cases.
- Mobile hosts may move between SHs after opening transport connections. Should the old SH continue to be responsible for these connections or should control be transferred to the new SH?

All of the above issues can be summarized in the form of two questions. The answer to these questions will determine the transport layer design.

(1) Should the transport layer be aware of the mobility of MHs?

(2) Should the transport layer be aware of bandwidth fluctuations at the wireless link?

If we were to strictly follow the layering idea of the OSI hierarchy, mobility and bandwidth fluctuations will have to be concealed from the transport layer. For mobile networks, however, even though we can adhere to this philosophy, it will result in degraded performance and increased message overhead. To see why this is the case, let us consider two scenarios that will be commonplace in a mobile environment. In the first, a MH with open data connection(s) moves into a cell where there are many other MHs. It is likely that the negotiated QOS for the connections of this MH can no longer be satisfied and will have to be renegotiated. Since renegotiation of QOS parameters involves the transport layer (and the network layer), it is not possible to conceal bandwidth fluctuations from the transport layer. We discuss this problem further in section 3.2.4.

Microsoft Corp. Exhibit 1009

For the second scenario, consider what happens if a MH with an open TCP connection moves from its current SH (where the TCP connection was established) to another SH. In this case it is not necessary to re-establish the TCP connection (because the network layer can forward datagrams from the old SH to the new SH). However, the efficiency of the TCP connection will be degraded for reasons discussed in section 1.1 (recall that I-TCP attempts to alleviate this efficiency problem by breaking the TCP connection in two). In our architecture, as we discuss in section 3.2.1 and section 4, since we perform a great deal of bandwidth management within each mobile sub-network (controlled by a single SH), it becomes necessary to re-establish connections when a MH moves from the sub-network of one SH into the sub-network of another. Thus, we believe, the transport layer will need to be aware of the mobility of MHs and bandwidth fluctuations for each open connection.

Our view for the transport layer of the mobile sub-network is shown in Figure 5. The prefix M stands for 'mobile'. Thus we have a version of mobile-TCP and mobile-UDP. M-CM refers to the mobile-continuous media protocol and is useful for implementing real-time services such as voice or video. It is loosely based on the continuous media protocol of Moran[12]. The mobility management module deals with the problems of mobility (i.e., re-establishing transport connections when a MH moves between sub-networks and renegotiating QOS parameters).

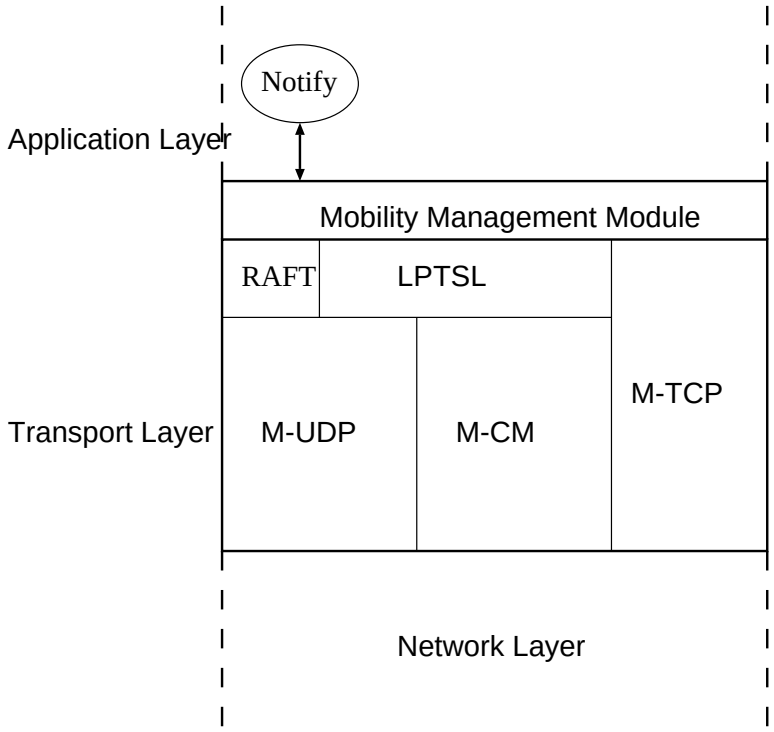


Figure 5: Transport Layer.

Microsoft Corp. Exhibit 1009

3.2.1 M-TCP

In section 1.1 we touched upon the problems associated with providing TCP-like service over mobile networks. Specifically, if the receiver is a MH, the TCP sender times out frequently because of the 'blackout' periods in mobile networks. The solution of Barke[1] breaks the connection in two (similar to our idea with the difference that they consider mobile networks to be part of the internet) – from the fixed host to the MSS and from the MSS to the MH. The MSS effectively serves as the TCP connection end-point from the point of view of the fixed host. The MSS is then responsible for forwarding all data reliably to the MH. Note that the *semantics* of this implementation differ from TCP semantics – it is possible for the sender to think that all data has been correctly delivered to the MH (since the MSS has sent ACKs) even if this is not the case.

In our design, the TCP connection is broken in two – fixed host to SH and SH to MH. The fixed host to SH part of the connection uses regular TCP while M-TCP is used for the SH to MH part of the connection. Since M-TCP is layered on top of a reliable virtual circuit connection (reliable within one mobile sub-network), its design is relatively simplified. Even though this design looks similar to I-TCP, unlike Brake[1], we implement almost TCP-like semantics that make it easier for the sending application to recover from the type of error described above. The TCP client on the SH always ACKs *all but the last byte* of data received from the sender. The last byte is ACKed only after it has been successfully sent to the MH by M-TCP at the SH. In this scenario, if the MH disconnects before receiving all data from the SH, the sender will never receive an ACK for the last byte. Thus, the sending application will know that the connection has failed and can take remedial action. In addition to the above change, we ensure that the buffers at the SH are not exhausted (which can happen, for instance, if the MH has been temporarily disconnected) by linking buffer availability (i.e., receiver window size) to the *available bandwidth on the wireless link*. This causes the TCP client to automatically choke the sender when the MH is in a crowded cell! This linkage between expected wireless bandwidth and TCP buffers is another unique feature of our design.

It is important to note, however, that M-TCP semantics are still slightly different from TCP semantics. This is best illustrated by considering a *talk* application. The application displays data on the user screen as and when it is received over the TCP connection. If the SH crashes, it is possible for the sender to think that almost all the lines it has typed thus far have appeared on the receiver's screen (because ACKs have been received for all but the last byte) even though this may not be the case (i.e., the SH crashed before that data was sent to the MH). This situation is not completely hopeless, however, because the sender will eventually realize that the SH has crashed and it can then take remedial action.

What happens when a MH moves from the sub-network of one SH into the sub-network of another? In our implementation this is handled in a manner similar to I-TCP. The old SH transfers TCP state to the new SH after it has been informed by the mobility management module about the move (recall that whenever a MH enters a new cell, it performs a handshake procedure

Microsoft Corp. Exhibit 1009

with the new MSS. The information exchanged during this handshake includes the identity of the previous SH and information about all open connections, see Ghai[6]). Meanwhile all datagrams arriving at the old SH are sent on to the new SH via IP loose source routing.

In our implementation, we require the MH to maintain the identity of the original SH (who first set up the TCP connection). If the MH moves from $SH_{original}$ to SH_1 to SH_2, after SH_1 has transferred TCP-state to SH_2, $SH_{original}$ is given the IP address of SH_2 so that it can route datagrams directly to SH_2 without going through SH_1 first. This method keeps the cost of forwarding datagrams small. In the future, if IP is replaced by a protocol like VIP, we will not have to worry about this particular problem. This is because the intermediate routers in the fixed network will automatically associate the IP address of SH_2 with the VIP address of the MH (see Singh[14] for more details) and route datagrams accordingly.

3.2.2 M-UDP

If we were to implement UDP in the mobile network without any changes, the performance seen by a MH would be very poor. This is because packets transmitted while the MH is moving between cells or is blocked by some physical obstruction are lost (note that in TCP these packets would be retransmitted). Only a small percentage of loss is due to lack of bandwidth. A high-level view of our M-UDP protocol as implemented is the following (a detailed protocol may be found in Brown[2]).

- Every UDP packet is buffered at the SH.
- The SH discards a packet if it has run out of buffer space or if it has been transmitted n times to the MSS.

The semantics of M-UDP are almost identical to UDP in the sense that delivery is not guaranteed. However, M-UDP attempts a 'best effort' service that is constrained only by buffer space availability at the SH and by bandwidth availability on the wireless link. In our experiments we observed a 2 to 3 fold improvement in the number of packets delivered by M-UDP in comparison to UDP (for mobile hosts).

3.2.3 M-CM

A large percentage of future applications will require transmission of data at regular intervals. This kind of data is referred to as continuous media and some examples include voice communication, video communication, etc. Continuous media applications have severe time deadlines and bandwidth requirements and thus cannot be implemented on top of message-based transport protocols such as M-TCP or M-UDP. Following Moran[12] we propose a separate transport protocol suite called M-CM (mobile-continuous media) that will provide the functionality required by such applications. Unfortunately, however, the protocols proposed in Moran[12] and other solutions

Microsoft Corp. Exhibit 1009

proposed for the high-speed network domain cannot be adapted to the mobile networks because of two reasons:

- bandwidth availability varies in an unpredictable manner as hosts roam,
- fading and handoff cause periods of disconnection.

All proposals for CM-type protocols require the network layer to provide strict guarantees regarding bandwidth availability and delay bounds for each connection. Because of the above reasons, however, this is not possible in the mobile domain.

Our approach is to provide a "best-possible" service to M-CM connections. Intuitively, this means that the SH will arbitrate between various MH connections to determine how much bandwidth must be allocated to each open connection. Thus, if a MH has an open ftp connection (via M-TCP) and an open video connection (via M-CM), a *scheduler* process (CS process, see section 4) will starve the ftp connection in favor of the video connection if the available bandwidth, within the current cell of the MH, gets reduced. The scheduler also interfaces with LPTSL in case some fraction of data along the M-CM connection(s) need to be discarded. This M-CM protocol has not yet been fully specified.

3.2.4 LPTSL (*Loss Profile Transport Sub-Layer*)

Future applications to be provided to mobile users will include audio (e.g., telephone, audio conferencing, etc.) and video applications (e.g., map information, viewing movies, etc.). These applications have real-time constraints and therefore need to be implemented over M-CM. However, we have a unique problem of *dynamic bandwidth changes during the lifetime of a connection* caused due to the **unpredictable mobility** of mobile hosts.

To illustrate a consequence of this unpredictable mobility, consider a situation where several mobile users have opened high-bandwidth connections. When these connections are set up, the network ensures that the users receive some guaranteed bandwidth. Since these users are all mobile, it is possible that many of them could move into the same cell. In such a situation, it is very likely that the requested bandwidth of the cell will exceed available bandwidth resulting in the original QOS (quality of service) parameters being violated. This situation does not arise in high-speed networks because users are not mobile during the life-time of a connection.

To deal with this situation, we propose that most open connections, *than can tolerate losses*, within the choked cell be penalized (either equally or differentially, based on some need-based policy). Thus, a penalized connection will see a $x\%$ reduction in available bandwidth. The question now is – does the MH renegotiate the QOS parameters to force the sender to reduce the connection bandwidth (e.g., use a higher compression ratio for a video connection) or is data for that connection discarded by the SH? The first option sounds attractive but is not necessarily the right choice for the following reasons:

Microsoft Corp. Exhibit 1009

- The bandwidth crunch is probably temporary and will be alleviated as soon as a MH roams out of the cell. When this happens, the QOS parameters will have to be renegotiated.
- The cell latency of a MH is of the order of several seconds. The end-to-end renegotiation process is time consuming and thus the bandwidth available may change even before this renegotiation is completed.
- While the renegotiation is going on, data will continue to be sent at the old rate. Since the wireless bandwidth is small, the buffers at the SH will possibly overflow.

We propose that the SH *judiciously* discard data for each penalized connection. Since the SH operates at the transport layer (it is a gateway), it can do this. Note, however, that indiscriminate discarding of data may result in garbage at the MH (e.g., if data is randomly thrown out of a compressed video stream, no video can be reconstructed). To solve this problem, we have proposed a new sub-layer called the Loss Profile Transport Sub-Layer. The sending application puts *flags* in the data stream by making calls to LPTSL. All data between a pair of flags represents a *logical segment* (e.g., one logical segment may be a single compressed frame in JPEG-video). The LPTSL at the SH discards entire logical segments in the event of a bandwidth crunch to ensure a $x\%$ reduction in bandwidth (note that the application at the MH can be informed of the location of the discarded segments in case that information is required – as in MPEG-2 video). A detailed protocol is presented in Seal[13]. See Figure 6 for an explanation of the operation of this layer. Here the data stream at the sender is broken into data segments (logical segments) separated by special flags. These flags are inserted by the LPTSL layer at the sender. The LPTSL layer at the SH is informed of the available bandwidth in the current cell of the MH and determines if any data needs to be discarded. If so, it discards entire logical segments (all data between consecutive flags). The LPTSL at the MH passes up the arriving data to its receiving application *and indicates the location and size of the discarded segments.*

A reason for discarding data at the LPTSL is because LPTSL provides different *discarding functions.* When a connection is set up, a QOS parameter negotiated is the discarding function(s) to be used in the event of a bandwidth crunch. For instance, if the connection is an audio connection, the user may prefer uniform random loss as opposed to bursty loss. On the other hand, if the data is compressed video, random loss will prove to be disastrous. In this case the user may opt for bursty loss (i.e., discard entire frames rather than random bytes from several frames). These discarding functions are provided in the form of a indexed library of sub-routines at the LPTSL layer of the SH.

3.2.5 RAFT (*Repetitive Almost-reliable Fast Transport*)

In addition to applications discussed in section 2, PDAs will be used as devices whereby critical data can be transmitted to the user quickly – much like today's pagers or beepers but with a

Microsoft Corp. Exhibit 1009

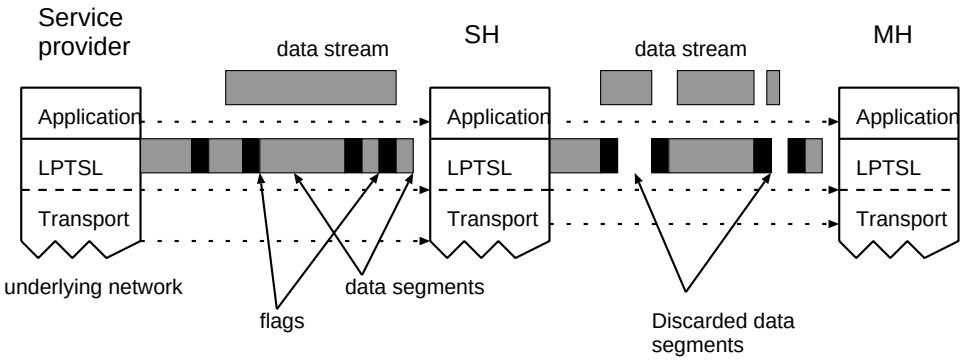


Figure 6: Loss Profile Transport Sub-Layer.

greater degree of sophistication. These type of *notification* applications need to be able to transmit data *quickly* and *reliably* to the mobile user. To facilitate the development of such applications we propose a transport sub-layer called RAFT that is built on top of M-UDP. The approach we use is the following – RAFT at the SH sends data several times. When the MH eventually receives all data correctly it sends a *shut_off* message to RAFT at the SH. RAFT data takes precedence over all other connections.

4 Management and Control

In Figure 7, we show the transport and network layer control and management functions at the SH. The SH needs to ensure that QOS parameters for open connections are maintained. This implies appropriate bandwidth management within each cell and buffer allocation for each connection at the SH. In Figure 7 the management and control functions of the protocol stack for the mobile sub-network is shown in detail. The different arrows indicate control paths, management paths, data paths and QOS negotiation paths.

At the network layer we have a process (CBM) that monitors the bandwidth utilization within each cell controlled by the SH and passes this information up to a transport layer process (CS) via a management path. The CS process arbitrates bandwidth within each cell between all open connections. Thus, if there is a real-time connection (e.g., audio) and a data connection into a cell, the CS may choose to starve the data connection in order to ensure the delay bounds for the real-time connection are met. Another control process at the network layer (NL_QOS) monitors the QOS being delivered to each VC and sends this information to the CS as well. This management path is required to ensure that QOS parameters (such as bandwidth usage or delays) for M-CM connections are met. If some M-CM connection has not been receiving its negotiated QOS, the CS process allocates more bandwidth to that connection.

CS is the process which is responsible for maintaining QOS for all M-CM connections and ensuring that data connections (M-UDP or M-TCP) do not get starved in the process. The CS

Microsoft Corp. Exhibit 1009

receives information for each M-CM connection's QOS contract. Note that this contract may be renegotiated during the lifetime of the connection and thus CS needs to be informed of this change via the management path. M-UDP connections are subject to data discarding (via LPTSL) in the event of a bandwidth crunch. Thus, we assume that some QOS negotiation also takes place for UDP connections and this information is passed on to the CS process as well. CS *periodically* informs M-CM, M-UDP and M-TCP of the available bandwidth for each connection via control paths. It is up to these protocols to ensure that they control each open connection (either choke the sender as in M-TCP, or discard data via LPTSL as in M-CM) adequately.

QOS negotiations take place between M-CM's QOS process, NL_QOS and possibly the QOS process on the fixed network (in case the connection is to a fixed host) via QOS negotiation paths. QOS control information is also exchanged between LPTSL and the QOS processes of M-UDP and M-CM. Finally, the buffer manager processes at both, the network layer and the transport layer, are responsible for buffer allocation to different connections. The data paths followed for some typical connections are illustrated in the figure as well.

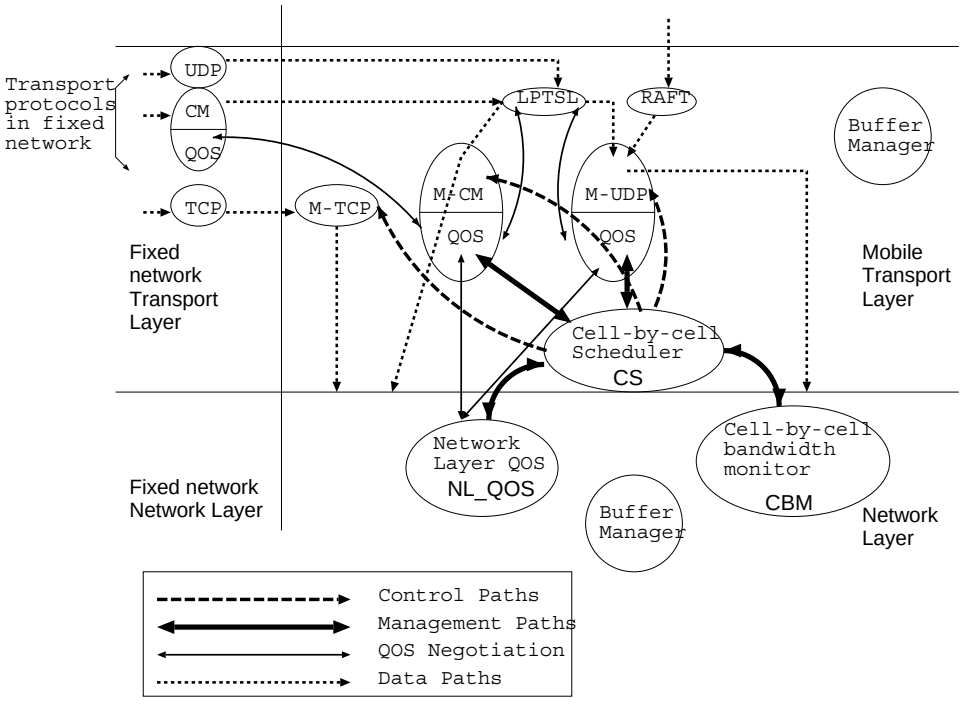


Figure 7: Management and Data flow at the transport and network layers of the SH.

5 Conclusions

In this paper we have proposed a complete design of the network layer and transport layer in a manner that best addresses the problems of the mobile environment. Our design is a radical departure from other researchers in that we propose that mobile networks be viewed as separate

Microsoft Corp. Exhibit 1009

from wired networks with connectivity being provided by special gateway nodes. These nodes provide a variety of transport level services that best meet the constraints of the mobile environment. A complete implementation of this architecture is underway and Figure 8 indicates the current status of this implementation.

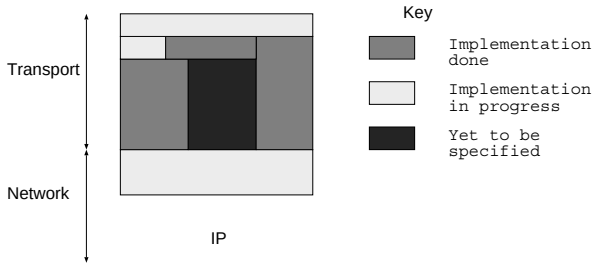


Figure 8: Current status of implementation.

References

[1] A. Bakre and B. R. Badrinath,"I-TCP: Indirect TCP for Mobile Hosts", *Technical Report* DCS-TR-314, Rutgers University, Piscataway, NJ 08855.

[2] K. Brown and S. Singh,"M-UDP: Mobile UDP", Manuscript.

[3] I. Arieh Cimet, "How to Assign Service Areas in a Cellular Mobile Telephone System", *IEEE ICC'94*, pp. 197-200, May 1994.

[4] J. S. DaSilva and B. E. Fernandes,"The European Research Program for Advanced Mobile Systems", *IEEE Personal Communications Magazine*, February 1995, pp. 14-19.

[5] D. Duchamp, Steven K. Feiner and G. Q. Maguire, "Software technology for wireless Mobile computing" *IEEE Network Mag.*, pp 12-18, November 1991.

[6] R. Ghai and S. Singh,"An Architecture and Communication Protocol for Picocellular Networks",*IEEE Personal Communications Magazine*, Vol. 1(3), 1994, pp. 36-46.

[7] David J. Goodman, "Cellular Packet Communications", *IEEE Trans. on Comm.*, vol. 38, no. 8, pp 1272-1280, August 1990.

[8] David J. Goodman, "Trends in Cellular and Cordless Communications",*IEEE Communications Magazine*, pp 31-40, June 1991.

[9] J. Ioanidis, D. Duchamp and G. Q. Maguire, "IP-based protocols for mobile internetworking" *Proc. of ACM SIGCOMM'91*, pp 235-245, September 1991.

Microsoft Corp. Exhibit 1009

[10] D. B. Johnson,"Mobile Host Internetworking Using IP Loose Source Routing", Technical Report CMU-CS-93-128, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.

[11] C.S. Joseph, *et al*, "Propagation Measurement to Support Third Generation Mobile Radio Network Planning", *43rd IEEE Vehicular Tech. Conf.*, May 1993, pp. 61-64.

[12] M. Moran and B. Wolfinger,"Design of a Continuous Media Data Transport Service and Protocol", Technical Report TR-92-019, Computer Science Division, University of California Berkeley, April 1992.

[13] K. Seal and S. Singh,"Loss Profiles: A Quality of Service Measure in Mobile Computing", *J. Wireless Networks*, (submitted).

[14] S. Singh,"Quality of Service Guarantees in Mobile Computing", *Journal Computer Communications*, (to appear).

[15] F. Teraoka and M. Tokoro, "Host Migration Transparency in IP Networks: The VIP Approach", *SIGCOMM*, Vol. 23, No. 1, Jan 1993, pp. 45-65.

Microsoft Corp. Exhibit 1009

Abstract

The ongoing European ACTS project *OnTheMove* provides support services for distributed mobile multimedia applications. The project defines, implements, and demonstrates a mobile middleware called a Mobile Application Support Environment (MASE) which is based on UMTS concepts. The mobile application programming interface (mobile API) of MASE, which will be submitted for standardization, allows common access to the underlying operating systems and network infrastructure, and facilitates the development of new, mobile-aware, multimedia applications.

UMTS: A Middleware Architecture and Mobile API Approach

BIRGIT KRELLER, SIEMENS AG

ANTHONY SANG-BUM PARK AND JENS MEGGERS, AACHEN UNIVERSITY OF TECHNOLOGY

GUNNAR FORSGREN, ERICSSON RADIO SYSTEMS AB

ERNÖ KOVACS AND MICHAEL ROSINUS, SONY INTERNATIONAL (EUROPE) GMBH

Any information at any time, at any place, in any form. This promise of mobile multimedia will be realized through third-generation mobile communication networks, which will offer high-bit-rate data services, guaranteed on-demand bandwidth, and low delays. The European Telecommunication Standards Institute (ETSI) is working on the Universal Mobile Telecommunications System (UMTS) [1, 2], which belongs to the family of similar or compatible standards developed within the International Telecommunication Union (ITU) called International Mobile Telecommunication in the year 2000 (IMT-2000) [3].

Today, mobile users already utilize a wide variety of mobile terminals ranging from simple mobile phones and personal digital assistants (PDA) to high-end multimedia notebooks. UMTS and the Mobile Broadband System (MBS) will offer suitable bandwidth and global connectivity to enable true mobile multimedia. As an early contribution to the UMTS service specifications, and in order to provide a smooth evolution path from second- to third-generation communication systems, the Advanced Communications Technologies and Services (ACTS) project *OnTheMove* has developed a mobile middleware system called the Mobile Application Support Environment (MASE). Along with the MASE, an application programming interface (API) has been defined that allows applications to access the MASE components. This API, called the *mobile API*, will be submitted for standardization. The purpose of the MASE middleware is to ease the development of mobile-aware applications by providing a common underlying platform. Furthermore, the middleware approach enables a smooth transition from current wireless networks, such as Global System for Mobile Communications (GSM) and Digital European Cordless Telecommunications (DECT) to future UMTS networks. Instead of directly accessing the operating system, mobile-aware applications make use of the *mobile* API and benefit from simple access to MASE services, which hide the complexity of heterogeneous networks and operating systems from the applications. Thus, MASE simplifies the development of mobile-aware applications and frees them from the complex processing caused by additional needs when accessing heterogeneous networks and running on mobile devices. Furthermore, MASE eases the evolution of multimedia applications toward UTMS [4] and enables legacy applications to benefit from a subset of its functionality.

In this article we focus on the services provided by MASE and their relation to UMTS. We demonstrate the interworking between different parts of MASE through a typical mobile-aware application, the CityGuide. The CityGuide uses geographical information provided by the MASE Location Manager module to display a map of the current geographical surroundings of the mobile user. Several different layers of interesting places (e.g., public transportation, administration buildings, accommodation, restaurants, and much more) are shown on the map and are linked to corresponding Web pages.

We will first elaborate on the generic MASE architecture. We will outline a possible scenario for the ongoing UMTS service definitions and then explain, step by step, the MASE components accessed by the CityGuide implementation.

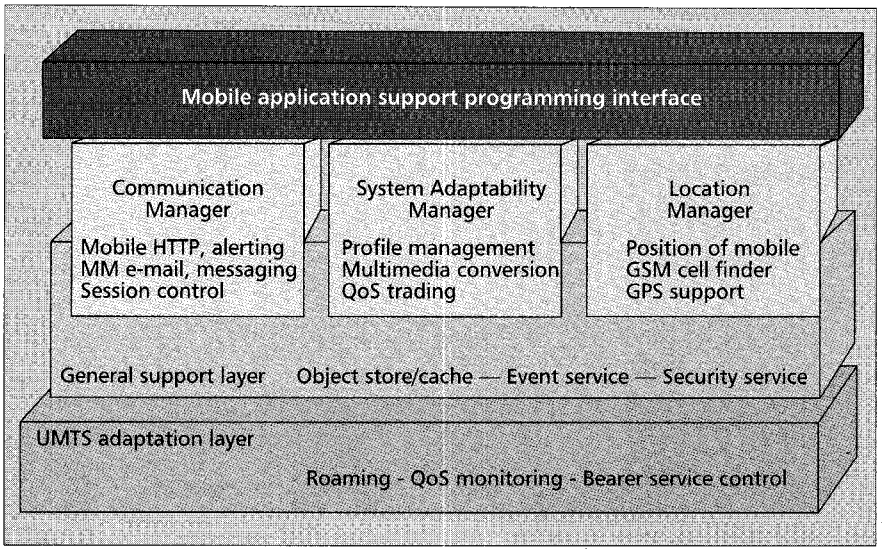
MASE

The Mobile Application Support Environment is a distributed system that runs on both the mobile device and the so-called mobility gateway. The latter acts as a mediator between the wireless and fixed network infrastructures. It works as an agent for mobile clients which are typically connected over unreliable wireless access networks with low bandwidth. MASE enables access to the UMTS adaptation layer (UAL), which provides applications and middleware components unified access to all possible underlying networks. An additional general support layer provides the functionality required for distributed systems. On top of both layers several manager components are installed, providing different dedicated services. Figure 1 shows the overall MASE architecture with its corresponding building blocks. All the components shown have been implemented.

MASE is built around the concepts of awareness, adapta-

1070-9916/98/$10.00 © 1998 IEEE

Microsoft Corp. Exhibit 1009



■ **Figure 1.** *Overall architecture of MASE.*

tion, and abstraction. MASE applications are aware of the current network quality of service (QoS) through sophisticated monitoring and management facilities which are provided by the UAL. The UAL hides network specific details by selecting the appropriate bearer service and protocol stack according to the requested QoS. The general functional features of the UAL are:

- Selecting appropriate transport protocols and configuring them for efficient use (e.g., adjusting packet size and timers to the bearer service parameters)
- Selecting and configuring an appropriate bearer service
- Managing roaming between different networks and bearer services as well as bearer service switching

Details of the UAL have been published elsewhere (e.g., [5]). The General support layer implements object storing and caching as well as event handling and security services. These services will also not be described in detail here.

Awareness – End terminal characteristics and user preferences are stored in profiles. They are managed by the Profile Manager, a part of the System Adaptability Manager (SAM), and are available on demand at all nodes of the system.

Adaptation – Profile information and monitored QoS are used by the MASE communication facilities to adapt their usage to the current QoS situation and user requirements. This adaptation is transparent to the application.

Abstraction – The MASE provides high-level abstractions, for example, an alerting function or location management. An alert is an abstraction of an important short message which has to be sent to the user. Depending on the current network situation, the alert manager maps an alert to different network services. If the user is connected to the network over TCP/IP, the message will be delivered directly to an alert server on the mobile device messaging service, or as a GSM short message service (SMS) notification if no such connection is available. The final version of MASE will deal with adaptation to varying degrees of QoS, robustness in the face of disconnected links, roaming between different operators and network types, personalized information filtering, and location-aware applications using various location trackers.

MASE also integrates legacy communication applications and improves communication over wireless networks. In this way, the MASE takes up the challenges of mobile multimedia and prepares for UMTS.

The CityGuide Application Scenario

The CityGuide is part of a set of mobile-aware applications (Fig. 2). It is a typical mobile user application providing access to a map of the surroundings of the mobile user. This map displays several information layers, such as hotels, restaurants, automated teller machines, bus stops, and phone booths. These information sources are linked to Web pages to allow instant access to further information about a particular location.

The CityGuide runs as a Java applet within a Web browser and provides access to maps describing the current surroundings. This application uses the *mobile* API to ask the MASE Location Manager for the actual coordinates (longitude and latitude) of the user. This information will be checked against the coordinates stored in the server. The most suitable map and the associated geographical objects will be loaded using HTTP. Since the browser is a legacy application and normal HTTP is not well suited to mobile communication, the MASE integrates these calls using an HTTP proxy system. In the following we describe the MASE components participating in this process.

Location Manager

The Location Manager (LM) helps users navigate in new environments. It enables applications and other MASE components to determine the parameters of the current geographical position of a mobile device as well as the accuracy of these values. This is another example of the abstractions provided by MASE because the geographical information is accessible through a simple uniform API and independent of the mechanism used by the LM to gather location data. A subset of the LM API calls is shown in Table 1.

The current LM implementation supports the Global Positioning System (GPS), a satellite-based radio navigation system developed and operated by the U.S. Department of Defense [6], and uses WaveLAN (a wireless LAN device from Lucent Technology [7]) cell identifier information. GPS provides latitude and longitude coordinates, velocity, and the user's moving direction with an accuracy of about 10–300 m.

Method	Description
getAccuracy	Returns the position error of the device
getLastDateOfUpdate	Returns information about last date of update of GPS location information
getLastTimeOfUpdate	Returns information about last time of update of GPS location information
getLatitude	Returns the latitude in WGS84 format
getLongitude	Returns the longitude in WGS84 format
getVelocity	Returns the speed of the device

■ **Table 1.** *A subset of the Location Manager API.*

Microsoft Corp. Exhibit 1009

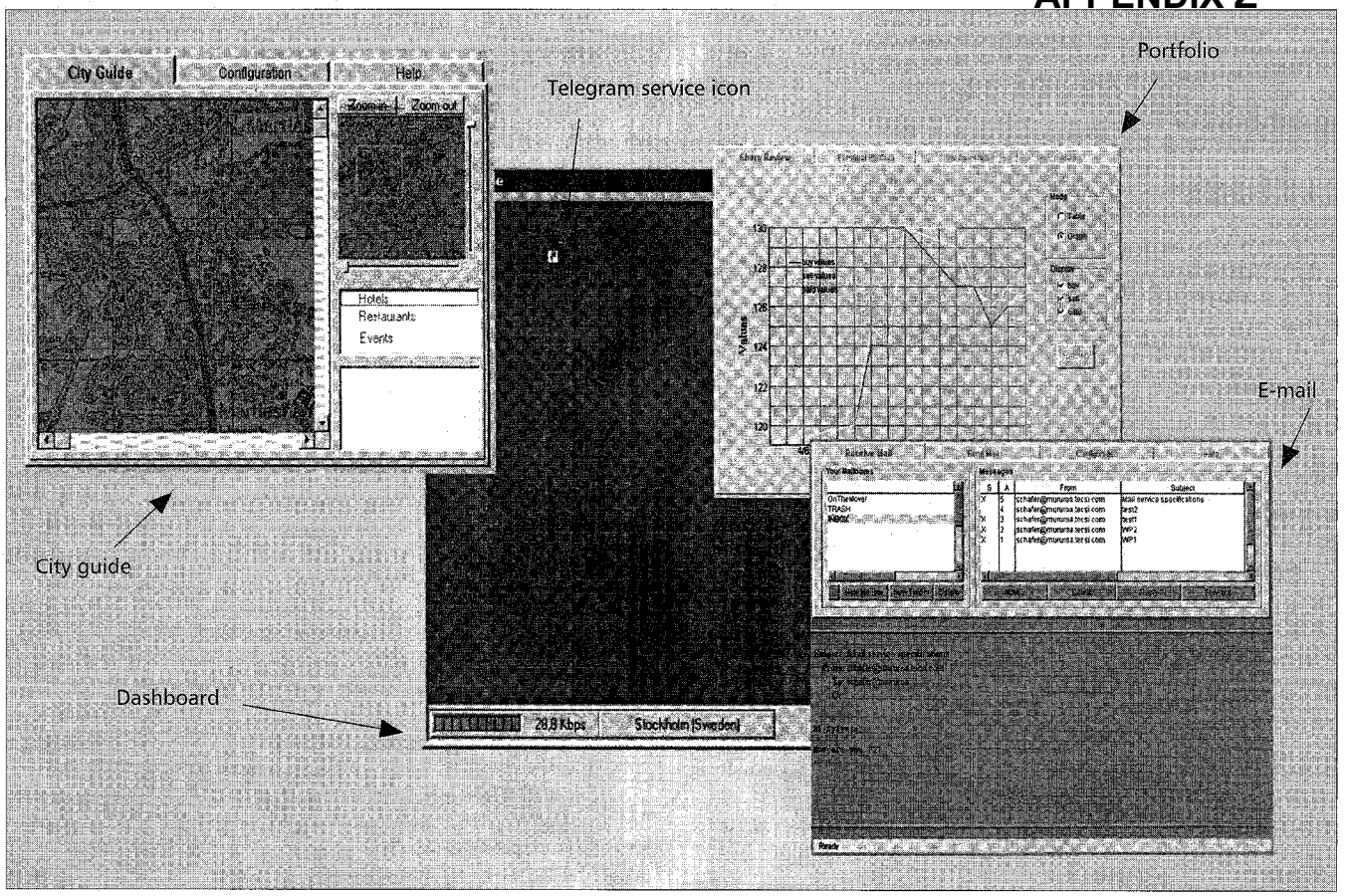


Figure 2. *OnTheMove desktop and mobile-aware applications.*

In wireless communication systems cell identifiers can help determine the position of the mobile device. The accuracy depends on the network cell size and varies from about 30 m (wireless LAN) up to a few kilometers using GSM. A table lookup maps these cell identifiers to the physical position of the terminal. A central LM process communicates with the available location information sources. The retrieved values are made available to MASE-aware applications and other MASE components by stub interfaces to the central process. This architecture is shown in Fig. 3.

Communication Manager

The Communication Manager (CM) of the MASE architecture supports HTTP and e-mail communication, HTTP prefetching, alerting, messaging, and disconnected operations. In the following we will focus on the HTTP proxy. The HTTP proxy system improves the performance and usability of HTTP over low-bandwidth network connections (cellular, radio LAN, modem). Requested multimedia objects can be processed/converted by MASE to match the current network bandwidth, terminal characteristics (display capability, storage capacity, etc.), and user requirements. This adaptation is performed at the mobility gateway before transmitting the data to the mobile terminal. The distribution of the HTTP proxy functionality is shown in Fig. 4.

Multiple users simultaneously access Web-based information using the HTTP protocol. Each mobile terminal runs a client-side HTTP proxy (CSP) that requests Web objects from the server-side HTTP proxy (SSP). SSP can serve multiple client connections in parallel, with each connection being handled by a separate connection handler thread. Each such handler communicates with a peer process on the CSP over a multiplexed logical communication session. Multiple sessions run on top of a single TCP/IP connection between a mobile terminal and the mobility gateway. The multiplexing scheme allows data transfer for all logical sessions from one terminal to be transmitted in parallel over a single full-duplex TCP/IP connection.

In a typical scenario, an HTTP client on the mobile terminal (such as the CityGuide application) will request HTML pages and related graphic objects from a remote HTTP server. The HTTP Proxy system intercepts these calls and communicates with the SAM to adapt the requested objects to the current QoS and user requirements. The application (in this case the browser) is configured to use the CSP as an HTTP proxy server.

As shown in Fig. 4, for each HTTP request the client will

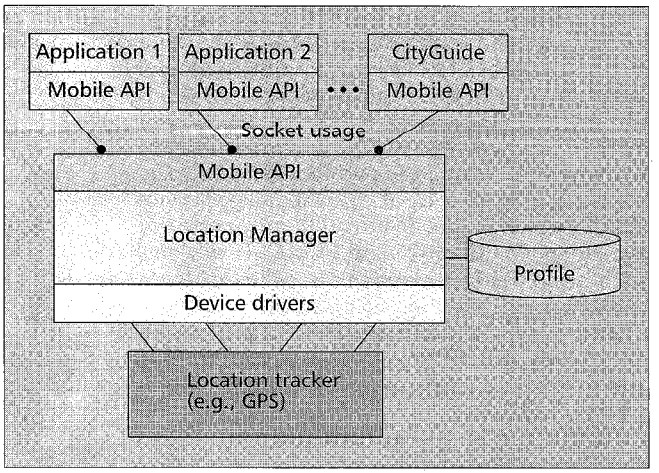


Figure 3. *Location Manager architecture.*

Microsoft Corp. Exhibit 1009

establish a TCP/IP connection to the CSP ①. The CSP checks whether the requested object is available locally by calling the `checkCache()` methods of the SAM ②. See Table 2 for the description of the SAM API. If the SAM indicates local availability, the object will be returned by the CSP to the requesting client.

Otherwise, the CSP will create a logical communication session to the SSP side and will pass the HTTP request to the SSP ③. Another `checkCache()` call ④ checks whether the requested object is available on the Mobility Gateway. If not, the object is fetched from the HTTP server ⑤ and inserted into the local cache using the `createMMO()` method. The SSP now calls the `trade()` method to adapt the multimedia object. This trading process is described in the next section. The SAM creates a variant of the original object and inserts it into the family of related objects managed by the local cache. The variant together with instructions for the post trading phase (e.g., which decompression method to use) is returned and transferred to the CSP. Here a post trading phase is initiated and the resulting object passed to the requesting client.

Method	Description
trade()	Performs trading
postTrade()	Performs required processing steps on the receiver side (e.g., decompression)
checkCache()	Searches cache for a local available MMObject
createMMO()	Creates an initial MMObject
insertVariant()	Inserts an MMObject variant into the family of related objects
removeVariant()	Removes variant from the family

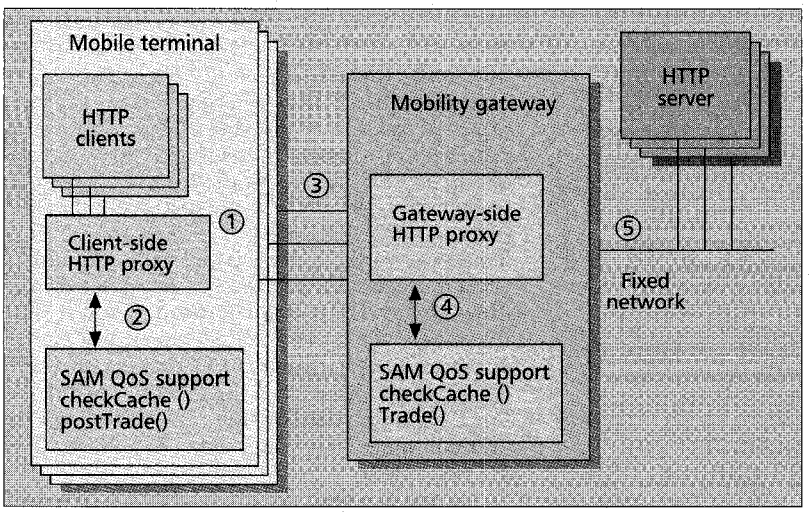
■ **Table 2.** *A subset of the QoS trader API.*

System Adaptability Manager

The System Adaptability Manager (SAM) is responsible for the provision of optimized and personalized mobile services. User-, network-, and terminal-specific QoS parameters are managed in profiles handled by the Profile Manager. These profiles are used to compute the best adaptation of the MASE communication services, taking into account the current network and end system QoS parameters as well as to the user's personal preferences. The SAM has several adaptation possibilities:

- It can make a choice between different available networks based on the available bandwidth, bandwidth guarantees, and cost.
- It can compress, convert, transcode, or reduce the multimedia objects prior to transmission.

For example, an image is supposed to be transmitted to a terminal with a black and white screen. In this case color information can be eliminated by the mobility gateway. Other reasons for adaptation can arise, for example, from the available network bandwidth and the cost involved. These decisions are made by the QoS trader within the SAM.



■ **Figure 4.** *The partitioning of the HTTP proxy.*

user.image.reductionAllowed	True
user.image.maxTransmissionSize	20,000 bytes
user.image.maxWaitTime	5 s
user.image.resolutionReduction	Yes
system.terminal.display	Black and white
system.terminal.memory	2 MB
system.terminal.diskSize	20 MB
network.currentBearer	GSM
network.currentBearer.expThroughput	9600 b/s

■ **Table 3.** *Profile values.*

Profile Manager

The terminal characteristics of the mobile device are stored in a *terminal profile*. The network characteristics of the mobile device's current (wireless) connections are stored in the *network profile*. The network profile is constantly updated by the UAL. User-specific preferences are stored in a *user profile*. A profile generally contains a hierarchical tree of properties (name/value pairs), each describing a certain property. Profiles are replicated on demand and stored persistently throughout the MASE system.

A MASE-aware application can access the profile values at either the mobile device or the mobility gateway. Consistency can be enforced independently at every host and for each node to reduce the communication overhead. Table 3 shows examples of profile values for the user, system and network tree.

QoS Trader

The QoS trader adapts the MASE communication services according to the user's preferences and the current terminal/network situation as reflected in the profile. Instead of transmitting multimedia objects directly to the mobile client, the HTTP proxy calls the QoS Trader to perform a trading process. This process consists of the steps illustrated in Fig. 5.

During the *QoS gathering* phase the QoS trader accesses the current terminal and network QoS which are stored in the profile. The profile also contains the user's preferences, which are later used to compute filters and preferences for the planning process. It further examines the properties of the current multimedia object, and then enters the *planning* phase, during which it decides about the actions to perform on the current multimedia object. This process generates a "new" plan from a knowledge base (*plan generation*) and the gathered QoS parameters.

A plan consists of one or more compression, conversion or reduction steps. During the *plan*

Microsoft Corp. Exhibit 1009

filtering phase the static properties of the new plan are matched against the user preferences and terminal requirements. Small devices, for instance, might have implemented only one or two decompression methods. The plan filtering phase will only select plans suitable for this particular device.

Subsequently, the QoS trader predicts the prospective outcome of the current plan using knowledge provided by the multimedia conversion (MMC) routines. MMC works *online*. It offers lookup functions that enable it to estimate conversion time, and execute conversions if appropriate. MMC provides some general-purpose methods for image manipulation, such as conversion of images to other image formats by means of scaling, graining, and color reductions.

Two methods that support the QoS prediction and filtering phases of the QoS trader, by estimating the required conversion time and the size of the reduced objects, are important for the SAM, which checks whether varying the reduction R of an object x can fulfill Eq. 1.

$$T_{MaxWait} > T_{Seek}(x_{org}) + T_{Reduce}(x_{org}, R) + T_{Trans}(x_{red}, B) \quad (1)$$

$T_{Trans}(x,B)$ is the transmission time of an object x at bandwidth B (b/s), $T_{Reduce}(x,R)$ is the estimated reduction time of the original object x at reduction R. $T_{Seek}(x)$ is the time used to estimate the reduction time of an object x to the requested file size (planning process). $T_{MaxWait}$ (stored in the profile under `user.image.maxWaitTime`) is a user preference parameter which defines the maximal time the mobile client wants to wait for an image. If $T_{MaxWait} > T_{Trans}(x_{org}, B)$, there is no need to reduce the file size, and the SAM transmits the original image.

A set of suitable conversion commands have been selected for our purpose. For images we use scaling, reducing colors, dithering, and converting to black and white to reduce image sizes. Some formats like JPEG allow conversion by scaling and reducing the overall quality. All those commands are "lossy"; they reduce the quality of an image and hence reduce the file size. To obtain a table of relative values for conversion predictions we have measured a set of images with all available conversion commands.

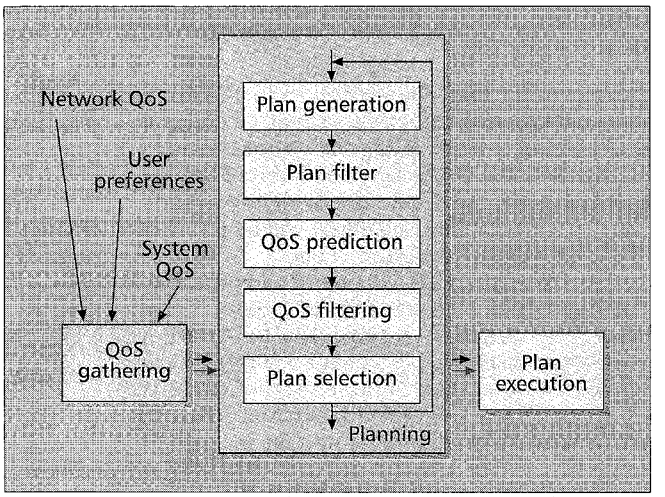


Figure 5. *The QoS trading process.*

Using this knowledge the trader can estimate the QoS parameters resulting from converting and transmitting the converted multimedia object (e.g., the overall processing time). These predicted QoS parameters are matched against the filters derived from the profile setting. During the *plan selection* phase a preference index is computed for each plan which has passed the filters. The used preference function is selected according to a user-defined criterion (e.g., the smallest resulting object size, the shortest time required for converting and transmitting the result, the best quality remaining).

When the planning is finished, the best plan with the highest preference index is executed. The resulting object is either returned to the application (local trading) or stored locally and will be marshaled for transmission.

Method	Description
compress	Compresses a file
decompress	Decompresses a file
estimateReductionTime	Returns estimated time to perform a desired reduction
reduceFileSize	Reduces file size by the desired reduction
convert	Converts an image into a given type
colorRedQuant	Reduces the number/depth of colors
colorRedDepth	Reduces the depth of color
colorRedDither	Reduces the number of shades per color
getHeigth	Returns the height of an image
getMaxVal	Returns the max value of shades per color
getSize	Returns the file size in bytes
getType	Returns the image type as String
getWidth	Returns the width of the image
isCompressed	Returns true if the image is compressed
isImage	Returns true if the instance is an image

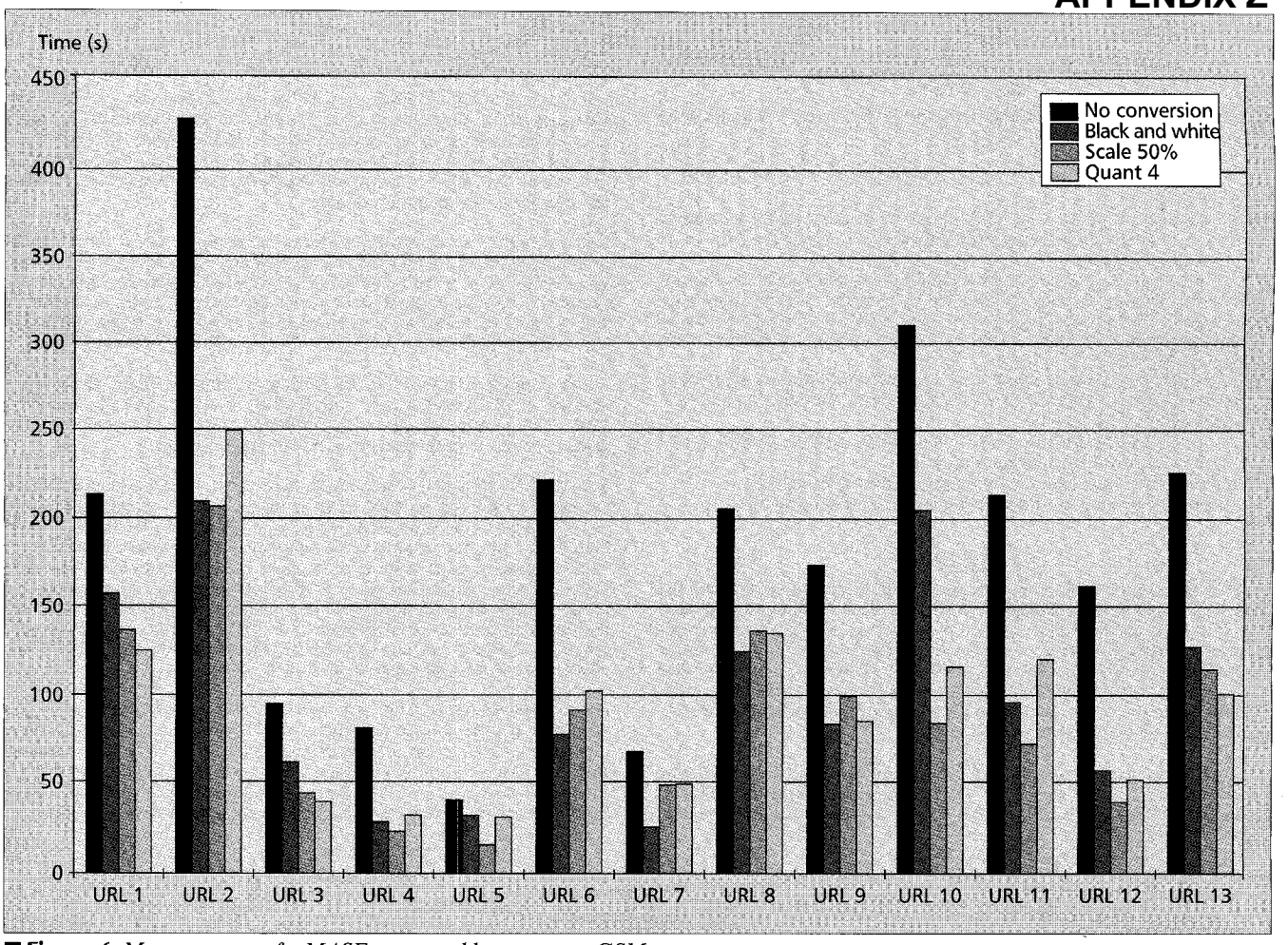
Table 4. *Subset of multimedia object conversion API.*

Example – The *plan generation* phase generated the plan "SCF(0.5); TRA; SCF(2)," which defines a scaling operation by the factor of 0.5, the transfer of the image, and the rescaling of the image to the original size during the `postTrading()` call on the mobile terminal. During the *plan filtering* phase this plan is compared with user preferences stored in the profile. For example, the parameter user.image.resolutionReduction defines whether or not the user accepts resolution reductions (scaling). If the user does not allow scaling, the filter derived from that value will prevent the above plan to be selected.

During the *QoS prediction* phase, the time required to execute the above plan will be computed by predicting the time required for the conversion SCF (0.5), for the transmission of the reduced image, and for the reconversion (SCF (2)) on the mobile device. The time and resulting image size for both conversions are computed by using average results derived from former conversions which were initially performed offline. Furthermore, the computing power of the mobile device is taken into account by using relative computing indices from the terminal profile for this device. In addition, the time to transfer the original image and a user-specific quality factor are computed.

During the *QoS filtering* phase, these results are used in Eq. 1 to check whether the user's

Microsoft Corp. Exhibit 1009



■ **Figure 6.** *Measurements of a MASE-supported browser over GSM.*

requirement `maxTransmissionTime` is fulfilled and whether the reduction will result in a faster transmission. If the plan is still valid, a preference index will be computed. Several preference functions are possible, ranging from a selection based purely on the processing time to a mixture of processing time and the quality of the reduced image. Finally, the plan with the best preference index will be executed.

Figure 6 shows the achieved measurements over the actual cellular system, GSM, that has data transport capabilities of up to 9600 b/s. The results show the advantage of the QoS trading and multimedia conversion. Depending on the automatically chosen conversion method the interaction of CM, SAM, and UAL achieve up to 70 percent acceleration of the HTTP transmission time. Table 4 represents a small subset of the SAM API that application programmers may use to realize conversions offline, or even during online connections.

Future Work

Future work will deal with the downsizing of the current architecture and implementation to fit the requirements of small mobile devices (e.g., PDAs and PICs), which usually have only limited memory and CPU power. The implementation of the Location Manager will be extended by supporting more input devices, such as an interface to a DECT system which holds information concerning the actual interworking unit. Further development will be done on a Session Manager that provides a higher abstraction of connections than TCP/IP does. The Session Manager controls the ongoing communication sessions and allows scheduling of different MASE operations like prefetching and caching. Results of the project's field trials will be used to improve the current status.

Conclusions

The data services of future third-generation mobile telecommunication systems play a critical role in facilitating new and innovative mobile-aware applications and services. UMTS will define a set of services enabling seamless roaming between different networks, QoS monitoring, and bandwidth on demand. In this article we have introduced the OnTheMove approach, which employs middleware to support the special needs of mobile-aware applications. This middleware not only allows the development of mobile-aware applications in an easy way, it also shields today's application developer from the ongoing developments toward UMTS. The MASE mobile middleware will pave the road to future mobile telecommunication systems.

Acknowledgments

The authors would like to acknowledge all contributing partners of the OnTheMove project [8] which is sponsored partially by the European Commission in the ACTS program under contract AC034. The project participants are Ericsson Radio Systems AB, T-Mobil GmbH, Ericsson Eurolab GmbH, Siemens AG, Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), IBM France, Tecsi, BT, Bonnier Business Press, Royal Institute of Technology (KTH), Sony International (Europe) GmbH, Burda Com Media Solutions GmbH, Centre for Wireless Communications (CWC), and Iona.

Microsoft Corp. Exhibit 1009

References

[1] EC DG XIII/B (1/3/96), "UMTS Task Force Final Report," Brussels,Belgium, ACTS InfoWin, http://www.infowin.org/ACTS/.

[2] J. Schwarz da Silva *et al.*, "Evolution Towards UMTS," ACTS InfoWin,http://www.infowin.org/ACTS/IENM/CONCERTATION/MOBILITY/umts0.htm.

[3] M. H. Callendar, Ed., *IEEE Pers. Commun.*, Special Issue on International Mobile Telecommunications-2000: Standards Efforts of the ITU, vol. 4, no. 4, Aug. 1997.

[4] J. Meggers and A. S. Park, "Mobile Middleware: Additional Functionality to Cover Wireless Terminals," *Proc. 3rd Int'l. Wkshp. Mobile Multimedia Commun.* (MoMuC-3), Princeton, NJ, Sept. 1996; D. J. Goodman and D. Raychaudhuri, Eds., *Mobile Multimedia Communications*, Plenum, 1997, pp. 151–57.

[5] J. Meggers, A. S. Park, and R. Ludwig, "Roaming between GSM and Wireless LAN," *ACTS Mobile Commun. Summit*, Granada, Spain, Nov. 1996, pp. 828–34.

[6] U.S. Coast Guard Navigation center, "GPS, DGPS, LORAN, OMEGA, LNM," http://www.navcen.uscg.mil/gps/gps.htm.

[7] WaveLAN Wireless Computing, http://www.wavelan.com.

[8] OnTheMove home page, http://www.sics.se/~onthemove.

Biographies

BIRGIT KRELLER (birgit.kreller@mchp.siemens.de) received her Dipl.-Inform. (Master's in computer science) from the University of Magdeburg, Germany, in 1996 on the subject of mobile agents for load balancing in large telecommunication systems. She joined the Siemens Corporate Research and Development Department in March 1996, where she is working in the ACTS project OnTheMove. Her current research interests include mobile computing architectures, wireless networks, geographical positioning systems for mobile devices, and mobile agents.

ANTHONY SANG-BUM PARK received his Dipl.-Inform. from Aachen University of Technology (RWTH), Germany, in 1995, previously studying at the University of Koblenz. From 1989 to 1992 he was with Philips Communication Industry AG in quality control, and with Parsytec Computer GmbH focusing on massive parallel computing. Since 1995 he has been a researcher and Ph.D. candidate at RWTH, Department of Computer Science, responsible for agent technology research projects and working in ACTS projects. Research topics are mobile computing and personal multimedia communications. Activities concerning distributed systems and middleware architectures are mainly in the area of mobile agent technology.

JENS MEGGERS received his Dipl.-Inform. in computer science from RWTH, Germany, in 1995. He joined the Department of Computer Science of the same university as a Ph.D. candidate in computer science in 1995. He participates in various internal and external research activities, including the ACTS project OnTheMove. His main research focus lies in QoS supporting network and transport protocols for mobile multimedia communications.

GUNNAR FORSGREN received his B.S.E.E degree from Härnösand Gymnasium, Sweden, in 1978 and has been with Ericsson in various R&D positions since 1980. His research interests include information/communication services on wireless devices and their interaction with agent services.

ERNÖ KOVACS received his Dipl.-Inform.from the University of Kaiserslautern, Germany in 1991. During 1986–1990 he worked at IBM's European Networking Centre (ENC) in various research projects concerning multimedia e-mail, multimedia documents, and distributed hypermedia systems. From 1991 to 1996 he worked at the Institute of Parallel and Distributed High-Performance Systems (IPVR) of the University of Stuttgart. He conducted several projects in the area of middleware for distributed systems. In 1997 he joined Sony's Research and Development Department in Stuttgart and worked in the ACTS project OnTheMove. His current research interests include mobile multimedia, quality-of-service trading, and mobile agent systems.

MICHAEL ROSINUS received his Dipl.-Inform. at the Universität des Saarlandes, Germany, in 1996 and worked at the German Research Center for Artificial Intelligence (DFKI) in the area of intelligent agents. In May 1996 he joined the Sony Research and Development Department where he is working in the OnTheMove project. His current research interests include mobile and intelligent agents, multimedia data processing, and wireless networks.

Microsoft Corp. Exhibit 1009

Real-time synthetic vision cockpit display for general aviation

Andrew J. Hansen, W. Garth Smith, and Richard M. Rybacki

MetaVR, Inc.
http://www.metavr.com[1]

ABSTRACT

Low cost, high performance graphics solutions based on PC hardware platforms are now capable of rendering synthetic vision of a pilot's out-the-window view during all phases of flight. When coupled to a GPS navigation payload the virtual image can be fully correlated to the physical world. In particular, differential GPS services such as the Wide Area Augmentation System WAAS will provide all aviation users with highly accurate 3D navigation. As well, short baseline GPS attitude systems are becoming a viable and inexpensive solution. A glass cockpit display rendering geographically specific imagery draped terrain in real-time can be coupled with high accuracy (7m 95% positioning, sub degree pointing), high integrity (99.99999% position error bound) differential GPS navigation/attitude solutions to provide both situational awareness and 3D guidance to (auto) pilots throughout en route, terminal area, and precision approach phases of flight.

This paper describes the technical issues addressed when coupling GPS and glass cockpit displays including the navigation/display interface, real-time 60Hz rendering of terrain with multiple levels of detail under demand paging, and construction of verified terrain databases draped with geographically specific satellite imagery. Further, on-board recordings of the navigation solution and the cockpit display provide a replay facility for post-flight simulation based on live landings as well as synchronized multiple display channels with different views from the same flight. PC-based solutions which integrate GPS navigation and attitude determination with 3D visualization provide the aviation community, and general aviation in particular, with low cost high performance guidance and situational awareness in all phases of flight.

Keywords: situational awareness, real-time visualization, correlated terrain databases, geographically specific satellite imagery

1. INTRODUCTION

A remarkable transition in state-of-the-art image generation is taking place as single purpose, specialized rendering hardware is being replaced with off the shelf components driven by PC-based processors. The dramatic performance improvements realized by the PC graphics industry in the last two years has leveraged the broad base of innovation across the industry. The rich mix of focused development efforts in chip design, bus architecture, software driver standards, and processor technology feeds the continuous improvement in PC graphics capability that is reaching the upper echelons of visualization-simulation (VizSim) performance standards. This paper focuses on the underlying capabilities needed to render the virtual environment in a mobile platform such as an aircraft. Primarily these are the image generator hardware and software implementation and the generation of a three-dimensional database of the environment including terrain, aircraft, and cultural features. It does not address symbology and information content that should be displayed and interested readers are referred to [1,2].

These low cost, high performance graphics solutions based on PC hardware platforms are now capable of rendering both moving map displays and synthetic out-the-window views of a moving vehicle with an extremely high degree of realism. By coupling with a GPS navigation/attitude payload the virtual image can be fully correlated to the physical world in real time. In particular, differential GPS services such as the FAA's Wide Area Augmentation System (WAAS) [4] will provide users with highly accurate 3D navigation information. The WAAS position solution is specified to have accuracy better than 7.5m 95% and a guaranteed (99.99999%) confidence interval [5]. Prototype implementations of WAAS are achieving nominal accuracy of about 1-2m 1-sigma in all three dimensions [6]. Carrier phase GPS based attitude heading references system (AHRS) prototypes are also being implemented [7,8,9] which can provide sub degree accuracy in all three axes, roll/pitch/yaw. Integration of accurate position/velocity/attitude state information and a highly capable rendering engine enables synthetic image generation of the physical scene.

[1] Correspondence: Email: {ahansen,wgsmith,rmrybacki}@metavr.com; Telephone: (617)739-2667

Part of the SPIE Conference on Enhanced and Synthetic Vision 1999
Orlando, Florida • April 1999 SPIE Vol. 3691 • 0277-786X/99/$10.00

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

The underlying resource that ties these two pieces together is an accurate and reliable geographic database that describes the physical environment. It must be accessed efficiently to serve the real time application but also have the fidelity to enhance rather than detract from the pilot's situational awareness. Our solution incorporates geographically specific satellite imagery and cultural features into an efficient terrain database. The satellite imagery provides contextual information for situational awareness while specific cultural features such as runways can be inserted as additional objects with higher resolution and special properties. For run time flexibility, the resulting database can be stored in memory or demand paged off of a storage device using a "look ahead" algorithm.. Real time performance for extremely high-resolution terrain (100m post) and imagery (5m) is supported using hierarchical level of detail switching. In addition the render engine has variable field of view and far horizon clipping plane parameters so that the necessary display refresh rates can be maintained. The solution we describe below borrows heavily from the visual simulation concepts developed in distributed interactive simulation (DIS) research with the new twist that high fidelity high performance can be achieved on PC platforms. The economies of scale in this arena provide low cost systems and the impetus to move toward embedded solutions. While all of the features currently supported by 3D VizSim applications may not be appropriate in the cockpit, we identify them here as candidates for use and leave their designation to the community at large.

The remainder of this paper first touches briefly on the linkage of position/velocity/attitude state information and the virtual environment in the computer. We then focus on the visualization hardware and the innovations in the graphics industry which now provide the power to render one or more 3D out-the-window scenes or 2D top down moving maps (so called plan view displays). The next section focuses on our database construction process, database storage requirements, and its correlation to truth. We close with some comments on the opportunity to extend the cockpit display to a networked solution where, given a low bandwidth communication channel, information from multiple entities could be included in the display. In this mode the display could provide additional situational awareness vis-a-vis TCAS I/II systems that have a cockpit display of traffic information (CDTI).

2. LINKING AIRCRAFT STATE TO VIRTUAL ENVIRONMENT

2.1 Navigation: Position, Velocity, and Attitude

In order to place the virtual aircraft at the appropriate position and orientation in the virtual environment, sensor system outputs of the aircraft's position, velocity, and attitude must be available to the graphical render engine in real time. Differential GPS navigation and attitude determination is a low cost option for obtaining these states in the aircraft. The geographical extents covered in aviation applications are well served by wide area differential GPS (WADGPS) systems for real time positioning. Likewise the global coverage of GPS allows a user with multiple antennas to compute an attitude solution at any position within aviation capability.

The FAA is specifically developing the Wide Area Augmentation System (WAAS) for seamless, high integrity navigation in all phases of flight. Successful prototype signal-in-space flight tests have already been implemented and carried out by the FAA Technical Center with Stanford Telecommunications [13] and Stanford University [5]. The WAAS uses a geosynchronous satellite broadcast channel for continental scale coverage and high data link availability. In cooperation with the FAA and industrial representatives, RTCA, Inc. has written the WAAS Minimum Operational Performance Standards [12] (WAAS MOPS) to specify the WAAS signal structure and the application of the differential corrections to stand alone GPS measurements. The WAAS navigation payload includes a GPS receiver capable of receiving an additional 250 bps WAAS data stream from a geosynchronous satellite. The WAAS message stream is unpacked to form differential corrections for satellite clock and ephemeris errors as well as a differential ionospheric correction. These corrections are then applied to the standard GPS measurements for each satellite in view. The differentially corrected signals form the basis for the navigation solution and its associated confidence interval. This navigation solution, which contains both position and velocity, is fed directly to the image generator in the form of WGS84 coordinates.

Synthetic vision applications are very sensitive to errors in attitude determination because the entire field of view is controlled by the orientation of the viewpoint. Low cost AHRS based on carrier phase GPS are now incorporating rate or inertial aiding [7,8] to provide the accuracy and noise performance necessary to drive cockpit displays. Strapdown AHRSs are also shrinking the antenna baselines needed to achieve sub degree accuracy in all three directions [9]. The resulting attitude solution in body coordinates can be input directly to the image generator. Adequate systems require an update rate on navigation and attitude of at least 10 Hz [1] in order to reach suspension of disbelief for the operator. Of course the faster the better, but in any case if the sensor inputs do not update at the frame rate of the display system a model of the aircraft is propagated forward to update the synthetic vision viewpoint to maintain the 60 Hz visual update rate.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

In flight, updating the aircraft model which must reside in the same coordinate system as the terrain database requires a transformation of the navigation solution from WGS84 coordinates to the local coordinates of the database. This does require some additional but necessary computation. The virtual environment database needs sufficient fidelity to fully

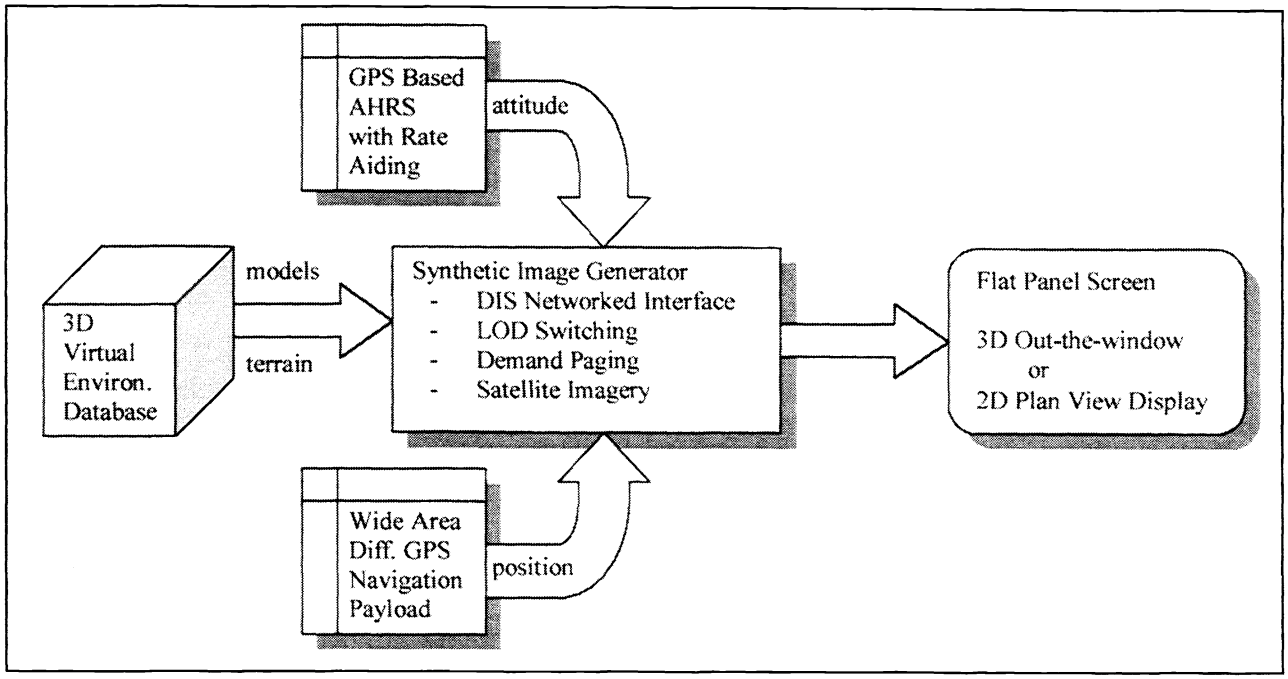


Figure 1. The image generator ingests the 3D-database and real time data from position/attitude sensors to determine the viewpoint for the synthetic scene and pushed onto graphics card for rendering.

correlate the sensor states with the terrain in the rendered image. Spheroidal coordinate systems such as WGS84 cannot provide that level of fidelity and the database must use local coordinate system based on a geoid.

2.1 Distributed Interactive Simulation Protocol

As briefly mentioned above the sensor states may need to be propagated forward some number of epochs as the render engine my update faster than the navigation and attitude updates are available. Our implementation abstracts the input linkage from the sensor systems to the render engine. We utilize the IEEE standardized [14] Distributed Interactive Simulation (DIS) protocol for inserting new sensor updates into the virtual environment. This DIS protocol exists at the application layer of the communications stack. It is built upon User Datagram Protocol (UDP) packets called Protocol Data Units (PDUs). These PDUs are well defined in the DIS standard and include necessary elements such as kinematic model parameters as well as graphical information in the form of texture and polygonal models.

One added benefit to the DIS approach is that multiple views from the same entity can be added simply by plugging in another render engine. Another, and we believe more powerful benefit, is that multiple entities can appear in the same virtual environment exactly as they do in the physical environment. An entire suite of functionality including multicasting, loss tolerance, forward state prediction, and communication protocol is already defined and implemented by the VizSim community. DIS is abstracted from the physical layer so that the network could be a high speed wired intranet or just as easily a low bandwidth wireless LAN so far as the application is concerned. In fact, DIS is expressly designed for a heterogeneous network where some paths have much greater bandwidth than others. This flexibility has direct benefits for future applications that include multiple entities (other air traffic).

An additional feature of the DIS network solution is the ability to log PDU packets being transmitted. By logging the state information in PDU form, the entire flight can be captured for playback. This is particularly useful for experimental or

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

testing purposes in early development of operational systems to replay and view from any vantage point using a six-degree-of-freedom pointing device.

3. IMAGE GENERATION

The image generator is the core of the cockpit display. As shown in Figure 1 it ingests the terrain database and aircraft models and accepts input from the navigation and attitude payloads. One or more graphics engines slaved to the image generator can then render images to the screen with one viewpoint for each engine. The baseline mode is a single 3D out-the-window view out the front of the aircraft showing the terrain and cultural features in the environment. Adding an additional graphics engine or switching modes to the plan view display provides the pilot with a top down moving map. This approach is very appealing as it can immediately utilize advances in hardware performance offered by the industry as it continues to improve.

The image generator is currently hosted on a PC platform. It requires a graphics card that supports the DirectX API and can utilize either the PCI or AGP bus as the graphics pipeline. A host platform consisting of a 450MHz Pentium II processor with 512Mb of RAM and a Canopus Spectra 2500 with 16Mb of VRAM consistently maintains 60Hz frame rates for fields of view covering a 50 km radius at velocities up to Mach 3. Note that these frame rates are significantly higher than the update rates that the navigation/attitude sensors support. An important consideration in the software development of the image generator was the graceful degradation in performance as either the host platform is scaled back or the fidelity of the database is scaled up. We have already implemented laptop PCs rendering 3D out-the-window views of the virtual world.

DirectX and OpenGL capable graphics cards are designed to render polygonal shapes as triangular patches in hardware. Textures may also be stored in memory and applied to these polygons as part of the hardware processing of the render engine. The image generator is responsible for pushing the textures up into video memory and then pipelining the polygonal shapes from the terrain database up to the graphics card using in our case either the PCI or AGP bus under the DirectX protocol. As such there is a balance that needs to be struck to ensure that the central processor and the graphics chip set are reasonably well matched in performance. The CPU must index and arrange the terrain polygons based on the current aircraft state and the graphics card is responsible for rendering the textured polygons.

The importance of the level of detail (LOD) switching and demand paging now becomes clear. LOD switching aids in balancing the load between the CPU and graphics card. If the current field of view has too many textured polygons for the graphics card to handle then the CPU can switch some of the far field regions to lower resolution and thereby reduce the number of polygons being rendered. This LOD switching is an improvement over the simplest form of switching which is the insertion of a clipping plane that limits the field of view. To accomplish LOD switching the image generator and database must be intimately coupled as not only does the polygon resolution switch but also the textures applied to them. For platforms that have memory limitations the image generator can invoke demand paging of regions of the database that are coming into the field of view. Knowing velocity states allows the image generator to look ahead in the database to see if upcoming regions are loaded into memory and ingest them in the event that they are not.

The core process in the image generator is to continuously update the viewpoint of the virtual aircraft at each epoch. Under the DIS paradigm the aircraft state is propagated forward from the last PDU update. In the host platform this is nominally never longer than 20 msec. The local region of the terrain database which is stored in memory is then interrogated to assemble the textured polygons for pipelining up to the graphics renderer. If other entities besides the host aircraft are in view they are also updated and their virtual representation is pushed up the graphics pipeline.

There are many other features available from VizSim applications that are probably not useful in the cockpit such as variable visible spectrum, DirectSound output, atmospheric emulation of fog and clouds, and six degree of freedom input device compatibility. However, the existence of these features demonstrates the head room available in this implementation which can be converted into other more pertinent features such as situational awareness symbology, tunnel-in-the-sky guidance [2], and traffic information.

Neglecting the navigation and attitude platforms, the hardware necessary for the image generator is currently on the order of $4000. Once available, projections for WAAS and AHRS system prices are in the ones of thousands of dollars. This places hardware costs for first generation integrated cockpit displays at around $10-15k plus installation. Our expectation on cost trends is that they would follow the precedence set in the rest of the VizSim market: continual improvement in the price/performance point at market. There are also certification and recurring costs associated with any avionics systems which we do not have sufficient experience to comment on with the exception of generating and updating the 3D databases.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

Because of the proliferation of the underlying resources (elevation data and satellite imagery) and competing utilities for database construction we also expect the database generation costs to decline. The ultimate goal for a cockpit display is the integration of the attitude/navigation/renderer into a single embedded system which could drive a flat panel display

4. TERRAIN AND IMAGERY RESOURCES

Terrain information is typically available in raw form as digital elevation maps (DEMs) or digital terrain elevation data (DTED) at various levels of resolution. In order to create a terrain database that can be rendered on a computer this information must be converted into polygonal surfaces that represent the surface of the terrain. These polygonal elements are well suited to image generation as the industry has optimized graphics chip sets to handle them in hardware. This optimized hardware is now readily available on PC platforms. One of the most important considerations in this conversion is the need for extreme efficiency in both the size and accessibility of the resulting database so that the image generator can ingest and render the virtual scene. For even medium fidelity terrain information, say 125m, post, and a reasonable coverage area for aviation, say 5° x 5° cells, the raw data for elevation information alone can run into the hundreds of megabytes in size. We defer the discussion of our conversion process and the resulting database to the next section.

Another important part of generating a convincing image for the user is the texture overlay on the terrain surface. Our approach is to apply geographically specific satellite imagery on the terrain polygons. By overlaying real imagery that is coordinated directly to the terrain data, the scene that is eventually rendered by the visualization engine has a very high degree of realism. The sources for satellite imagery are increasing rapidly and we anticipate that the vast majority if not all of the national air space (NAS) will be covered and in fact frequently and continuously renewed with world wide coverage soon to follow. Even at this time custom imagery for any particular location can be ordered directly off of the World Wide Web from commercial vendors.

Our secondary approach to overlays follows the standard approach of texture mapping each surface. Here synthetic textures are created and used in place of the satellite imagery where it is not available. In either case, real imagery or synthetic textures, rendering of terrain polygons is treated exactly the same by the render engine as the database is responsible for arbitrating the virtual world including the overlays.

There are several sources of elevation maps in digital form. NIMA outputs Digital Terrain Elevation Data (DTED) at various levels of resolution, typically only the lowest level is openly available. The USGS supplies elevation data in the form of Digital Elevation Maps (DEMs) which can be purchased on the web. ERDAS Imagine, CTDB, and other formats commonly used for GIS applications are also available commercially. We utilize primarily DTED, CTDB, and soon DEM data formats as the raw elevation resource for constructing the terrain database.

Satellite an aerial imagery is the other commodity we rely on for generating high fidelity databases. The commercial availability of high-resolution geographically specific imagery is growing. Individual providers are already offering photo to order imagery purchases over the web. We have already worked with products from ImageLinks in the 10-50m range. Although not openly available classified customers also have access to high resolution (1m) satellite imagery from NIMA. The important and necessary condition of the imagery is that it be geographically specific, that is ortho-rectified and pin-pointed to a reference in a standard coordinate system. This real imagery can then be draped onto the terrain surface and replace older approaches using synthetic textures.

For application specific information such as that needed for aviation, cultural features may be inserted into the database as explicit objects. An example is the runway and markings at a specific airport. These types of features can be created in a number of different formats. The industry standard is the OpenFlight format from MultiGen, Inc. but many other new and heritage formats such as VMAP, AGRD formats. The importance of the OpenFlight format is that it also supports models for dynamic entities such as aircraft. We invoke the OpenFlight format to ingest cultural features generated from graphics and modeling tools available in the industry as well as the ARDG format for cultural features on the plan view display. Although some models are already available commercially, most of the creation of cultural features and object models is carried out on a specific project basis. We see this approach eventually converging to a pool of models openly available to the community.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

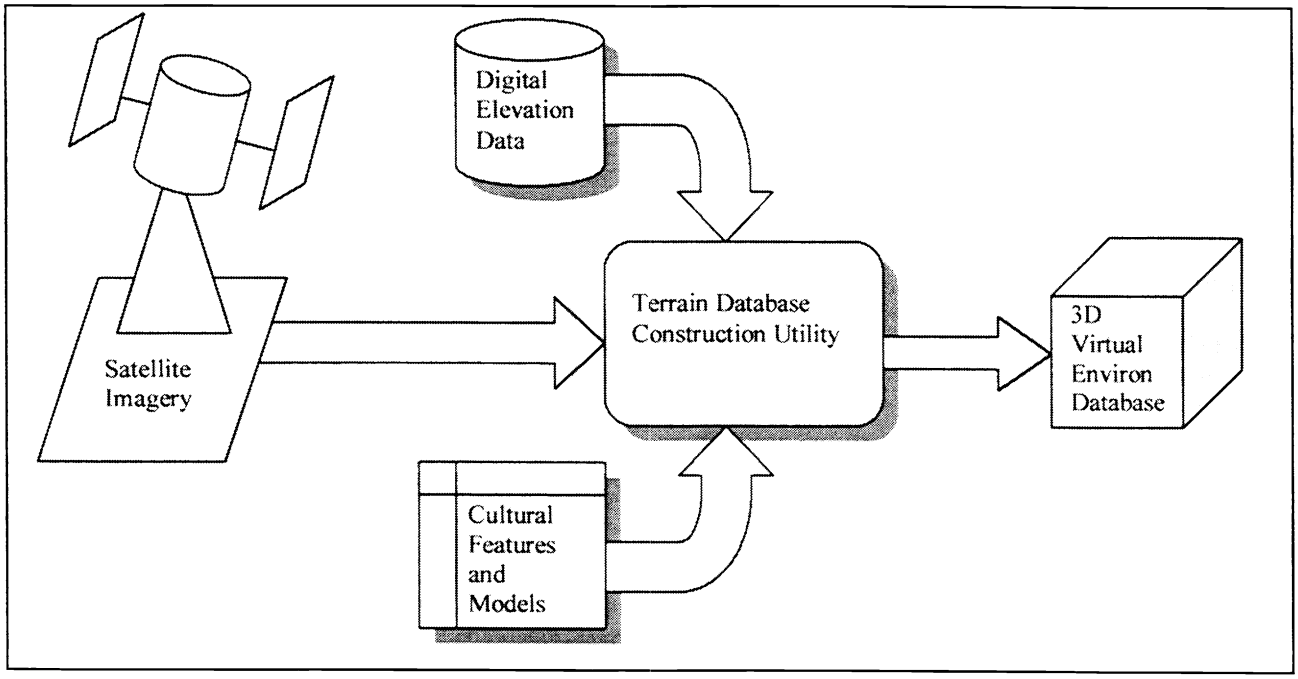


Figure 2. The database construction process assembles a 3D environmental database suitable for input to the image generator from three basic resources: digital elevation data, geospecific imagery, and designated cultural features.

5. DATABASE CONSTRUCTION

The elegance of our terrain database construction which uses geographically specific imagery is that it provides a real world source for constructing cultural features such as buildings, road networks, vegetation, bridges, even power grids if the image resolution is high enough. We are currently developing a palette base utility for constructing 3D terrain databases that integrate these three elements, digital elevation maps, geographically specific imagery, and automated cultural feature generation into one umbrella application. This utility will be capable of directly feeding the image generator during database design for viewing the construction on the fly. As such it will provide a suitable facility for mission planning, mission briefing, and mission rehearsal. The current database construction utility functions as a wizard type application which allows the user to enter raw data resource file names and then automatically generates the resulting database.

The three fundamental components of our database construction are the ingest of the digital elevation information, ingest of bitmap based geographically specific imagery, recognition and conversion of imagery details into cultural features (not yet available), and export to any of four database formats: MDB, MDX (both MetaVR specific), CTDB, and OpenFlight. To support the highest levels of image rendering, the MDX database format supports hierarchical levels of detail with switching controlled by range thresholds on both terrain and textures. This allows the central processor in the image generator to match the rendering capabilities of the highest end graphics cards that have 180Mpixel fill rates. The database format also allows terrain information to be loaded by the render engine incrementally using look-ahead demand paging.

The most important consideration is full correlation between entity state coordinates and the terrain database. As noted by Barrows [2] and Ourston [10], lack of correlation in some current implementations is unacceptable, particularly in exercises as the virtual scene is not convincing and detracts heavily from the qualitative performance of the simulation. MetaVR's

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

MDB and MDX database formats are fully correlated terrain databases that eliminate any such anomalous behavior. By using a local geoid coordinate system and transforming the WGS84 coordinate aircraft states in the image generator the entities are guaranteed to be consistent with the terrain. This of course does not mitigate errors due to the level of resolution for the terrain elevation data.

The raw elevation data is tessellated into triangular patches [11] using a Delaunay triangulation where the vertices of the triangluation are the locations of the elevation data in the local coordinate system. This is a fast routine and is computed repeatedly on sub sampled intervals to generate various levels of detail. The geographically specific imagery is then cut into patches corresponding to the levels and indexed to the appropriate triangular surfaces.

The incorporation of cultural features is currently supported on an internal format (MetaVR CLT). Development of a VMAP capable module is in process to support interoperable ingest/export of cultural features. This will provide a clear path for an imagery to cultural feature format that is sharable. We envision the downstream capability, given adequate imagery, to professionally construct and modify scenarios for training and planning.

The 3D virtual environment database also contains the models of physical entities, e.g. aircraft. These are critical for the image generator as they provide the mechanism by which the virtual scene can be propagated at very high rates (60 Hz) for rendering to the screen. At each epoch, if new state information is not available from the navigation or attitude subsystem, the states are propagated according to the properties specified in the aircraft model. This of course leads to models which are specific to the type of aircraft being simulated, e.g. Cessna 152 versus Boeing 737, in order to capture the pertinent physical properties. The inclusion of such models is particularly important if one desires to render other aircraft in the virtual scene as we describe in the next section.

The following four figures depict the underlying terrain database and its full rendering with the satellite imagery overlay. Figures 3 and 4 show a relatively flat region with the satellite imagery capturing the road network and surrounding buildings. The contextual information in the satellite imagery provides very strong situational awareness that is not available in texture mapping and extremely user intensive to design by hand in graphical models. The pair of images in Figure 5 is the wire frame and full imagery of a coastal region in Alaska, Prince William Sound, and demonstrate the LOD capability. The geographically specific satellite imagery is 25m at highest resolution. Figure 6 is a screen capture of the 2D plan view display mode of the graphical render engine. It is most useful in applications that require a moving map display with a great deal of cultural information and possibly other traffic information for situational awareness.

6. POTENTIAL FOR COLLISION AVOIDANCE APPLICATIONS

The network capable image generator described in Sections 3 and 4 provides the possibility of displaying other entities on the cockpit display to realize a CDTI. That is, given a low bandwidth communications channel upon which other aircraft could transmit time tagged state information the display could render those aircraft on the display. The DIS protocol mentioned above is well suited for such an application because it codifies the packet format and content necessary to propagate entities in the virtual environment. Indeed this was its designated purpose in simulated training applications for the U.S. military where it originated.

In the current application each aircraft would broadcast its identity, type, and state information over a given communication channel, eventually say the automatic dependent surveillance ADS-B data link. Gazit [11] gives general overview of this type of improved aircraft tracking and avoidance as well as the data link implications. The image generator would have an internal model of all other aircraft types or at least the capability to request and incorporate such a model. An instantiation of any one of these models may be propagated by the image generator for each unique aircraft broadcasting state information intermittently in the local region. The high level of fidelity already available in aircraft models reduces the bandwidth burden of updating other entities in the virtual environment. Unlike the host aircraft whose state information must be tightly coupled to the image generator, other traffic would require much lower update rates to realistically render their modeled entities.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

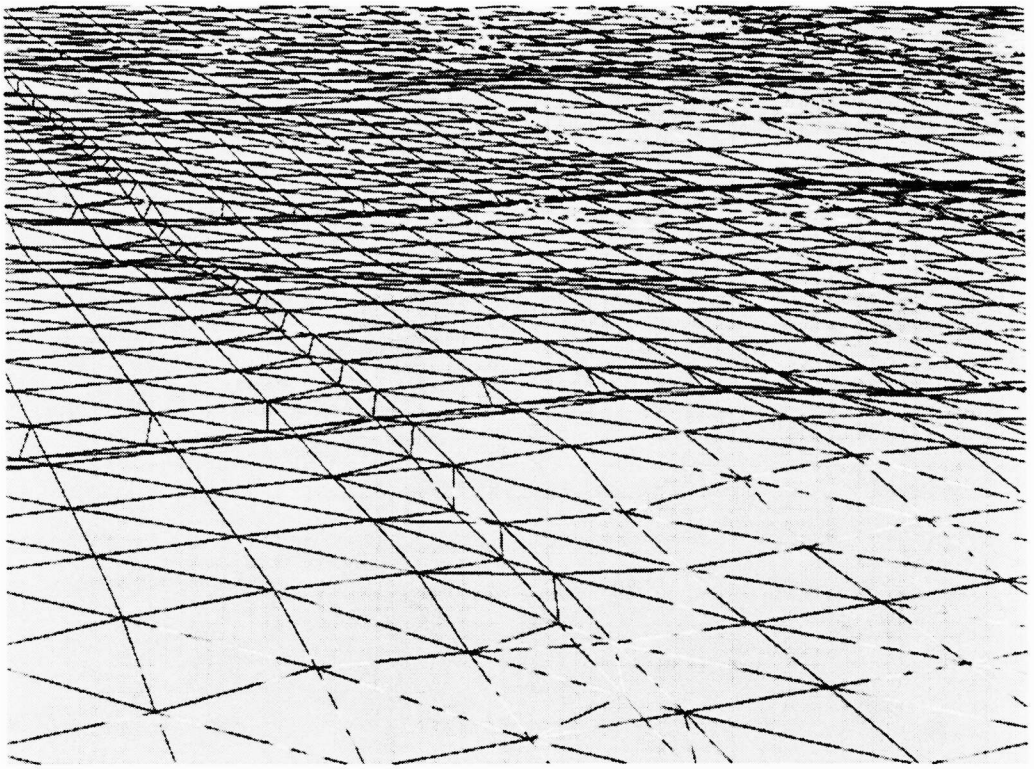


Figure 3. The wire frame image of the underlying terrain polygons shows the varying levels of detail that are stored within the virtual environment database. This image is a screen capture from the real time image generator.

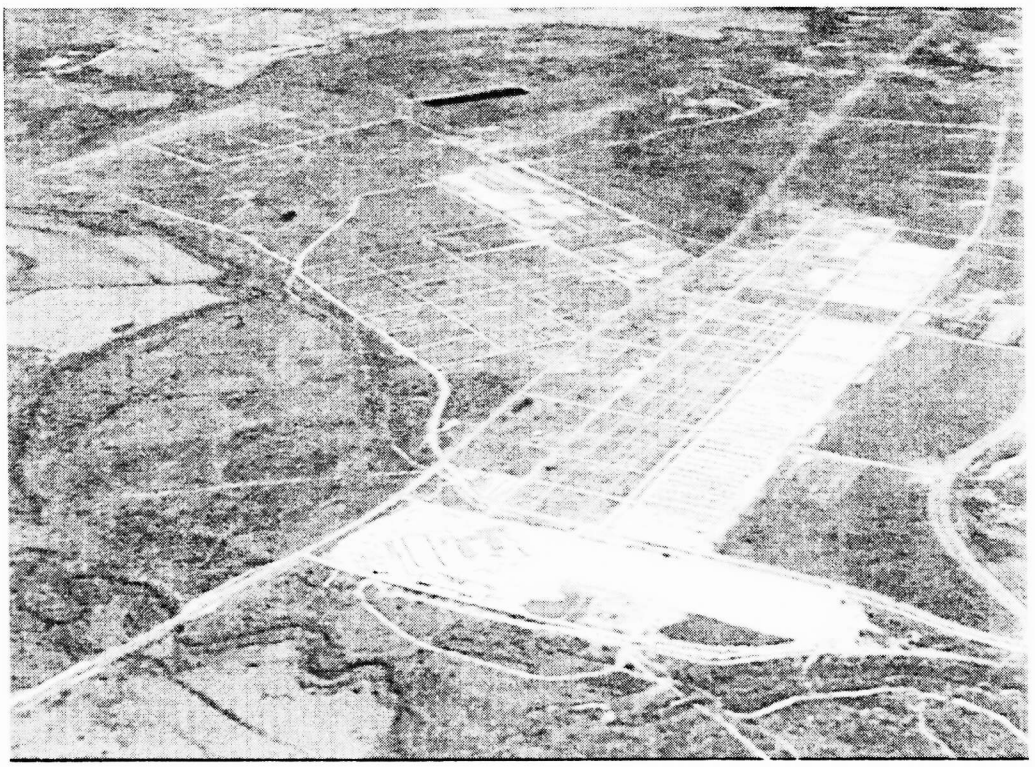


Figure 4. Geographically specific satellite imagery is applied to the terrain polygons in patches. Multiple patch sizes are encoded into the database for varying levels of detail. This image is a B/W screen capture from the real time image generator.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

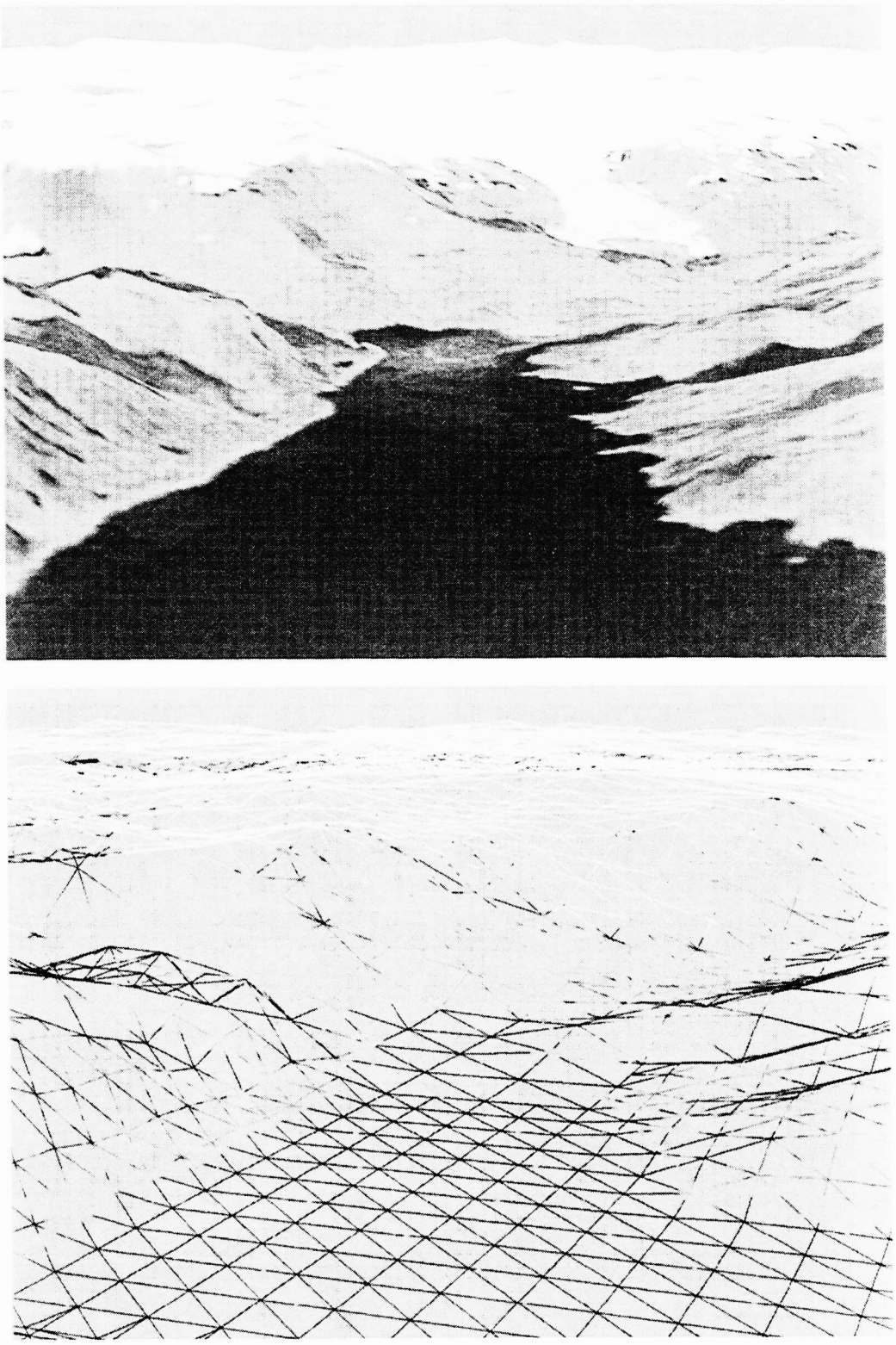


Figure 5. Terrain with extreme amounts of structure can be accommodated with high fidelity. The bottom graphic is a wire frame image of the Alaskan coastline. On the top is the fully rendered scene with imagery.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

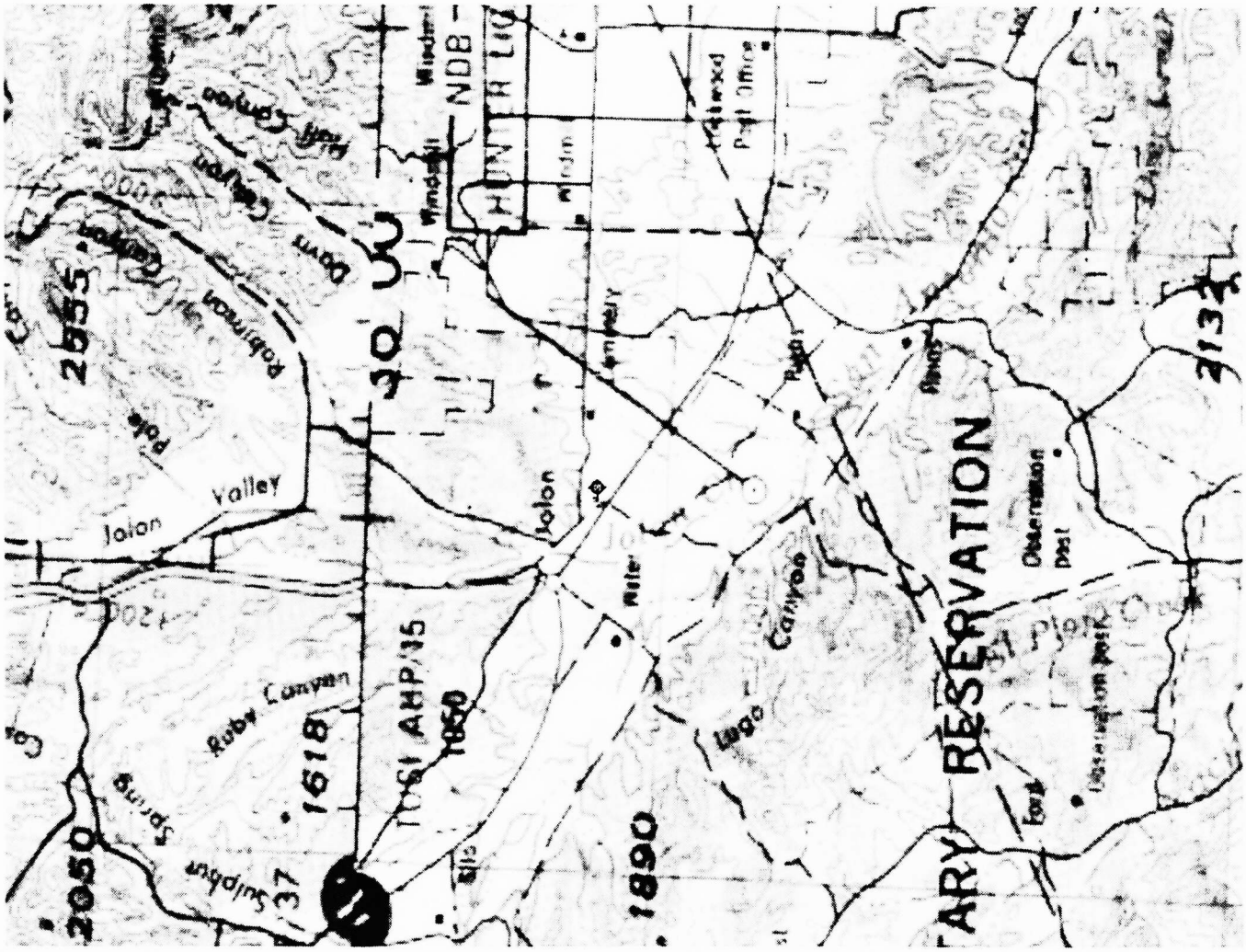


Figure 6. Top down views of additional cultural features are also possible using the plan view display mode. This image displays a moving map that is shifting underneath the aircraft's viewpoint.

7. CONCLUSIONS

The integration of fully capable low-cost image generators, high fidelity terrain databases, and differential GPS navigation/attitude determination provides a viable path to the production of 3D glass cockpit displays for aviation applications, and general aviation in particular. The end goal of such a system is ultimately to aid the pilot by providing enhanced situational awareness. We have described here the basic components of the underlying system: low cost/high accuracy navigation and attitude sensors that are reliable, fully capable image generators that degrade gracefully, and high fidelity virtual environment databases that have complete correlation with the navigation system.

In the fullness of time the FAA's WAAS will provide high accuracy navigation solution with integrity to all equipped aircraft. The continuous, incremental improvement in PC graphics capability will, we predict, push this type of prototype implementation into the realm of an embedded system. At that point the economies of scale would again dramatically reduce the cost of an integrated solution.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

8. REFERENCES

1. Barrows, A., K. Alter, P. Enge, B. Parkinson, and J. Powell, "Operational experience with and improvements to a tunnel-in-the-sky display for light aircraft", ION GPS'97, September 1997.
2. Barrows, A., P. Enge, B. Parkinson, and J. Powell, "Flying curved approaches and missed approaches: 3-D display trials onboard light aircraft", ION GPS '96, September 1996.
3. Alter, K., A. Barrows, C. Jennings, P. Enge, and D. Powell, "3-D cockpit displays for general aviation in mountainous terrain", ION GPS'98, September 1998.
4. Enge, P., et. al., "Wide area augmentation of the Global Positioning System", *Proceedings of the IEEE*, **84**, #8, 1996.
5. Comp, C., et. al., "Demonstration of WAAS aircraft approach and landing in Alaska", ION GPS'98, September 1998.
6. Walter, T., P. Enge, and A. Hansen, "A proposed integrity equation for WAAS MOPS", ION GPS'98, September 1998.
7. Teague, H., "Low-cost GPS/inertial attitude and heading reference system (AHRS) for EV/SV applications", *Proceedings of SPIE*, **3691**, Aerosense/Enhanced and Synthetic Vistion, April 1999.
8. Gebre-Egziabher, D., R. Hayward, and J. Powell, "A low-cost GPS/inertial attitude heading reference system (AHRS) for general aviation applications", IEEE PLANS '98, April 1998.
9. Hayward, R., and J. Powell, "Real time calibration of antenna phase errors for ultra short baseline attitude systems", ION GPS'98, September 1998.
10. Ourston and Reece, "Issues Involved with Integrating Live and Artificial Virtual Individual Combatants", Simulation Interoperability Workshop, March 1998.
11. Hansen, A., and P. Levin, "On conforming Delaunay mesh generation", *Advances in Engineering Software*, **4**, #2, 1991.
12. Gazit, R., *Aircraft Surveillance and Collision Avoidance Using GPS*, PhD thesis, Stanford University, 1996.
13. RTCA Special Committee 159, *Minimum Operational Performance Standards for Airborne Equipment Using Global Positioning System/Wide Area Augmentation*, RTCA/DO-229 Change 3, RTCA, Inc., November 1997.
14. Pogorelc, S., et. al., "Flight and static test results for NSTB", ION GPS '96, September 1996.
15. IEEE, "DIS Exercise Management and Feedback—Recommended Practice", IEEE Standard 1278.3, Institute of Electrical and Electronic Engineers, Inc., 1995.

Microsoft Corp. Exhibit 1009

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 05/06/2015 Terms of Use: http://spiedl.org/terms

US005760783A

United States Patent [19]

Migdal et al.

[11] Patent Number: 5,760,783

[45] Date of Patent: Jun. 2, 1998

[54] **METHOD AND SYSTEM FOR PROVIDING TEXTURE USING A SELECTED PORTION OF A TEXTURE MAP**

[75] Inventors: **Christopher Joseph Migdal**, Mt. View; **James L. Foran**, Milpitas; **Michael Timothy Jones**, Los Altos; **Christopher Clark Tanner**, San Jose, all of Calif.

[73] Assignee: **Silicon Graphics, Inc.**, Mountain View, Calif.

[21] Appl. No.: **554,047**

[22] Filed: **Nov. 6, 1995**

[51] **Int. Cl.**[6] **G06T 11/00**

[52] **U.S. Cl.** **345/430**

[58] **Field of Search** 395/128–132, 395/125–127; 345/430

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,727,365	2/1988	Bunker et al.	340/728
4,974,176	11/1990	Buchner et al.	364/522
5,097,427	3/1992	Lathrop et al.	395/130
5,490,240	2/1996	Foran et al.	395/130

FOREIGN PATENT DOCUMENTS

0 447 227 A2	9/1991	European Pat. Off. .
0 513 474 A1	11/1992	European Pat. Off. .

OTHER PUBLICATIONS

Blinn, Jim, "Jim Blinn's Corner: The Truth About Texture Mapping," *IEEE Computer Graphics & Applications*, Mar., 1990, pp. 78–83.

Foley et al., "17.4.3 Other Pattern Mapping Techniques," *Computer Graphics: Principles and Practice*, 1990, pp. 826–828.

Cosman, M., "Global Terrain Texture: Lowering the Cost," *Proceedings of the 1994 Image VII Conference*, Tempe, Arizona: The Image Society, pp. 53–64.

Dungan, W. et al., "Texture Tile Considerations for Raster Graphics," *Siggraph '78 Proceedings* (1978) pp. 130–134.

Economy, R. et al., "The Application of Aerial Photography and Satellite Imagery to Flight Simulation," pp. 280–287.

Foley et al., *Computer Graphics Principles and Practice*, Second Edition, Addison–Wesley Publishing Company, Reading, Massachusetts (1990), pp. 742–743 and 826–828.

Watt, A., *Fundamentals of Three–Dimensional Computer Graphics*, Addison–Wesley Publishing Company, USA (1989), pp. 227–250.

Williams, L., "Pyramidal Parametrics," *Computer Graphics*, vol. 17, No. 3, Jul. 1983, pp. 1–15.

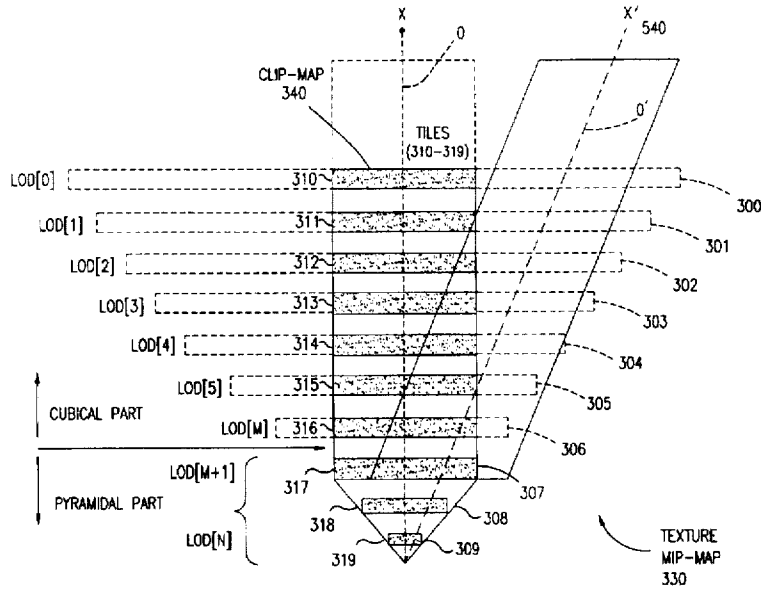
Primary Examiner—Almis R. Jankus

Attorney, Agent, or Firm—Sterne, Kessler, Goldstein & Fox P.L.L.C.

[57] **ABSTRACT**

An apparatus and method for quickly and efficiently providing texel data relevant for displaying a textured image. A large amount of texture source data, such as photographic terrain texture, is stored as a two-dimensional or three-dimensional texture MIP-map on one or more mass storage devices. Only a relatively small clip-map representing selected portions of the complete texture MIP-map is loaded into faster, more expensive memory. These selected texture MIP-map portions forming the clip-map consist of tiles which contain those texel values at each respective level of detail that are most likely to be mapped to pixels being rendered for display based upon the viewer's eyepoint and field of view. To efficiently update the clip-map in real-time, texel data is loaded and discarded from the edges of tiles. Attempts to access a texel lying outside of a particular clip-map tile are accommodated by utilizing a substitute texel value obtained from the next coarser resolution clip-map tile which encompasses the sought texel.

28 Claims, 14 Drawing Sheets



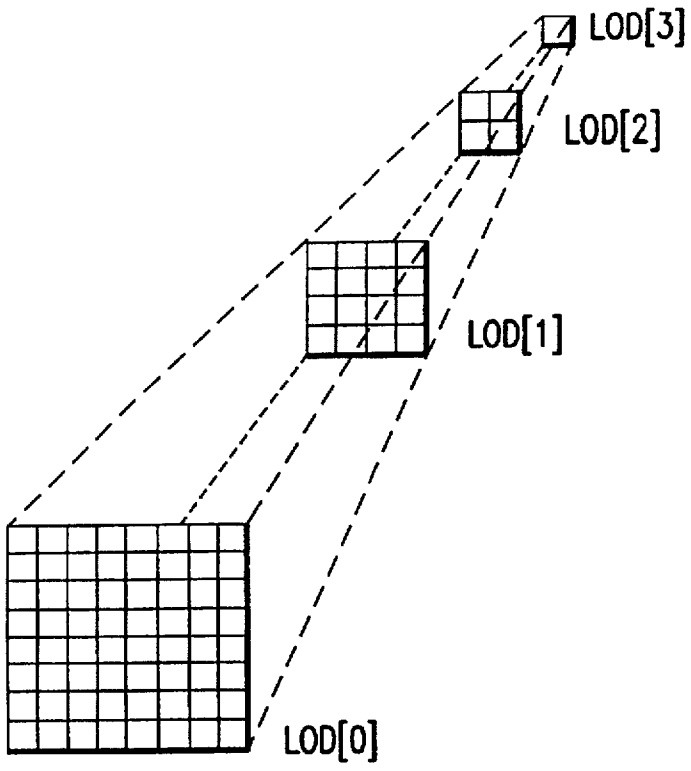


FIG.1A

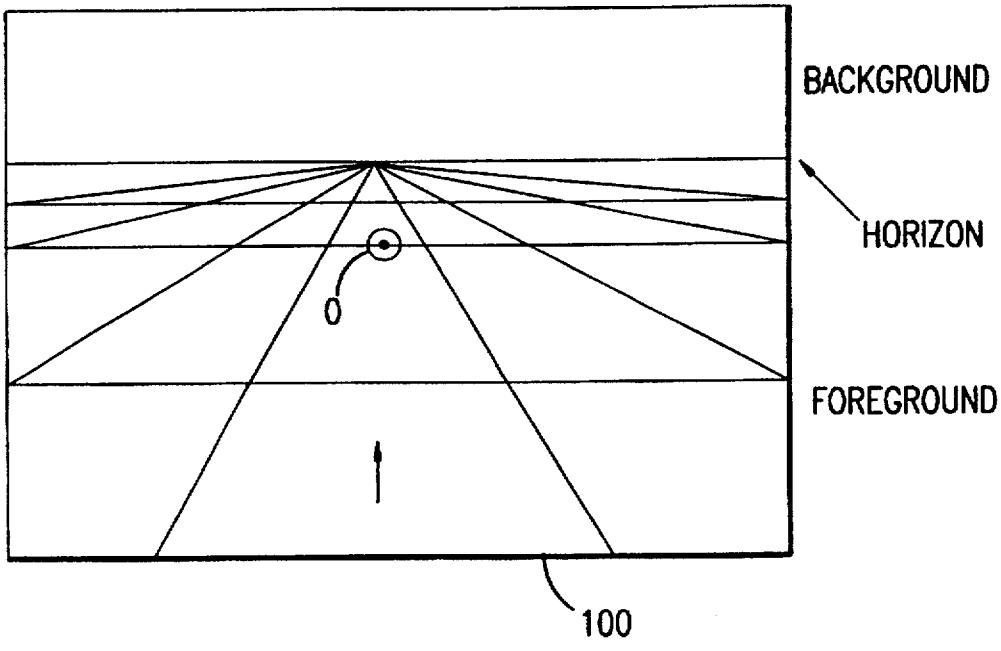


FIG.1B

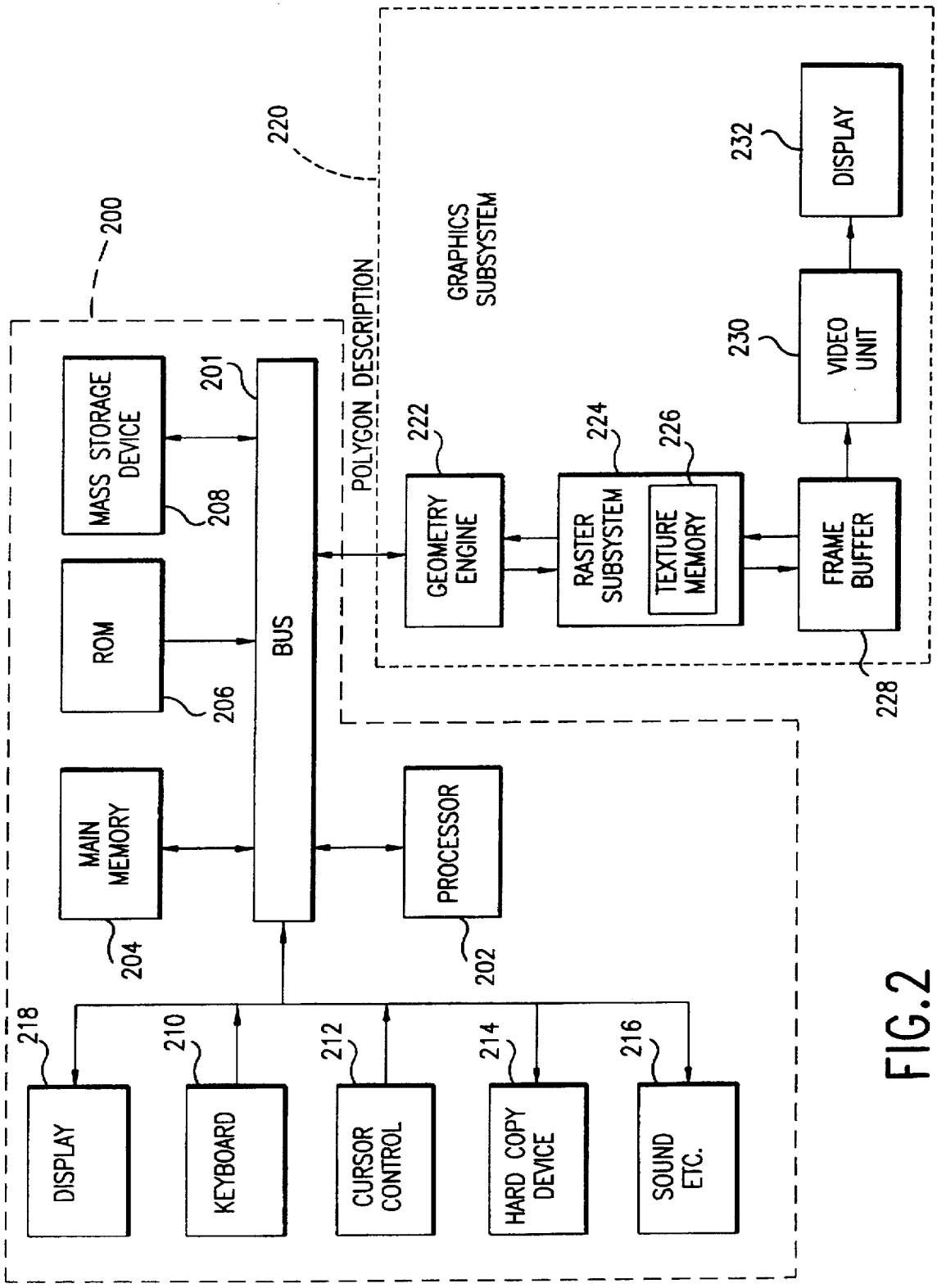

200
220
GRAPHICS SUBSYSTEM
POLYGON DESCRIPTION
MASS STORAGE DEVICE
201
208
ROM
206
BUS
MAIN MEMORY
204
PROCESSOR
202
GEOMETRY ENGINE
222
RASTER SUBSYSTEM
224
TEXTURE MEMORY
226
FRAME BUFFER
228
VIDEO UNIT
230
DISPLAY
232
DISPLAY
218
KEYBOARD
210
CURSOR CONTROL
212
HARD COPY DEVICE
214
SOUND ETC.
216

FIG.2

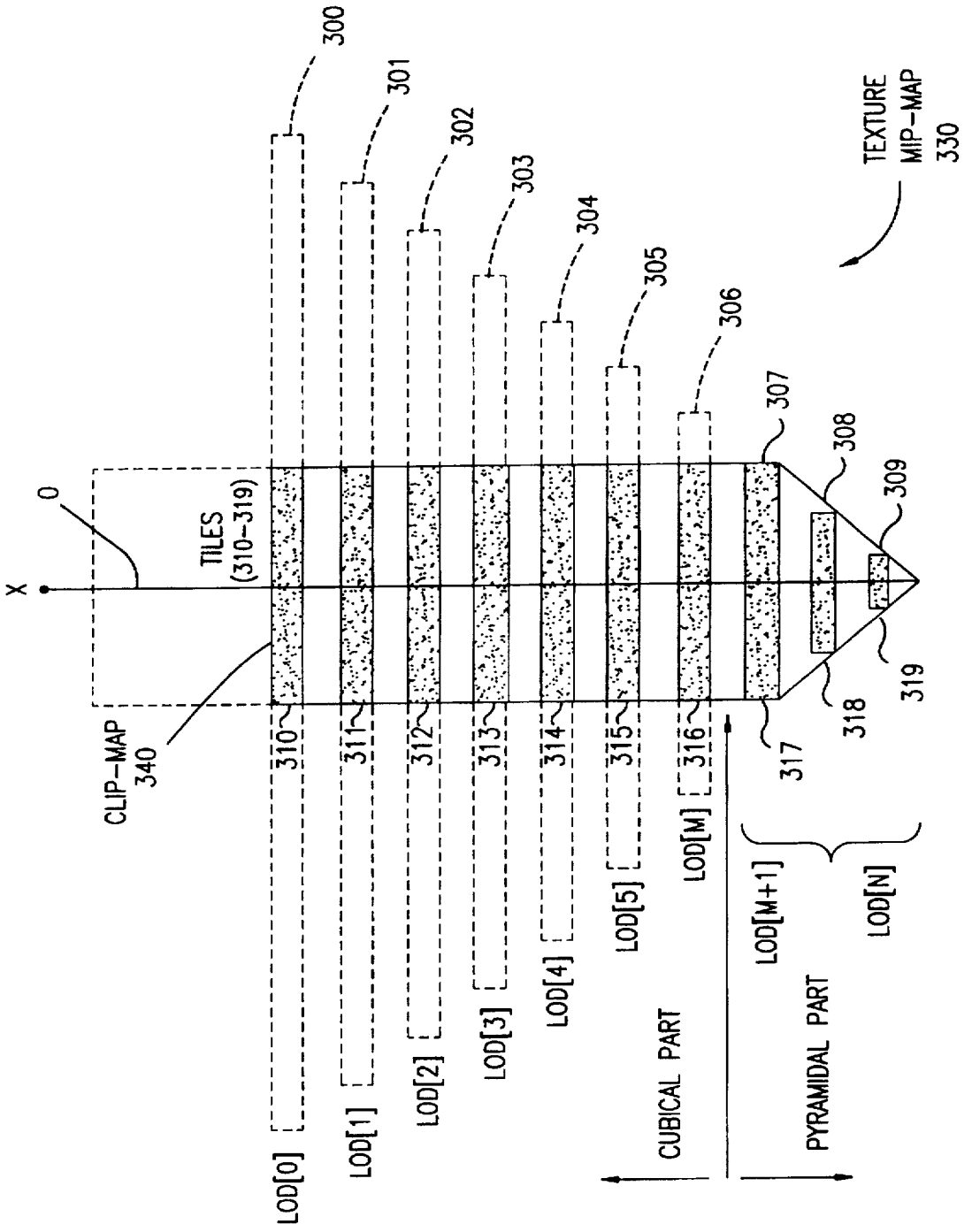

X
0
TILES
(310–319)
CLIP–MAP
340
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
TEXTURE
MIP–MAP
330
LOD[0]
LOD[1]
LOD[2]
LOD[3]
LOD[4]
LOD[5]
LOD[M]
LOD[M+1]
LOD[N]
CUBICAL PART
PYRAMIDAL PART

FIG.3

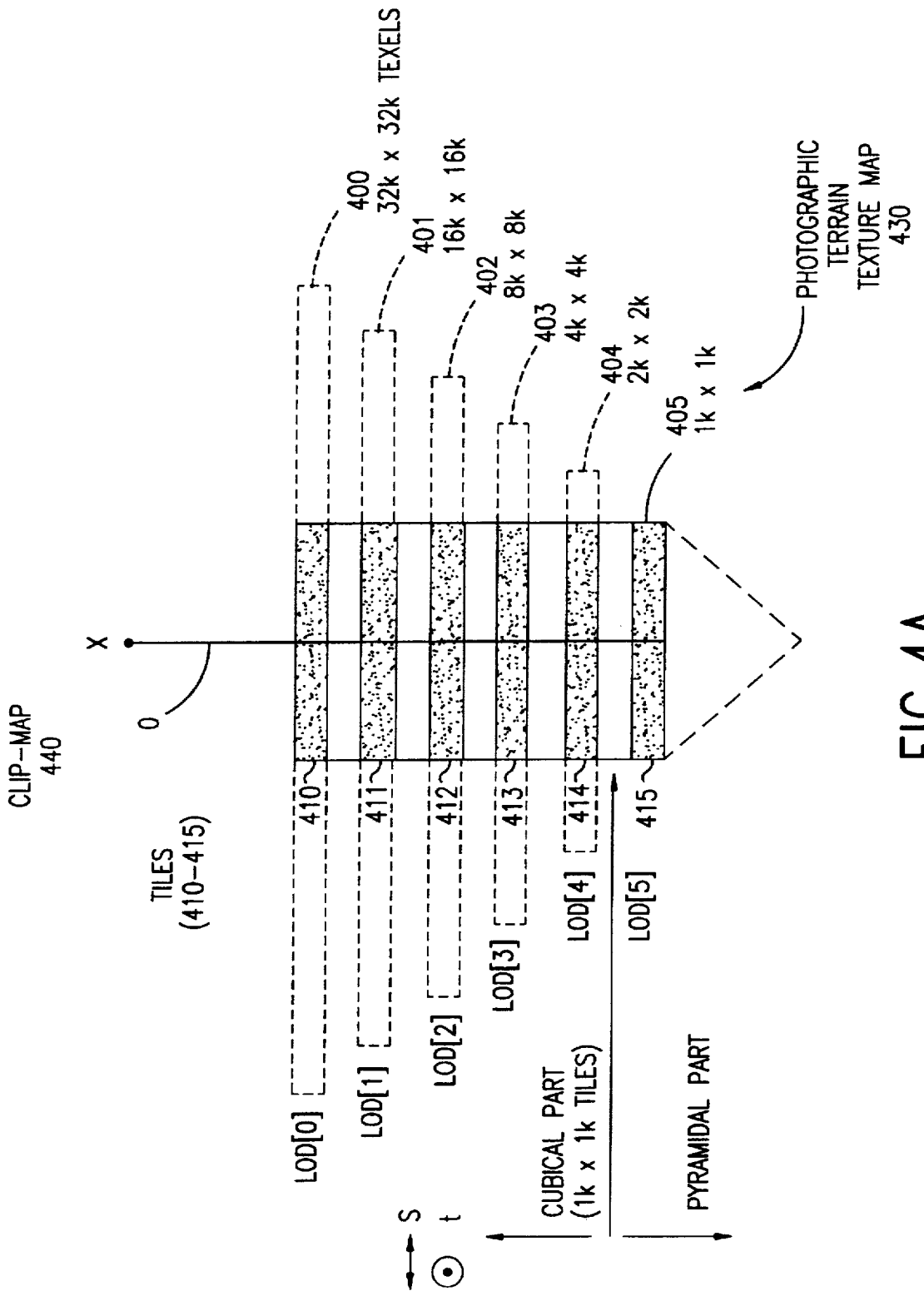

CLIP-MAP
440
X
0
TILES
(410-415)
400
32k x 32k TEXELS
401
16k x 16k
402
8k x 8k
403
4k x 4k
404
2k x 2k
405
1k x 1k
PHOTOGRAPHIC
TERRAIN
TEXTURE MAP
430
410
411
412
413
414
415
LOD[0]
LOD[1]
LOD[2]
LOD[3]
LOD[4]
LOD[5]
CUBICAL PART
(1k x 1k TILES)
PYRAMIDAL PART
S
t

FIG. 4A

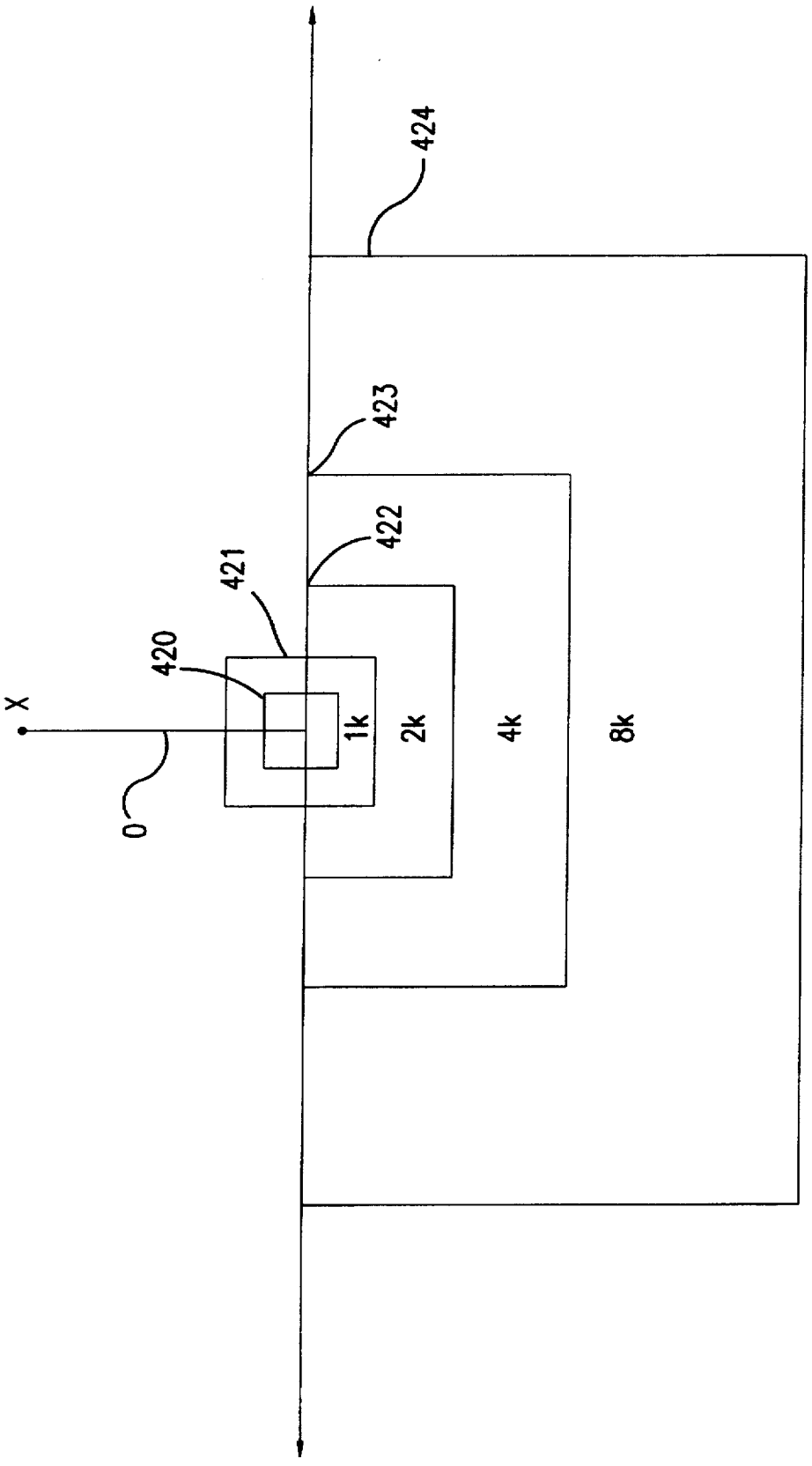


FIG.4B

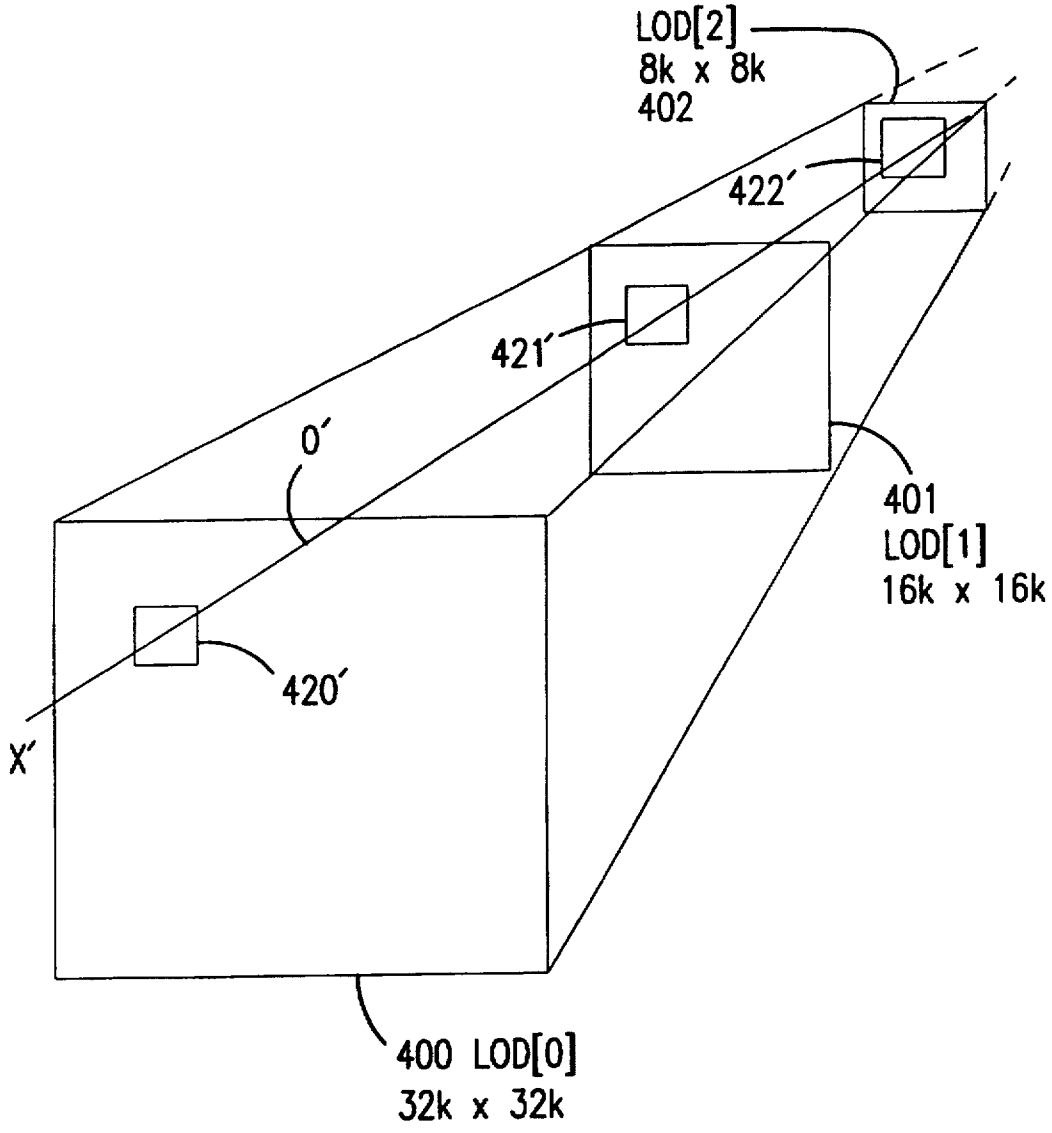

LOD[2]
8k x 8k
402
422'
421'
0'
401
LOD[1]
16k x 16k
420'
X'
400 LOD[0]
32k x 32k

FIG.4C

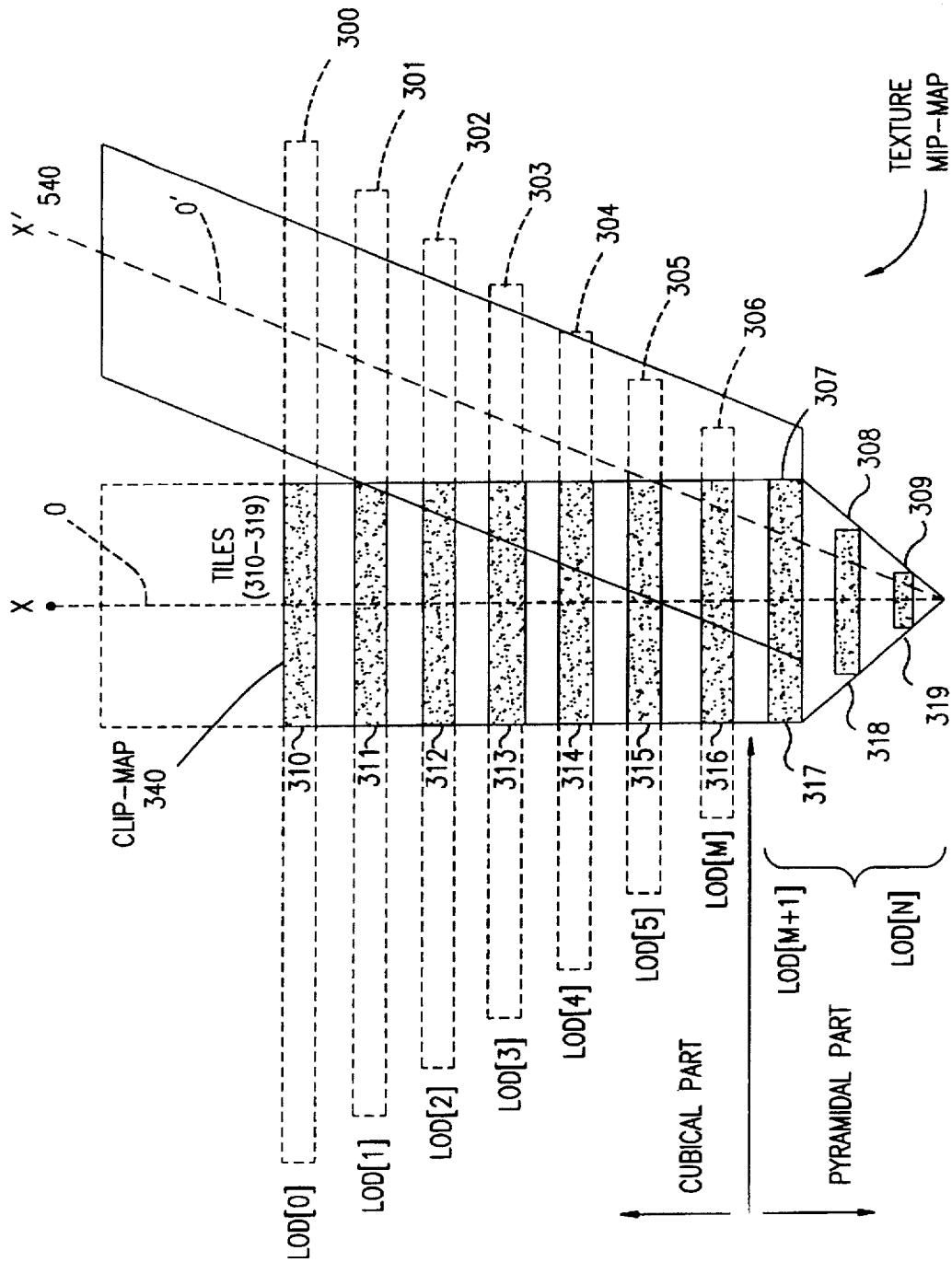

X'
540
0'
X
0
TILES
(310-319)
CLIP-MAP
340
TEXTURE
MIP-MAP
330
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
LOD[0]
LOD[1]
LOD[2]
LOD[3]
LOD[4]
LOD[5]
LOD[M]
LOD[M+1]
LOD[N]
CUBICAL PART
PYRAMIDAL PART

FIG.5

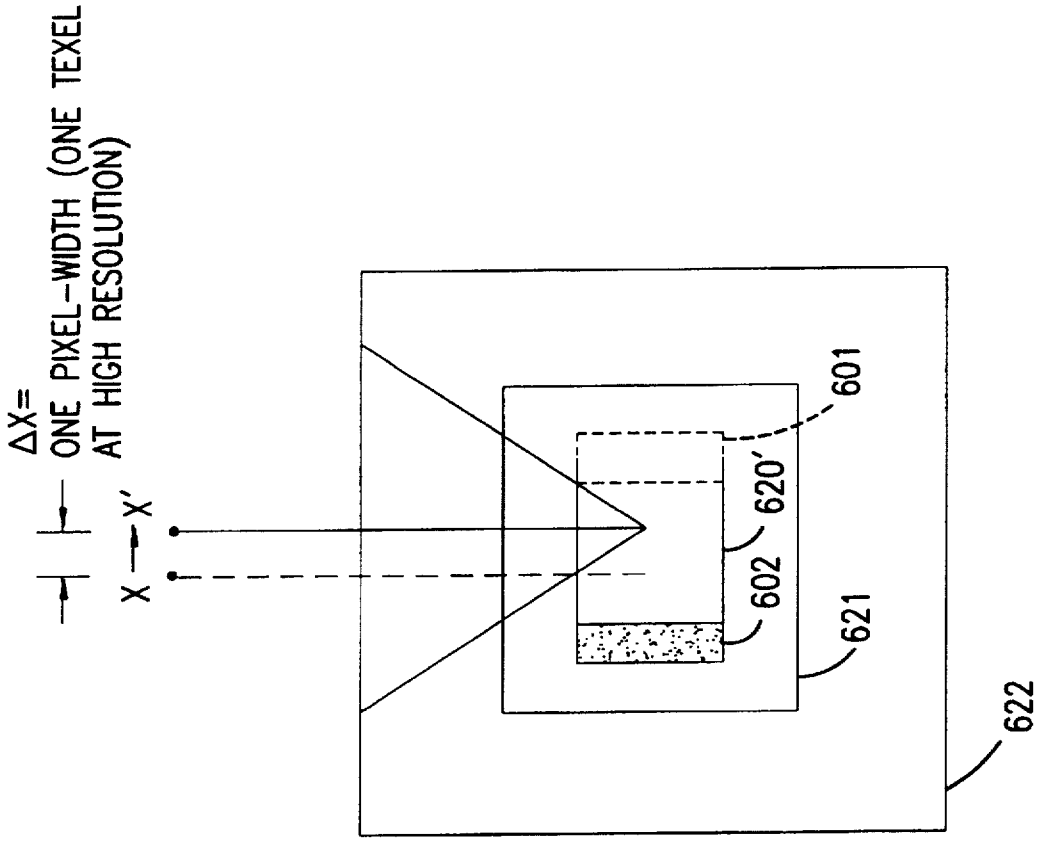

ΔX=
ONE PIXEL-WIDTH (ONE TEXEL AT HIGH RESOLUTION)
X
X'
601
620'
602
621
622

FIG.6B

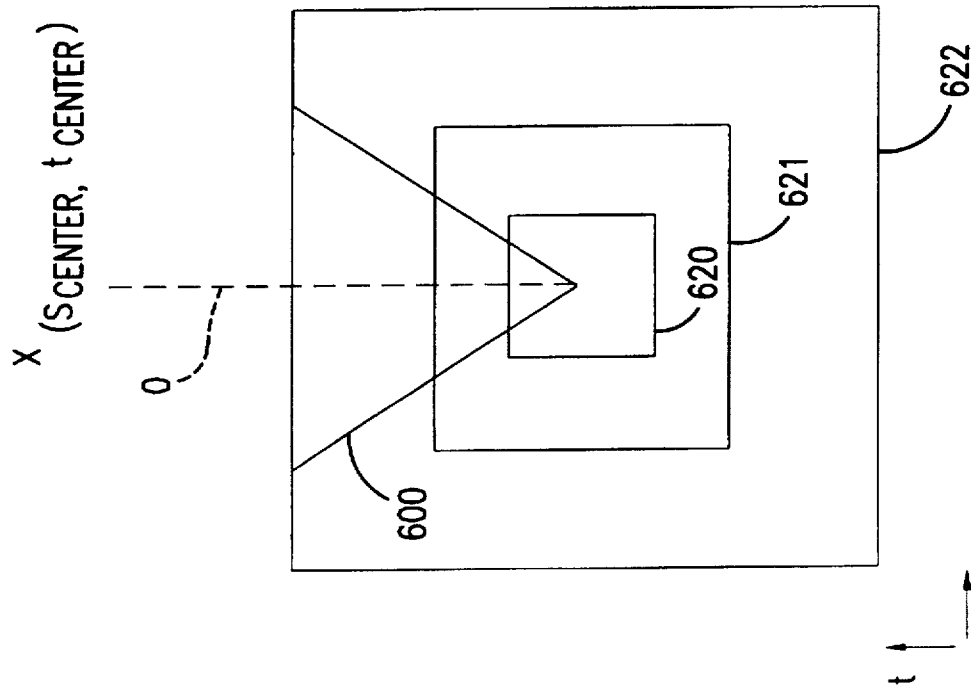

X (SCENTER, t CENTER)
0
600
620
621
622
s
t

FIG.6A

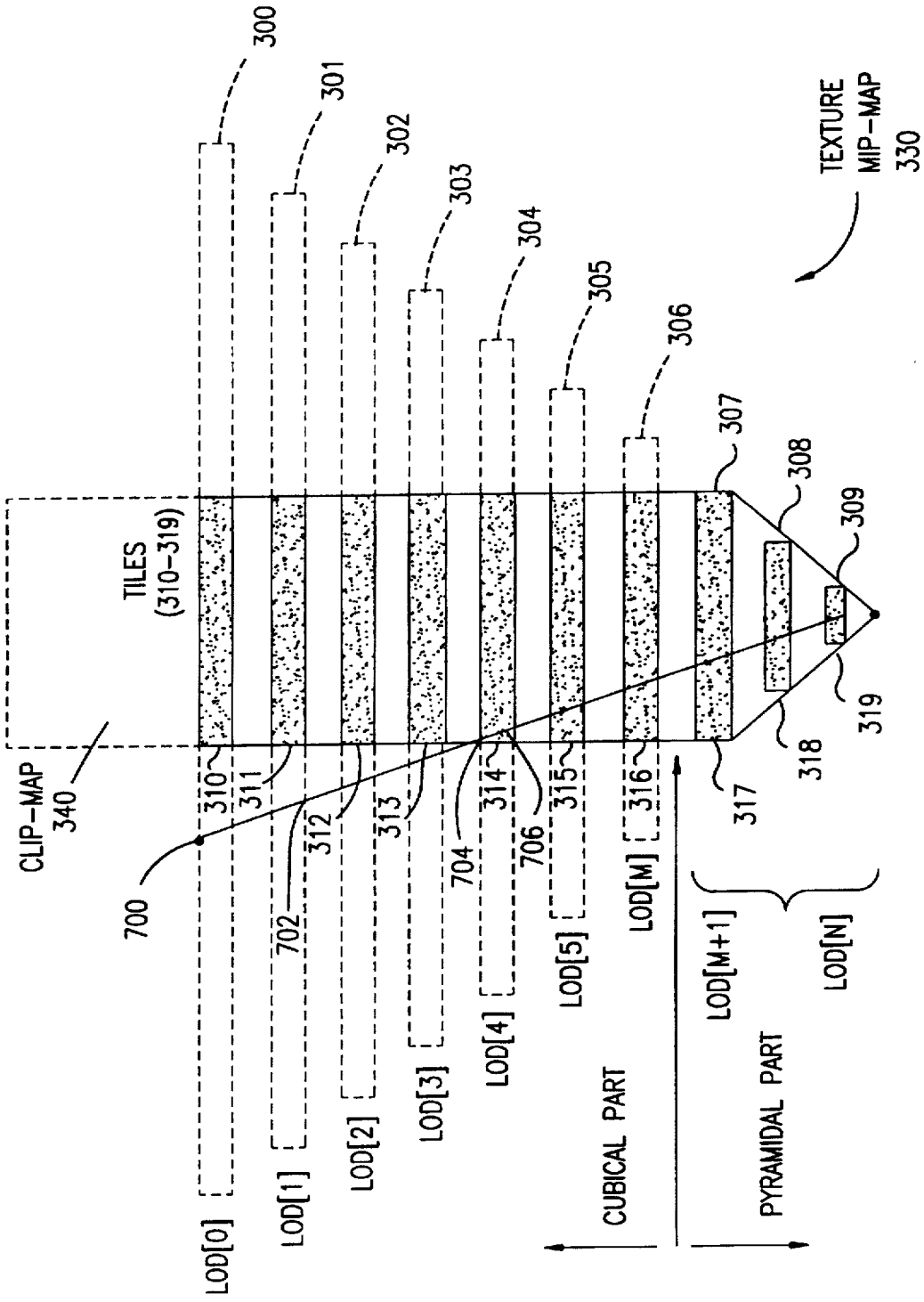

300
301
302
303
304
305
306
307
308
309
TEXTURE MIP-MAP 330
TILES (310-319)
CLIP-MAP 340
700
310
311
312
313
314
315
316
317
318
319
702
704
706
LOD[0]
LOD[1]
LOD[2]
LOD[3]
LOD[4]
LOD[5]
LOD[M]
LOD[M+1]
LOD[N]
CUBICAL PART
PYRAMIDAL PART

FIG. 7

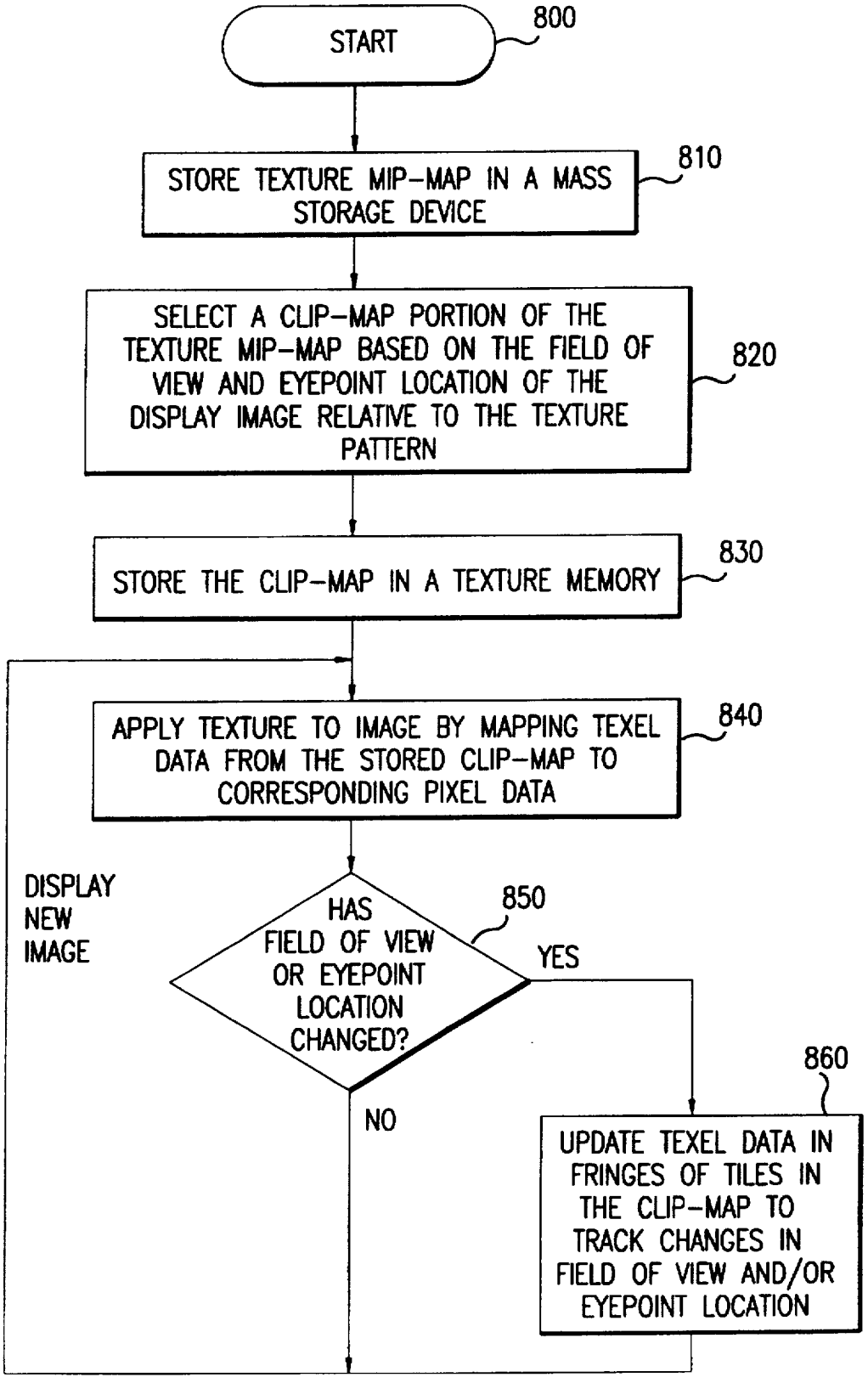

START
800
STORE TEXTURE MIP-MAP IN A MASS STORAGE DEVICE
810
SELECT A CLIP-MAP PORTION OF THE TEXTURE MIP-MAP BASED ON THE FIELD OF VIEW AND EYEPOINT LOCATION OF THE DISPLAY IMAGE RELATIVE TO THE TEXTURE PATTERN
820
STORE THE CLIP-MAP IN A TEXTURE MEMORY
830
APPLY TEXTURE TO IMAGE BY MAPPING TEXEL DATA FROM THE STORED CLIP-MAP TO CORRESPONDING PIXEL DATA
840
DISPLAY NEW IMAGE
HAS FIELD OF VIEW OR EYEPOINT LOCATION CHANGED?
850
YES
NO
860
UPDATE TEXEL DATA IN FRINGES OF TILES IN THE CLIP-MAP TO TRACK CHANGES IN FIELD OF VIEW AND/OR EYEPOINT LOCATION

FIG.8A

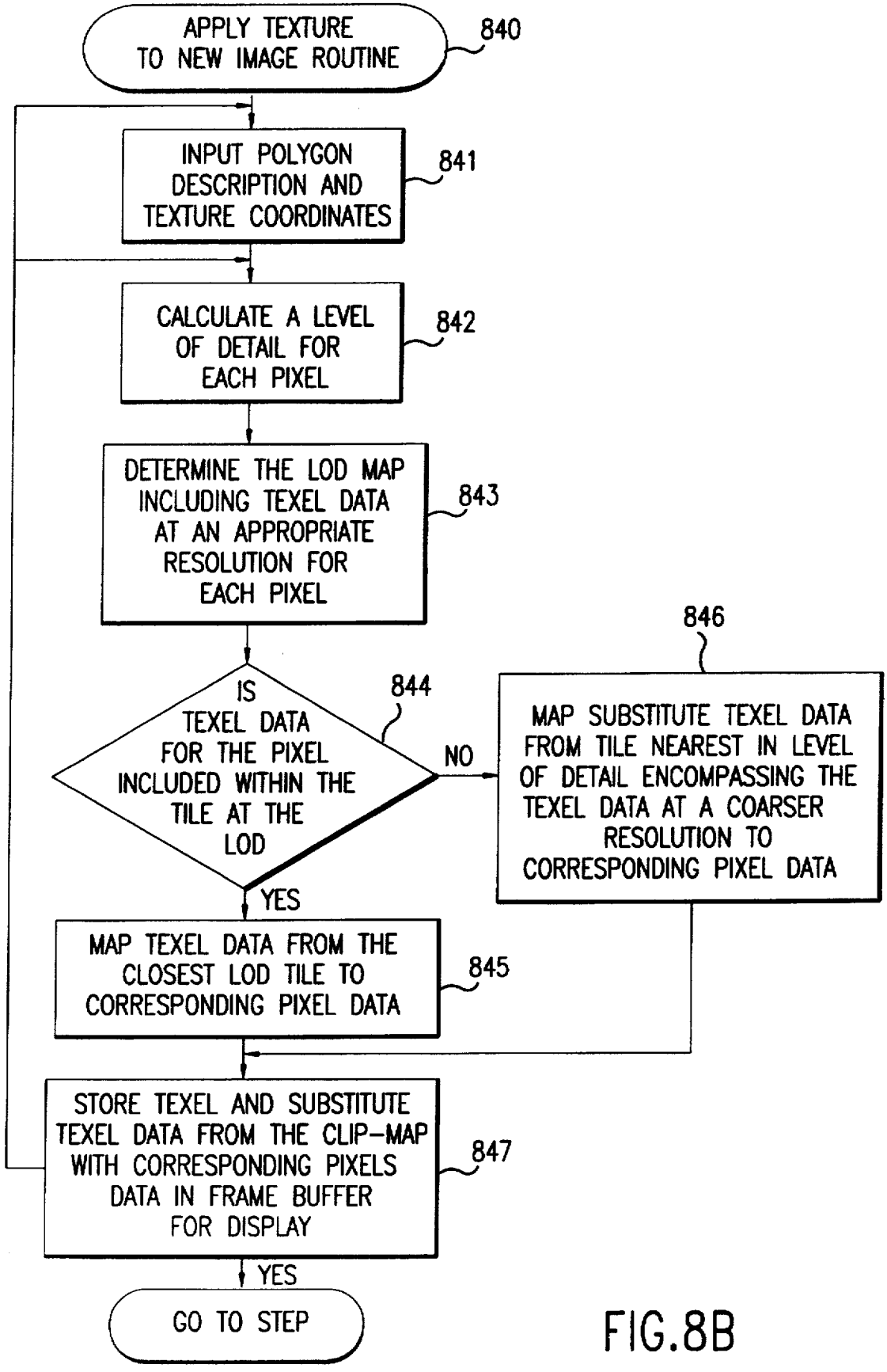

APPLY TEXTURE TO NEW IMAGE ROUTINE
840
INPUT POLYGON DESCRIPTION AND TEXTURE COORDINATES
841
CALCULATE A LEVEL OF DETAIL FOR EACH PIXEL
842
DETERMINE THE LOD MAP INCLUDING TEXEL DATA AT AN APPROPRIATE RESOLUTION FOR EACH PIXEL
843
IS TEXEL DATA FOR THE PIXEL INCLUDED WITHIN THE TILE AT THE LOD
844
NO
YES
MAP SUBSTITUTE TEXEL DATA FROM TILE NEAREST IN LEVEL OF DETAIL ENCOMPASSING THE TEXEL DATA AT A COARSER RESOLUTION TO CORRESPONDING PIXEL DATA
846
MAP TEXEL DATA FROM THE CLOSEST LOD TILE TO CORRESPONDING PIXEL DATA
845
STORE TEXEL AND SUBSTITUTE TEXEL DATA FROM THE CLIP-MAP WITH CORRESPONDING PIXELS DATA IN FRAME BUFFER FOR DISPLAY
847
YES
GO TO STEP

FIG.8B

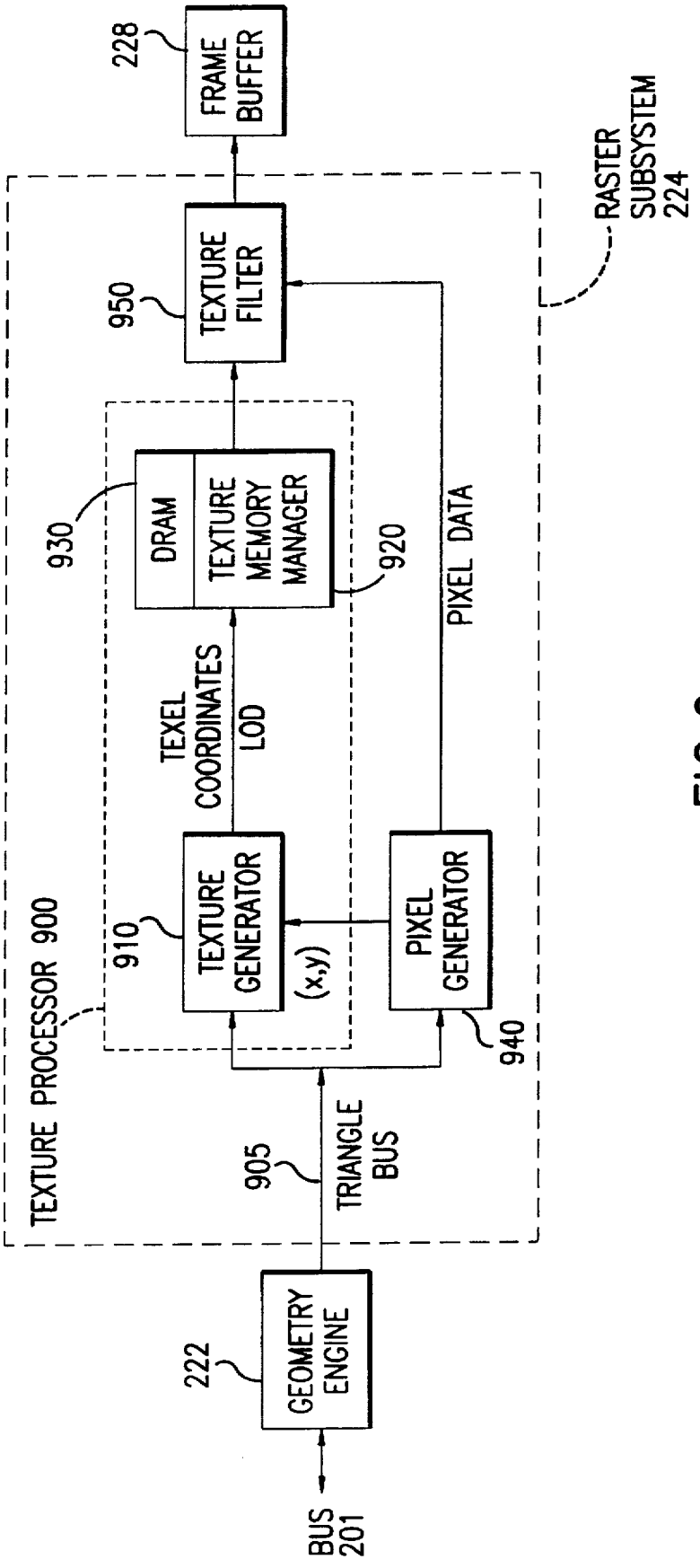

228
FRAME BUFFER
RASTER SUBSYSTEM 224
950
TEXTURE FILTER
930
DRAM
TEXTURE MEMORY MANAGER
920
PIXEL DATA
TEXEL COORDINATES
LOD
910
TEXTURE GENERATOR
(x,y)
PIXEL GENERATOR
940
TEXTURE PROCESSOR 900
905
TRIANGLE BUS
222
GEOMETRY ENGINE
BUS 201

FIG.9

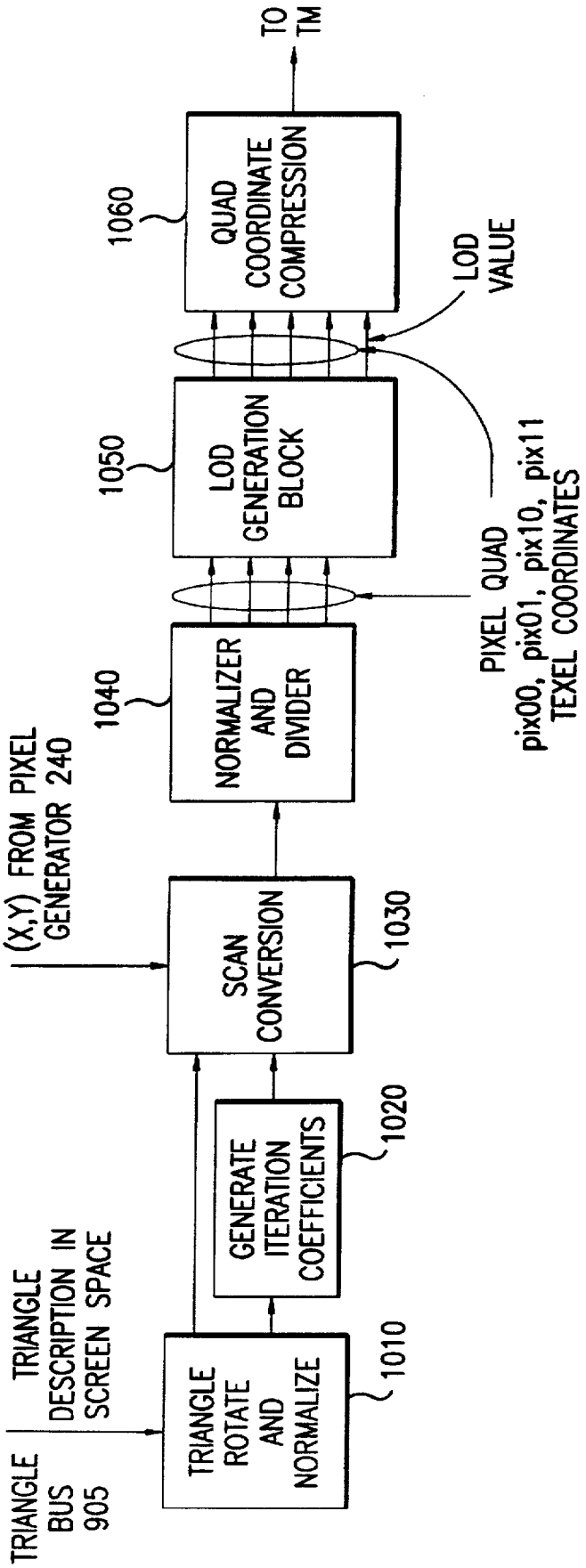

TRIANGLE BUS 905
TRIANGLE DESCRIPTION IN SCREEN SPACE
TRIANGLE ROTATE AND NORMALIZE
1010
GENERATE ITERATION COEFFICIENTS
1020
(X,Y) FROM PIXEL GENERATOR 240
SCAN CONVERSION
1030
NORMALIZER AND DIVIDER
1040
PIXEL QUAD
pix00, pix01, pix10, pix11
TEXEL COORDINATES
LOD GENERATION BLOCK
1050
LOD VALUE
QUAD COORDINATE COMPRESSION
1060
TO TM

FIG. 10

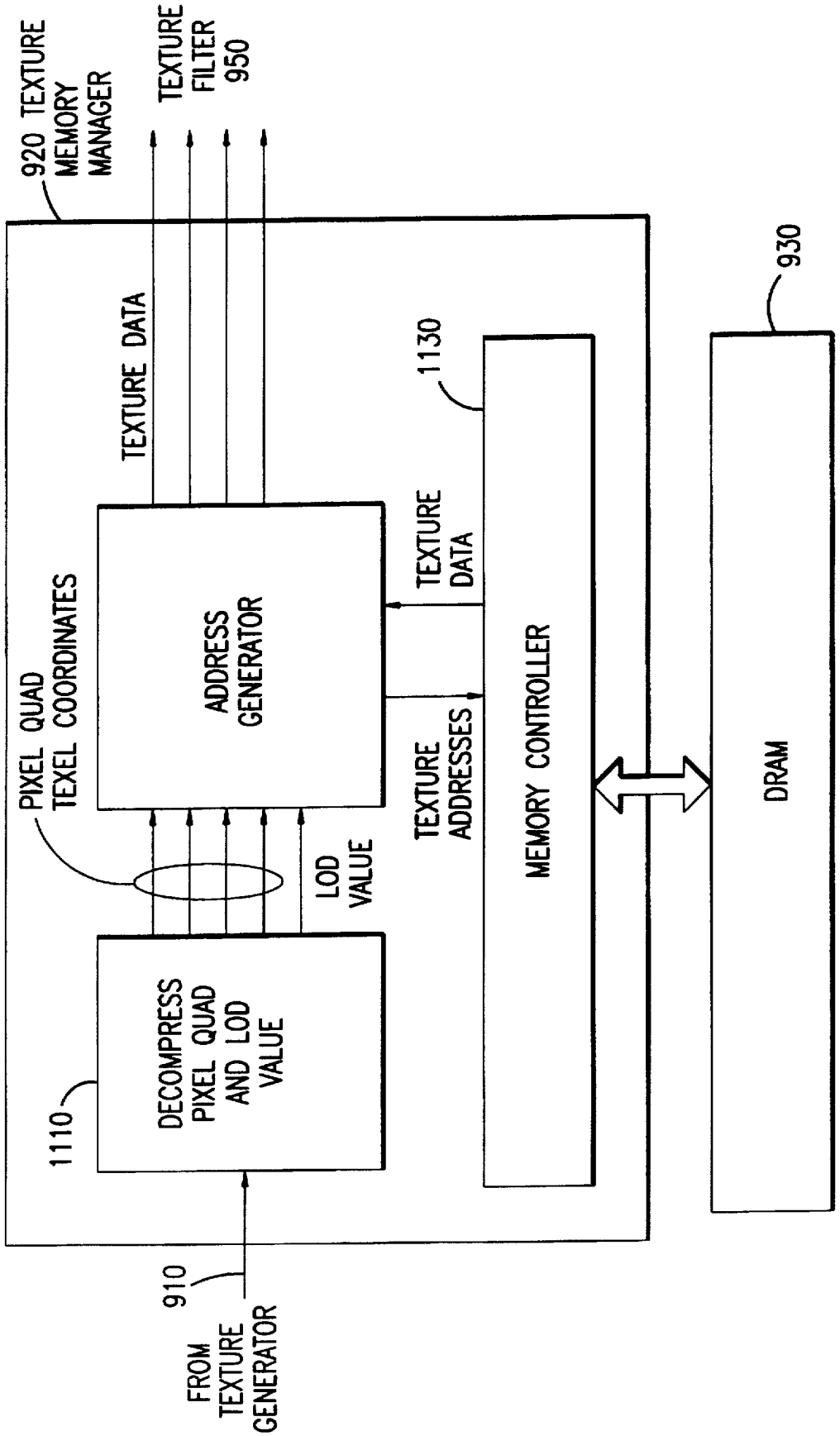

920 TEXTURE MEMORY MANAGER
TEXTURE FILTER 950
TEXTURE DATA
1130
930
PIXEL QUAD TEXEL COORDINATES
ADDRESS GENERATOR
TEXTURE DATA
TEXTURE ADDRESSES
MEMORY CONTROLLER
DRAM
LOD VALUE
1110
DECOMPRESS PIXEL QUAD AND LOD VALUE
910
FROM TEXTURE GENERATOR

FIG. 11

METHOD AND SYSTEM FOR PROVIDING TEXTURE USING A SELECTED PORTION OF A TEXTURE MAP

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention pertains to the field of computer graphics. More particularly, the present invention relates to an apparatus and method for providing texel data from selected portions of a texture MIP-map (referred to herein as a clip-map).

2. Related Art

Computer systems are commonly used for displaying graphical objects on a display screen. These graphical objects include points, lines, polygons, and three dimensional solid objects. By utilizing texture mapping techniques, color and other details can be applied to areas and surfaces of these objects. In texture mapping, a pattern image, also referred to as a "texture map," is combined with an area or surface of an object to produce a modified object with the added texture detail. For example, given the outline of a featureless cube and a texture map defining a wood grain pattern, texture mapping techniques can be used to "map" the wood grain pattern onto the cube. The resulting display is that of a cube that appears to be made of wood. In another example, vegetation and trees can be added by texture mapping to an otherwise barren terrain model. Likewise, labels can be applied onto packages or cans for visually conveying the appearance of an actual product. Textures mapped onto geometric surfaces provide motion and spatial cues that surface shading alone might not provide. For example, a sphere rotating about its center appears static until an irregular texture or pattern is affixed to its surface.

The resolution of a texture varies, depending on the viewpoint of the observer. The texture of a block of wood displayed up close has a different appearance than if that same block of wood were to be displayed far away. Consequently, there needs to be some method for varying the resolution of the texture (e.g., magnification and minification). One approach is to compute the variances of texture in real time, but this filtering is too slow for complex textures and/or requires expensive hardware to implement.

A more practical approach first creates and stores a MIP-map (multum in parvo meaning "many things in a small place"). The MIP-map consists of a texture pattern pre-filtered at progressively lower or coarser resolutions and stored in varying levels of detail (LOD) maps. See, e.g., the explanation of conventional texture MIP-mapping in Foley et al., *Computer Graphics Principles and Practice,* Second Edition, Addison-Wesley Publishing Company, Reading, Mass. (1990), pages 742–43 and 826–828 (incorporated by reference herein).

FIG. 1A shows a conventional set of texture LOD maps having pre-filtered texel data associated with a particular texture. Four different levels of detail (LOD[O]-LOD[3]m) are shown. Each successive coarser texture LOD has a resolution half that of the preceding LOD until a unitary LOD is reached representing an average of the entire high resolution base texture map LOD[0]. Thus, in FIG. 1A, LOD[O] is an 8×8 texel array; LOD[1] is a 4×4 texel array; LOD[2] is a 2×2 texel array; and LOD [3] is a single 1×1 texel array. Of course, in practice each LOD can contain many more texels, for instance, LOD[0] can be 8k×8k, LOD[1] 4k×4k, and so forth depending upon particular hardware or processing limits.

The benefit of MIP-mapping is that filtering is only performed once on texel data when the MIP-map is initially created and stored in LOD maps. Thereafter, texels having a dimension commensurate with pixel size are obtained by selecting the closest LOD map having an appropriate resolution. By obtaining texels from the pre-filtered LOD maps, filtering does not have to be performed during run-time. More sophisticated filtering operations can be executed beforehand during modeling without delaying real-time operation speed.

To render a display at the appropriate image resolution, a texture LOD is selected based on the relationship between the smallest texel dimension and the display pixel size. For a perspective view of a landscape **100**, as shown in FIG. **1B**, the displayed polygonal image is "magnified" in a foreground region relative to polygonal regions located closer to the center horizon and background along the direction indicated by the arrow. To provide texture for pixels in the closest foreground region, then, texels are mapped from the finest resolution map LOD[0]. Appropriate coarser LODs are used to map texel data covering pixels located further away from the viewer's eyepoint. Such multi-resolution texture MIP-mapping ensures that texels of the appropriate texture LOD gets selected during pixel sampling. To avoid discontinuities between images at varying resolutions, well-known techniques such as linear interpolation are used to blend the texel values of two LODs nearest a particular image pixel.

One significant drawback to conventional MIP-mapping, however, is the amount of memory consumed by the various texture LOD maps. Main memory in the form of a dynamic random access memory (DRAM) or a static random access memory (SRAM) is an expensive and inefficient site for a large texture MIP-map. Each additional level of detail map at a higher level of detail requires four times more memory. For example, a 16×16 texture array having 256 texture picture elements (texels), is four times bigger than an 8×8 texture array which has 64 texels. To put this increase in perspective, a texture MIP-map having six levels of detail requires over 4,096 times more memory than the texture map at the finest resolution. Implementing large texture MIP-maps quickly becomes an expensive luxury. In addition, for large texture MIP-maps, many portions of the stored MIP-map are not used in a display image.

Memory costs become especially prohibitive in photographic texture applications where the source texture, such as, satellite data or aerial photographs, occupy a large storage area. Creating a pre-filtered MIP-map representation of such source texture data further increases memory consumption.

This problem is further exacerbated by the fact that in order to increase the speed at which images are rendered for display, many of the high-performance computer systems contain multiple processors. A parallel, multiple processor architecture typically stores individual copies of the entire MIP-map in each processor memory.

Thus, there is a need to efficiently implement large texture maps for display purposes so as to minimize attendant memory and data retrieval costs. Visual quality must not be sacrificed for memory savings. Final images in an improved texture mapping system need to be virtually indistinguishable from that of images generated by a traditional MIP-map approach.

There is also a need to maintain real-time display speeds even when navigating through displays drawn from large texture maps. For example, flight simulations must still be

performed in real-time even when complex and voluminous source data such as satellite images of the earth or moon, are used to form large texture motifs.

SUMMARY OF THE INVENTION

The present invention pertains to an apparatus and method for providing texture by using selected portions of a texture MIP-map. The selected portions are referred to herein as a clip-map. Texel data relevant to a display image is stored, accessed, and updated efficiently in a clip-map in texture memory.

Entire texture MIP-maps are stored onto one or more mass storage devices, such as hard disk drives, optical disk drives, tape drives, CD drives, etc. According to the present invention, however, only a clip-map needs to be loaded into a more expensive but quicker texture memory (e.g., DRAM). Two dimensional or three dimensional texture data can be used. The clip-map is identified and selected from within a texture MIP-map based upon the display viewer's current eyepoint and field of view. The clip-map is composed of a set of selected tiles. Each tile corresponds to the respective portion of a texture level of detail map at or near the current field of view being rendered for display.

Virtually unlimited, large amounts of texture source data can be accommodated as texture MIP-maps in cheap, mass storage devices while the actual textured image displayed at any given time is readily drawn from selected tiles of corresponding clip-maps stored in one or more texture memories. In one example, the clip-map consists of only 6 million texels out of a total of 1.365 billion texels in a complete texture MIP-map—a savings of 1.36 billion texels! Where texture information is represented as a 8-bit color value, a texture memory savings of 10.9 gigabits (99.6%) is obtained.

According to another feature of the present invention, real-time flight over a large texture map is obtained through efficient updating of the selected clip-maps. When the eyepoint of a viewer shifts, the edges of appropriate clip-map tiles stored in the texture memory are updated along the direction of the eyepoint movement. New texel data for each clip-map tile is read from the mass storage device and loaded into the texture memory to keep the selected clip-map tiles in line with the shifting eyepoint and field of view. In one particularly efficient embodiment, when the eyepoint moves a distance equal to one texel for a particular LOD, one texel row of new texture LOD data is added to the respective clip-map tile to keep pace with the direction of the eyepoint movement. The texel row in the clip-map tile which encompasses texel data furthest from the moving eyepoint is discarded.

In a further feature of the present invention, a substitute texel value is used when an attempt is made to access a texel lying outside of a particular clip-map tile at the most appropriate resolution. The substitute texel value is obtained from the next coarser resolution clip-map tile which encompasses the texel being sought. The substitution texel that is chosen is the one closest to the location of the texel being accessed. Thus, this approach returns usable texel data from a clip-map even when mapping wayward pixels lying outside of a particular clip-map tile. Of course, for a given screen size, the tile size and tile center position can be calculated to guarantee that there would be no wayward pixels.

Finally, in one specific implementation of the present invention, texture processing is divided between a texture generator and a texture memory manager in a computer graphics raster subsystem. Equal-sized square tiles simplify texel addressing. The texture generator includes a LOD generation block for generating an LOD value identifying a clip-map tile for each pixel quad. A texture memory manager readily accesses the texel data from the clip-map using tile offset and update offset information.

Further embodiments, features, and advantages of the present inventions, as well as the structure and operation of the various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE FIGURES

The accompanying drawings, which are incorporated herein and form a part of the specification, illustrate the present invention and, together with the description, further serve to explain the principles of the invention and to enable a person skilled in the pertinent art to make and use the invention.

In the drawings:

FIG. **1A** shows a conventional multi-resolution MIP-map covering four levels of detail.

FIG. **1B** shows a conventional example of a polygon perspective of a landscape to which texture MIP-mapping can be applied.

FIG. **2** shows a block diagram of an example computer graphics system implementing the present invention.

FIG. **3** shows a side view of a ten-level texture MIP-map and the selected tiles that constitute a clip-map according to the present invention.

FIG. **4A** shows a side view of the first six levels of a clip-map for photographic terrain texture in one example of the present invention.

FIG. **4B** shows the progressively larger areas of a terrain texture covered by coarser tiles in the present invention.

FIG. **4C** shows three LOD-maps and associated clip-map tile areas relative to an observer's field of view.

FIG. **5** shows the shifting of selected tiles in a clip-map to track a change in the viewer eyepoint.

FIGS. **6A** and **6B** illustrate an efficient updating of clip-map tiles according to the present invention to follow eyepoint changes.

FIG. **7** shows obtaining a substitute texel value from the next closest clip-map tile having the highest resolution according to the present invention.

FIGS. **8A** and **8B** are flowcharts describing steps for obtaining a textured display image using a texture clip-map according to the present invention.

FIGS. **9** to **11** are block diagrams illustrating one example of a computer graphics subsystem implementing the present invention.

FIG. **9** shows a raster subsystem including a texture processor having a texture generator and a texture memory manager according to the present invention.

FIG. **10** shows a block diagram of the texture generator in FIG. **9**.

FIG. **11** shows a block diagram of the texture memory manager in FIG. **9**.

The present invention will now be described with reference to the accompanying drawings. In the drawings, like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit(s) of a reference number identifies the drawing in which the reference number first appears.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

I. Overview and Discussion

II. Terminology

III. Example Environment
IV. Computer Graphics System
V. Texture MIP-Mapping
VI. Selecting Portions of a Texture MIP-Map
VII. Photographic Terrain Texture
VIII. Updating the Clip-Map During Real-Time Operation
IX. Efficiently Updating the Clip-Map
X. Substitute Texel Data
XI. Overall Clip-Map Operation
XII. Specific Implementation
XIII. Square Clip-Map Tiles Example
XIV. Conclusion

I. Overview and Discussion

The present invention provides an apparatus and method for efficiently storing and quickly accessing texel data relevant for displaying a textured image. A large amount of texture source data is stored as a multi-resolution texture MIP-map on one or more mass storage devices. Only a relatively small clip-map representing selected portions of the complete texture MIP-map is loaded into faster, more expensive texture memory. These selected texture MIP-map portions include tiles which contain those texel values at each respective level of detail that are most likely to be mapped to pixels being rendered for display.

When the eyepoint or field of view is changed, the tiles stored in the texture memory are updated accordingly. In one efficient embodiment for updating the clip-map in real-time, new texel data is read from the mass storage device and loaded into the fringes of tiles to track shifts in the eyepoint. To maintain the size of the clip-map, tile texel data corresponding to locations furthest away from a new eyepoint is discarded. Anomalous attempts to access a texel lying outside of a particular clip-map tile are accommodated by utilizing a substitute texel value obtained from the next coarser resolution clip-map tile which encompasses the texel being sought.

II. Terminology

To more clearly delineate the present invention, an effort is made throughout the specification to adhere to the following term definitions as consistently as possible.

The term "texture map" refers to source data representing a particular texture motif at its highest resolution. A "texture MIP-map" and equivalents thereof such as a "MIP-map of texel data" are used to refer to conventional multum in parvo MIP-map representations of a texture map at successive multiple levels of details (LOD), that is, varying degrees of resolution.

On the other hand, "clip-map" is used to refer to portions of a MIP-map selected according to the present invention. Thus, a clip-map is a multi-resolution map made up of a series of tiles wherein at least some of the tiles represent smaller, selected portions of different levels of detail in a MIP-map. When two-dimensional texture data sets are used, these tiles are two-dimensional texel arrays. When three-dimensional texture data sets are used, these tiles are three-dimensional texel arrays, i.e. cubes.

Finally, texel array dimensions are given in convenient 1k×1k, 2k×2k, etc., shorthand notation. In actual implementations using digital processing, 1k equals 1,024, 2k equals 2,048, etc.

III. Example Environment

The present invention is described in terms of a computer graphics display environment for displaying images having textures drawn from multi-resolution texture MIP-maps. Moreover, sophisticated texture motifs covering a large area, such as satellite data and aerial photographs, are preferred to fully exploit the advantages of the clip-map system and method described herein. As would be apparent to a person skilled in the pertinent art, the present invention applies generally to different sizes and types of texture patterns limited only by the imagination and resources of the user.

Although the present invention is described herein with respect to two-dimensional texture mapping, the present invention can be extended to three-dimensional texture mapping when the requisite additional software and/or hardware resources are added. See e.g. the commonly-assigned, U.S. patent application Ser. No. 08/088,716, now U.S. Pat. No. 5,490,240 (Attorney Docket No. 15-4-99.00 (1452.0140000), filed Jul. 9, 1993, entitled "A System and Method of Generating Interactive Computer Graphic Images Incorporating Three Dimensional Textures," by James L. Foran et al. (incorporated herein by reference in its entirety).

IV. Computer Graphics System

Referring to FIG. **2**, a block diagram of a computer graphics display system **200** is shown. System **200** drives a graphics subsystem **220** for generating textured display images according to the present invention. In a preferred implementation, the graphics subsystem **220** is utilized as a high-end, interactive computer graphics workstation.

System **200** includes a host processor **202** coupled through a data bus **201** to a main memory **204**, read only memory (ROM) **206**, and mass storage device **208**. Mass storage device **208** is used to store vast amounts of digital data relatively cheaply. For example, the mass storage device **208** can consist of one or more hard disk drives, floppy disk drives, optical disk drives, tape drives, CD ROM drives, or any number of other types of storage devices having media for storing data digitally.

Different types of input and/or output (I/O) devices are also coupled to processor **202** for the benefit of an interactive user. An alphanumeric keyboard **210** and a cursor control device **212** (e.g., a mouse, trackball, joystick, etc.) are used to input commands and information. The output devices include a hard copy device **214** (e.g., a laser printer) for printing data or other information onto a tangible medium. A sound recording or video option **216** and a display screen **218** can be coupled to the system **200** to provide for multimedia capabilities.

Graphics data (i.e. a polygonal description of a display image or scene) is provided from processor **202** through data bus **201** to the graphics subsystem **220**. Alternatively, as would be apparent to one skilled in the art, at least some of the functionality of generating a polygonal description could be transferred to the computer graphics subsystem as desired.

Processor **202** also passes texture data from mass storage device **208** to texture memory **226** to generate and manage a clip-map as described below. Including the software and/or hardware in processor **202** for generating and managing clip-maps is one example for implementing the present invention. Separate modules or processor units for generating and managing a texture clip-map could be provided along data bus **201** or in graphics subsystem **220**, as would be apparent to one skilled in the art considering this description.

The graphics subsystem **220** includes a geometry engine **222**, a raster subsystem **224** coupled to a texture memory **226**, a frame buffer **228**, video board **230**, and display **232**.

Processor **202** provides the geometry engine **222** with a polygonal description (i.e. triangles) of a display image in object space. The geometry engine **222** essentially transforms the polygonal description of the image (and the objects displayed therein) from object space (also known as world or global space) into screen space.

Raster subsystem **224** maps texture data from texture memory **226** to pixel data in the screen space polygonal description received from the geometry engine **222**. Pixel data and texture data are eventually filtered, accumulated, and stored in frame buffer **228**. Depth comparison and other display processing techniques can be performed in either the raster subsystem **224** or the frame buffer **228**. Video unit **230** reads the combined texture and pixel data from the frame buffer **228** and outputs data for a textured image to screen display **232**. Of course, as would be apparent to one skilled in the art, the output image can be displayed on display **218**, in addition to or instead of display **232**. The digital data representing the output textured display image can also be saved, transmitted over a network, or sent to other applications.

The present invention is described in terms of this example high-end computer graphics system environment. Description in these terms is provided for convenience only. It is not intended that the invention be limited to application in this example environment. In fact, after reading the following description, it will become apparent to a person skilled in the relevant art how to implement the invention in alternative environments.

V. Texture MIP-Mapping

In the currently preferred embodiment of the present invention, large texture maps are stored on one or more mass storage devices **208**. A MIP-map representation of the texture maps can either be pre-loaded onto the mass storage device **208** or can be computed by the processor **202** and then stored onto mass storage device **208**. Two-dimensional or three-dimensional texture data sets are accommodated.

As is well-known in computer graphics design, the filtering and MIP-structure development necessary to derive and efficiently store the successive levels of detail for a texture MIP-map can be effectuated off-line prior to run-time operation. In this way, high-quality filtering algorithms can be utilized over a large texture map or database without hindering on-line image display speed and performance. Alternatively, if a less flexible but fast approach is acceptable, hardware can be used to produce the successive coarser levels of detail directly from input texture source data.

Under conventional texture mapping techniques, even if texture data were to be accessed from a remote, large texture MIP-map, the rendering of a textured image for display in real-time would be impractical, if not impossible. The present invention, however, realizes the advantages of accommodating large texture MIP-maps in one or more mass storage devices **208** without reducing texture access time. A relatively small clip-map representing only selected portions of a complete texture MIP-map is stored in a texture memory **226** having a fast rate of data return. In this way, texture memory **226** acts as a cache to provide texture rapidly to the raster subsystem **224**.

This hierarchical texture mapping storage scheme allows huge texture MIP-maps to be stored rather inexpensively on the mass storage device **208**. Based on the viewer eye point and/or field of view, only selected portions of a texture MIP-map corresponding to the texture motif to be rendered for display need to be loaded into the texture memory **226**. In this manner, large 2-D or 3-D texture MIP-maps can be used to provide texture rather inexpensively, yet the textured images can be rendered in real-time.

VI. Selecting Portions of a Texture MIP-Map

The process for determining which portions of a complete texture MIP-map are to be loaded from mass storage devices **208** into texture memory **226** to form a clip-map will now be described in more detail. FIG. **3** shows a side view of a complete texture MIP-map **330** having ten conventional levels of details (not shown to actual geometric scale). The levels of detail **300** to **309** each correspond to successively coarser resolutions of a texture map. The highest level of detail LOD[0] corresponds to the finest resolution texel map **300**. Each subsequent level of detail map **301** to **309** are filtered to have half the resolution of the preceding level of detail. Thus, each coarser level of detail covers an area of the texture map four times greater than the preceding level of detail.

FIG. **3** further illustrates the selected portions of the texture MIP-map **330** constituting a clip-map **340** according to the present invention. Clip-map **340** consists of relatively small tiles **310–319** which are regions of the levels of detail maps **300–309**. The actual size and shape of these tiles depends, inter alia, on the eye point and/or field of view of the display viewer. Each of these tiles must substantially encompass a potential field of view for a display view. To simplify addressing and other design considerations, equal-sized square tiles, i.e. square texel arrays, are used which can each be addressed relative to a common, fixed central eye point X and a center line O running through the clip map. For 3-D texture, square cubes consisting of a 3-D texel array are used.

Clip-map **340** essentially consists of a set of tiles, including a cubical part (**310–316**) and a pyramidal part (**317–319**). The cubical part consists of a shaft of tiles (**310–316**) of equal size. In the pyramidal part, the tiles consist of the actual level of detail maps (LOD[M+1]-LOD [N]). The pyramidal part begins at the first level of detail map (LOD[M+1]) which is equal to or smaller than a tile in the cubical part and extends down to a 1×1 texel (LOD[N]).

The reduced memory requirements for storing a clip-map instead of a complete texture MIP-map are clear. A complete, conventional 2-D texture MIP-map having dimensions given by "size in s" and "size in t" uses at least the following memory M:

$$M_{Texture\ MIP\text{-}map}=4/3 * (\text{size in s}) * (\text{size in t}) * \text{texel size (in bytes)}.$$

The smaller, clip-map example having equal-sized, square tiles in the cubical part only uses the following memory M:

$$M_{Texture\ clip\text{-}map}=[(\text{number of levels in cubical part} * (\text{tile size})^2)+ 4/3 * (\text{size in s of pyramidal part}) * (\text{size in t of pyramidal part})] * \text{texel size (in bytes)}.$$

Ten levels of detail are shown in FIG. **3** to illustrate the principle of the present invention. However, a smaller or greater number of levels of detail can be utilized. In a preferred example of the present invention, **16** levels of detail are supported in a high-end interactive computer graphics display workstation.

VII. Photographic Terrain Texture

Substantial reductions in memory costs and great improvements in real-time display capability are immedi-

ately realized by using a clip-map to render textured images. These advantages are quite pronounced when large texture maps such as a photographic terrain texture are implemented.

For example, source data from satellites covering 32 or more square kilometers of a planet or lunar surface is available. Such terrain can be adequately represented by a photographic texture MIP-map **430** having sixteen level of detail maps. The six highest resolution LOD maps **400–405** and tiles **410–415** are shown in FIG. **4**A. The highest resolution level LOD[0] consists of a 32k×32k array of texels. Successive level of details LOD[1]-LOD[5] correspond to the following texel array sizes: 16k×16k, 8k×8k, 4k×4k, 2k×2k, and 1k×1k. The remaining pyramidal part not shown consists of texel arrays 512×512, 256×256, . . . 1X1. Thus, a total of 1.365 billion texels must be stored in mass storage device **208**.

The size of the clip-map, however, is a function of the field of view and how close the observer is to the terrain. Generally, a narrow field of view requires a relatively small tile size increasing the memory savings. For example, the higher resolutions in the cubical part of clip-map **440** need only consist of 1k×1k tiles **410–414** for most close perspective images. The entire clip-map **440** then contains 6 million texels—a savings of 1.36 billion texels! Where texture information is represented as a 8-bit color value, a memory savings of 10.9 gigabits (99.6%) is obtained.

By storing the smaller clip-map **440** in texture memory **226**, further advantages inherent in a hierarchial memory system can be realized. The complete texture MIP-map **430** of 1.365 billion texels can be stored in cheap mass storage device **208** while the small clip-map **440** is held in a faster texture memory **226**, such as DRAM or SRAM. Sophisticated texel data can then be used to render rich textured images in real-time from the easily-accessed clip-map **440**. For example, a screen update rate of 30 to 60 Hz, i.e. 1/30 to 1/60 sec., is realized. The transport delay or latency of 1 to 3 frames of pixel data is approximately 10 to 50 msec. The above screen update rates and latency are illustrative of a real-time display. Faster or slower rates can be used depending on what is considered real-time in a particular application.

As shown in FIG. **4**B, the texel data stored in clip-map **440** can provide texture over a large display image area. For example, each high resolution texel of LOD[0] in a 32k×32k texel array can cover one square meter of geographic area in the display image. The 1k×1k tile **410** contains texel data capable of providing texture for a display image located within one square kilometer **420**.

Moreover, typical images are displayed at a perspective as described with respect to FIG. **1**B. The highest texel resolution included in tile **410** need only be used for the smaller areas magnified in a foreground region. Because of their progressively coarser resolution, each successive tile **411–414** covers the following broader areas 4 square kilometers (**421**), 16 square kilometers (**422**), 64 square kilometers (**423**), and 256 square kilometers (**424**), respectively. The complete 1,024 square kilometer area covered by texel data in tile **415** is not shown due to space limitations.

Even though the tiles may be equal-sized texel arrays, each tile covers a geometrically large area of a texture map because of filtering, albeit at a coarser level of detail. FIG. **4**C shows a perspective view of regions **420'** to **422'** covered by tiles **410** to **412** within the respective first three level of detail maps **400** to **402**. Each of the regions **420'** to **422'** are aligned along a center line O' stemming from an eyesight location X' to the center of the coarsest 11X1tile (not shown) in the pyramidal part of the clip-map **440**.

Thus, the clip-map **440** contains sufficient texel data to cover larger minified areas in the background of a display where coarser texture detail is appropriate. As a result, high quality textured display images, in perspective or warped, are still obtained for large texture patterns by using texel data from a clip-map.

VIII. Updating the Clip-Map During Real-Time Operation

The discussion thus far has considered only stationary eyepoints (X or X'). Many graphics display applications, such as flight applications over a textured terrain, present a constantly changing display view. As is well-known in graphics design, the display view can simulate flight by constantly moving the eyepoint along a terrain or landscape being viewed. Such flight can be performed automatically as part of a program application, or manually in response to user input such as mouse or joystick movement. Hyperlinks or jumps can be selected by a user to abruptly select a new viewpoint and/or field of view.

Regardless of the type of movement, today's user demands that new views be displayed in real-time. Delays in mapping texture data directly from large texture maps are intolerable. Reloading an entire new texture MIP-map for a new display viewpoint is often impractical.

As shown in FIG. **5**, when the eyepoint X shifts to a new point X' for a new display view, the texel data forming the clip-map **340** must similarly shift to track the new field of view along the axis O'. According to one feature of the present invention, portions of the texture MIP-map **330** forming a new "slanted" clip-map **540** are loaded into the texture memory **226**. The "slanted" clip-map **540** is necessarily drawn in a highly stylized and exaggerated fashion in FIG. **5** in the interest of clarity. Actual changes from one display view to the next are likely less dramatic. The actual tiles in the slanted clip-map would also be more staggered if the level of detail maps were drawn in true geometric proportion.

New tiles can be calculated and stored when the eyepoint and/or field of view changes to ensure that clip-map **540** contains the texel data which is most likely to be rendered for display. Likewise, the size and/or shape of the tiles can be altered to accommodate a new display view.

Texel data can be updated by loading an entire new slanted clip-map **540** from mass storage device **208** into texture memory **226**. Full texture loads are especially helpful for dramatic changes in eyepoint location and/or field of view.

IX. Efficiently Updating the Clip-Map

According to a further feature of the present invention, subtexture loads are performed to efficiently update texel data at the edges of tiles on an on-going basis. For example, as the eyepoint shifts, a new row of texel data is added to a tile in the direction of the eyepoint movement. A row located away from a new eyepoint location is discarded. Coarser tiles need not be updated until the eyepoint has moved sufficiently far to require a new row of texel data. Thus, the relatively small amount of texel data involved in a subtexture loads allows clip-map tiles to be updated in real-time while maintaining alignment with a moving eyepoint X.

For example, FIGS. **6**A and **6**B illustrate, respectively, the areas of texel data covered by clip-map tiles before and after

a subtexture load in the highest resolution tile **410**. Each of the areas **620** to **622** correspond to the regions of a texture map covered by 1k×1k tiles **410–412**, as described earlier with respect to the terrain of FIG. **4B**. A field of view **600** along the direction O marks the display area which must be covered by texture detail. Hence, only those texels residing within triangle **600** need to be stored or retained in the texture memory **226**. Texels around the fringes of triangle **600**, of course, can be added to provide additional texture data near the edges of a display image.

As shown in FIG. **6B**, each time the eyepoint advances one pixel-width (the pixel-width is exaggerated relative to the overall tile size to better illustrate the updating operation), a new texel row **601** located forward of the eyepoint is loaded from mass storage device **208** into the highest resolution tile **410** in texture memory **226**. The texel row **602** furthest from the new eyepoint X' is then discarded. In this way, tile **410** contains texel data for an area **620'** covering the new display area **600**. For small changes, then, coarser tiles (**411**, **412**, etc.) do not have to be updated. Because an equal amount of texels are discarded and loaded, the tile size (and amount of texture memory consumed by the clip-map) remains constant.

When the eyepoint moves a greater distance, texture data is updated similarly for the tiles at coarser LODs. Because two texels from an LOD are filtered to one texel in each direction s or t in texture space to form a successive LOD, the minimum resolution length for each LOD[n] is 2^n pixels, where n=0 to N. Accordingly, the tiles for LOD[1], LOD[2] . . . LOD[4] in the cubical part of a clip-map **440** are only updated when the eyepoint has moved two, four, eight, and sixteen pixels respectively. Because each level of detail in the pyramidal part is already fully included in the tile **415**, no updating is necessary in theory. To simplify an updating algorithm, however, when tiles in either the cubical part or the pyramidal part reach the end of a level of detail map, garbage or useless data can be considered to be loaded. Substitute texel data drawn from a coarser tile would be used instead of the garbage data to provide texture detail in those regions.

According to the present invention, then, the amount of texel data which must be loaded at any given time to update clip-map **540** is minimal. Real-time display operation is not sacrificed.

Texel data can be updated automatically and/or in response to a user-provided interrupt. Subtexture loads are further made in advance of when the texel data is actually rendered for display.

Finally, a check can be made to prevent attempts to draw an image using texel data which is being updated. Fringe regions are defined at the edges of tiles in the cubical part of the clip-map. The fringes include at least those texels being updated. To better accommodate digital addressing, it is preferred that the fringes consist of a multiple of eight texels. For example, in a 1k×1k tile having 1,024 texels on a side, eight texels at each edge form the fringe regions leaving 1,008 texels available to provide texture. Any attempt to access a texel in the fringe is halted and a substitute texel from the next coarsest level of detail is used instead. In this way, accesses by the raster subsystem **224** to specific texel data do not conflict with any texel updating operation.

X. Substitute Texel Data

According to a further feature of the present invention, substitute texel data is returned for situations where pixel data lying outside of a clip-map tile at a desired level of detail is to be mapped. When the raster subsystem **224** seeks to map a pixel not included in a clip-map tile corresponding to the desired level of detail, there is a problem in that the texture memory **226** cannot provide texel data at the most appropriate resolution at that particular pixel. The likelihood of such a situation arising can be minimized by brutishly mandating larger tile sizes. Of course, for a given screen size, the tile size and center position can be calculated to guarantee that there would be no wayward pixels.

The inventors, however, have discovered a more elegant solution which does not require an unnecessary expansion of the clip-map to accommodate wayward pixels. As shown in FIG. **7**, substitute texel data is derived for a pixel **700** lying outside of a clip-map **340**. A line **702** is first determined between the out-of-bounds pixel **700** and the apex of the pyramid part (center of the coarsest 1×1 texel tile LOD[N]). At some point **704**, this line **702** intersects the shaft of the clip-map. Substitute texel data **706**, covering pixel **700**, is then drawn from the nearest, coarser tile **314**.

In practice, when the resolution between levels of detail varies by a factor of 2, substitute texel data is easily drawn from the next coarser level of detail by shifting a texel address one bit. Operations for obtaining memory addresses for a texel located in a clip-map tile at the desired level of detail are further described below. Arithmetic and shifting operations to obtain a substitute texel memory address from the tile at the next nearest level of detail which covers the sought pixel is also described below.

By returning substitute texel having the next best level of detail, the overall texture detail remains rich as potential image degradation from pixels lying outside the clip-map is reduced. Moreover, by accommodating wayward pixels, greater latitude is provided in setting tile size, thereby, reducing the storage capacity required of texture memory **226**.

XI. Overall Clip-Map Operation

FIGS. **8A** and **8B** are flowcharts illustrating the operation of the present invention in providing texture data from a clip-map to display images.

First, a texture MIP-map representation of a texture map is stored in an economical memory such as, a mass storage device **208** (step **810**). Processor **202** can perform pre-filtering to calculate a texture MIP-map based on a texture map supplied by the user. Alternatively, the texture MIP-map can be loaded and stored directly into the mass storage device **208**.

Portions of the MIP-maps are then selected based on a particular field of view and/or eyepoint location to form a clip-map (step **820**). The clip-map is stored in a faster texture memory **226** (step **830**). Using texel data stored in the clip-map, texture can be mapped quickly and efficiently to corresponding pixel data to display a new textured image (step **840**).

To track changes in the field of view and/or eyepoint location of a display view, only the fringes of the tiles in the clip-map are updated (steps **850** and **860**). Such subtexture loads can be performed in real-time with minimal processor overhead. Unlike conventional systems, an entire texture load operation need not be performed.

FIG. **8B** shows the operation in step **840** for processing texture for a new image in greater detail. In step **841**, a description of polygonal primitives and texture coordinates is input. Triangle vertices are typically provided in screen space by a geometry engine **222**. A raster subsystem **224** then maps texture coordinates at the vertices to pixels.

Texture coordinates can be calculated for two-dimensional or three-dimensional texture LOD maps.

In step **842**, an appropriate level of detail is calculated for each pixel according to standard LOD calculation techniques based on the pixel dimension and texel dimension. A level of detail map closest to this appropriate level of detail is determined for the pixel (step **843**).

Texture closest to the appropriate level of detail is then obtained from the finest resolution tile in the clip-map which actually encompasses a texel corresponding to the pixel (steps **844** to **847**). First, a check is made to determine whether texel data for the pixel is included within a tile corresponding to the appropriate level of detail map determined in step **843**. Because the tiles are determined based on eyepoint location and/or field of view, texel data for a pixel is likely found within a tile at an appropriate level of detail. In this case, a texel is accessed from the corresponding tile and mapped to a corresponding pixel (step **845**).

As described earlier with respect to FIG. **7**, when a texel at the appropriate level of detail is not included within a corresponding tile, a coarser substitute texel is accessed. The substitute texel is chosen from the tile at the nearest level of detail which encompasses the originally-sought texel (step **846**). Texels mapped to pixels in step **845** and substitute texels mapped to corresponding pixels in step **846** are accumulated, filtered, and stored in a frame buffer **228** for subsequent display (step **847**).

Steps **841** to **847** are repeated for each input polygon description until a complete display image has been mapped to texel data and stored in the frame buffer **228**.

As would be apparent to one skilled in computer-generated textured graphics, the "clip-map" process described with respect to FIGS. **8**A and **8**B, can be carried out through firmware, hardware, software executed by a processor, or any combination thereof.

XII. Specific Implementation

FIGS. **9** to **11** illustrate one preferred example of implementing texture processing using a clip-map within computer graphics subsystem **220** according to the present invention. FIG. **9** shows a block diagram of a texture processor **900** within raster subsystem **224**. Texture processor **900** includes a texture generator **910** and a texture memory manager **920**. FIG. **10** shows a block diagram of the texture generator **910**. FIG. **11** shows a block diagram of a texture memory manager **920**. The operation of texture processor **900** in managing a clip-map to provide a texture display image will be made even more clear by the following description.

As shown in FIG. **9**, raster subsystem **224** includes texture processor **900**, pixel generator **940**, and texture filter **950**. The texture processor **900** includes a texture generator **910** coupled to a texture memory manager **920**. Texture memory manager **920** is further coupled to texture memory (DRAM) **930**.

Both the texture generator **910** and the pixel generator **940** are coupled to the geometry engine **222** via a triangle bus **905**. As explained earlier with respect to FIG. **2**, polygonal primitives (i.e. triangles) of an image in screen space (x,y), are output from the geometry engine **222**. Texture generator **910** outputs specific texel coordinates for a pixel quad and an appropriate LOD value based on the triangle description received from geometry engine **222** and the (x,y) screen space coordinates of the pixel quad received from pixel generator **940**. The LOD value identifies the clip-map tile in DRAM **930** which includes a texel at the desired level of detail. When a substitute texel must be used as described above, the LOD value identifies the clip-map tile at the closest coarser level of detail which covers the sought pixel data.

Texture memory manager **920** retrieves the texel or substitute texel from the clip-map stored in DRAM **930**. The retrieved texel data is then sent to a texture filter **950**.

Texture filter **950** filters texel data sent by the texture memory according to conventional techniques. For example, bi-linear and higher order interpolations, blending, smoothing, and texture sharpening techniques can be applied to textures to improve the overall quality of the displayed image. Texture filter **950** (or alternatively the frame buffer **228**) further combines and accumulates the texel data output from texture memory manager **930** and the corresponding pixel data output by the pixel generator **940** for storage in frame buffer **228**.

FIG. **10** shows component modules **1010–1060** forming texture generator **910**. Blocks **1010** to **1040** represent graphics processing modules for scan converting primitives. For purposes of this example, it is presumed that the primitive description consists of triangles and that pixel data is processed as 2×2 pixel quads. Module **1010** rotates and normalizes the input triangles received across triangle bus **905**. Module **1020** generates iteration coefficients. Scan conversion module **1030** then scan converts the triangles based on the outputs of modules **1010** and **1020** and the (x,y) coordinates output from a stepper (not shown) in pixel generator **940**. Texture coordinates for each pixel quad are ultimately output from scan conversion module **1030**. Normalizer and divider **1040** outputs normalized texture coordinates for the pixel quad. Such scan conversion processing is well-known for both two-dimensional and three-dimensional texture mapping and need not be described in further detail.

LOD generation block **1050** determines an LOD value for texture coordinates associated with a pixel quad. Compared to LOD generation blocks used in conventional texture MIP-mapping, LOD generation block **1050** is tailored to consider the contents of the clip-map and whether a substitute texel is used. The LOD value output from LOD generation block **1050** identifies the clip-map tile which includes texels or substitute texels covering a pixel quad.

LOD generation block **1050** essentially performs two calculations to derive the LOD value, as described previously with respect to steps **842** to **846**. LOD generation block **1050** first calculates an appropriate level of detail for the pixel quad (or pixel) according to standard LOD generation methods based on the individual pixel size and texel dimension. A level of detail map closest to the calculated level of detail is determined.

According to the present invention, then, a check is made to determine whether texel data for the pixel quad is included within a tile corresponding to the appropriate level of detail. When a texel is included within a tile at the appropriate level of detail, a LOD value corresponding to this tile is output. Otherwise, a LOD value is output identifying a tile at a lower level of detail which includes a substitute texel, as described earlier.

The above discussion largely refers to a pixel quad (i.e. a 2×2 pixel array). However, as would be apparent to one skilled in the art, the invention is not limited to use of a pixel quad. Using a pixel quad merely reduces the number of calculations and the amount of data which must be tracked. If desired, a separate LOD value could be calculated by LOD block **1050** for each pixel.

Other modules (not shown) can further use the pixel quad and LOD value output from LOD generation block **1050** to

perform supersampling, clamping, or other graphics display optimizing processes.

The present invention takes further advantage of the use of a pixel quad to reduce the amount of data required to be sent from the texture generator **910** to the texture memory manager **920**. Quad coordinate compression module **1060** compresses the texture coordinates of a pixel quad and the LOD value data sent to one or more texture memory managers **920**. In particular, in a 2×2 pixel quad, texture coordinates for one pixel are needed but the other three pixels can be defined relative to the first pixel. In this way, only the differences (i.e. the offsets) between the centers of the other three pixels relative to the center of the first pixel need to be transmitted.

FIG. **11** shows component modules forming texture memory manager **920**. Module **1110** decompresses texture coordinates of a pixel quad and LOD value information received from texture generator **910**. Texture coordinates for one pixel are received in full. Texture coordinates for the other three pixels can be determined by the offsets to the first pixel texture coordinates. The LOD value is associated with each pixel.

The texture coordinates sent from texture generator **910** are preferably referenced to the global texture MIP-map stored and addressed in mass storage device **208**. Address generator **1120** translates the texture coordinates for the pixel quad from the global texture MIP-map space of mass storage device **208** to texture coordinates specific to the clip-map stored and addressed in DRAM **930**. Alternatively, such translation could be carried out in the texture generator **910** depending upon how processing was desired to be distributed.

Address generator **1120** first identifies a specific tile at the level of detail indicated by the LOD value. To translate texture coordinates from global texture MIP-map space to the specific tile, the address generator **1120** considers both (1) the offset of the specific tile region relative to a complete level of detail map (i.e. tile offset) and (2) the center eyepoint location of a tile (i.e. update offset).

Memory addresses corresponding to the specific texture coordinates are then sent to a memory controller **1130**. Memory controller **1130** reads and returns texture data from DRAM **930**, through address generator **1120**, to texture filter **950**.

As would be apparent to one skilled in the art from the foregoing description, a conventional LOD value can be sent from the LOD generation block **1050** without regard to the selected portions of the texture MIP-map stored in the clip-map. The steps for determining whether a texel is within a tile and for determining a LOD value for a substitute texel would then be carried out at the texture memory manager **920**.

XIII. Square Clip-Map Tiles Example

Selecting, managing and accessing texel data in a clip-map will now be discussed with respect to the specific square clip-map **440**. According to another feature of the present invention, each tile in DRAM **930** is configured as a square centered about a common axis stemming from the eyepoint location. Each tile has a progressively coarser resolution varying by factor of 2. These restrictions simplify texel addressing for each tile in the cubical part of a clip-map considerably. The additional cost in hardware and/or software to address the tiles is minimal both when the tiles are initially selected and after any subtexture loads update the tiles.

By selecting square tiles to form the initial clip-map stored in DRAM **930**, the work of a processor **202** (or a dedicated processor unit in the graphics subsystem **220**) is straightforward. First, the center of the finest level (LOD[0]) is chosen. Preferably the center (s_{center},t_{center}) is defined as integers in global (s,t) texture coordinates. A finest resolution tile **410** is then made up from the surrounding texel data in LOD[0] map **400** within a predetermined distance d from the center point. All the other tiles for the levels of the cubical part (LOD[1]-LOD[M]) are established by shifting the center position down along the eyepoint. Thus, the texture coordinates ($s_{centern}$,$t_{centern}$) for a tile at level of detail LOD[n] are given by:

$$s_{centern}=s_{center>>n}$$

$$t_{centern}=t_{center>>n}.$$

where >>n denotes n shift operations. Texel data for the other tiles **401–404** is likewise swept in from regions a predetermined distance d in the s and t direction surrounding each center point.

Simple subtraction and comparison operations are carried out in texture generator **910** to determine whether a texel for a pixel quad is within a tile at an appropriate LOD. The finest, appropriate level of detail is determined by conventional techniques (see steps **842** and **843**). The additional step of checking whether the desired texel for a pixel quad is actually included within a tile at that LOD (step **844**) can be performed by calculating the maximum distance from four sample points in a pixel quad to the center of the tile. To be conservative, s and t distances for each of four sample points (s_0, t_0) . . . (s_3, t_3) can be calculated within a LOD generation block **1050** as follows:

$$s_0\,dist = |s_{centern} - s_0|$$

$$t_0\,dist = |t_{centern} - t_0|$$

. . .

$$s_3\,dist = |s_{centern} - s_3|$$

$$t_3\,dist = |t_{centern} - t_3|$$

where the tile center point at a LOD value n is given by ($s_{centern}$, $t_{centern}$)

Because the four samples of a pixel quad are strongly related, performing only two of the above subtractions is generally sufficient. Maximum distances s_{max} and t_{max} for a pixel quad are then determined by comparison. The use of square tiles means only one maximum distance in s or t needs to be calculated.

Based on the maximum distances in s and t, the finest available tile including texel or substitute texel data for the pixel quad is determined in a few arithmetic operations. If the constant size of the tiles is defined by s_{tile} and t_{tile} where the tile size equals ($2^{s\ tile}$, $2^{t\ tile}$) the finest available LOD value is given by the number of significant bits (sigbits) as follows:

$$\text{LOD } s \text{ finest}=\text{sigbits } (s_{max})-s_{tile};$$

$$\text{LOD } t \text{ finest}=\text{sigbits } (t_{max})-t_{tile}.$$

As would be apparent to one skilled in the art, LOD generation block **1050** can perform the above calculations and output the greater of the two numbers as the LOD value identifying the appropriate finest resolution tile containing texel or substitute data.

Finally, in addition to reducing the work of LOD generation block **1050**, the restrictive use of equal-sized square

tiles and power of two changes in resolution between tiles simplifies the work of the texture memory manager **920**. Address generator **1120** can translate global texture coordinates referencing LOD maps to specific tile texture coordinates in the cubical part of the clip-map by merely subtracting a tile offset and an update offset. The tile offset represents the offset from the corner of an LOD map to the corner of a tile. The update offset accounts for any updates in the tile regions which perform subtexture loads to track changes in the eyepoint location and/or field of view.

Thus, an address generator **1120** can obtain specific s and t tile coordinates (s_{fine}, t_{fine}) for a fine tile having a level of detail equal to the LOD value provided by the texture generator **910** as follows:

$$s_{fine}=s_{TG}-s \text{ tile offset}_{LOD\ value}-s \text{ update offset}_{LOD\ value}$$

$$t_{fine}=t_{TG}-t \text{ tile offset}_{LOD\ value}-t \text{ update offset}_{LOD\ value},$$

where s_{TG} and t_{TG} represent the global s and t texture coordinates provided by a texture generator **910**, s and t tile offset $_{LOD\ value}$ represent tile offset values in s and t for a tile at the LOD value provided by the texture generator **910**, and s and t update offset$_{LOD\ value}$ represent update offset values in s and t for a tile at the LOD value.

Some texture filters and subsequent processors also use texel data from the next coarser level of detail. In this case, texture memory manager **920** needs to provide texel data from the next coarser tile as well. Once the specific s_{fine} or t_{fine} coordinates are calculated as described above s and t coordinates (s_{coarse}, t_{coarse}) for the next coarser tile are easily calculated.

In particular, the global texture coordinates (s_{TG}, t_{TG}) are shifted one bit and reduced by 0.5 to account for the coarser resolution. The tile offset and update offset values (tile offset $_{LODcoarse}$ and update offset $_{LODcoarse}$) for the next coarser tile are also subtracted for each s and t coordinate. Thus, address generator **1120** determines specific texture coordinate in the next coarser tile as follows:

$$s_{coarse}=\text{trunc}[(s_{TG}>>1)-0.5)]-s \text{ tile offset}_{LODcoarse}-s \text{ update offset}_{LOD\ coarse}; \text{ and}$$

$$t_{coarse}=\text{trunc}[(t_{TG}>>1)-0.5)]-t \text{ tile offset}_{LOD\ coarse}-t \text{ update offset}_{LOD\ coarse}.$$

XIV. Conclusion

While specific embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. It will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined in the appended claims. Thus, the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

What is claimed is:

1. A computer graphics raster subsystem for providing texture from a texture pattern to an image to be rendered for display in real-time comprising:

texture memory for storing a select portion of a texture map representation of said texture pattern, said select texture map portion containing texture data at multiple levels of detail to substantially cover said image in a display view;

texture mapping means for mapping texture data from said select texture map portion stored in said texture memory to corresponding pixel data defining said display image; and

clip-map updating means for updating edges of said select texture map portion to track changes in the location of the eyepoint in real-time.

2. The system of claim **1**, wherein said texture memory stores said texture data in a two-dimensional or a three-dimensional texel array.

3. A computer graphics processing system for providing texture from a texture pattern in a display image, comprising:

first texture memory for storing a texture map, said texture map having multiple level of detail maps for storing texels representing the texture pattern filtered at successively coarser resolutions;

clip-map selecting means for selecting a clip-map within said texture map, said clip-map consisting of a set of tiles corresponding to regions of said level of detail maps substantially covering the display image;

second texture memory for storing said clip-map;

texture processor means for retrieving texels from said clip-map which map to pixel data forming the display image; and

clip-map updating means for updating texels at edges of said tiles stored in said second texture memory to track changes in the location of an eyepoint.

4. The computer graphics system of claim **3**, wherein said first and second texture memory constitute a hierarchial memory arrangement relative to said texture processor means, said texture processor means accesses texels stored in said second texture memory faster than said first texture memory.

5. The computer graphics system of claim **3**, wherein:

said first texture memory comprises a mass storage device, and

said second texture memory comprises at least one of a static random access memory (SRAM) device and a dynamic random access memory (DRAM) device.

6. The computer graphics system of claim **3**, wherein said clip-map selecting means determines the area of the texture pattern covered by each tile based on the location of an eyepoint of the display image.

7. The computer graphics system of claim **3**, wherein said clip-map updating means updates texels at edges of said tiles stored in said second texture memory to track changes in the location of a new eyepoint for a new display image.

8. The computer graphics system of claim **7**, wherein said clip-map updating means discards texels from said second texture memory and loads texels from said first texture memory into said second texture memory; said texels being discarded from at least one tile edge located furthest from said new eyepoint and said texels being loaded into at least one tile edge located closer to said new eyepoint.

9. The computer graphics subsystem of claim **3**, wherein, said texture processor means further retrieves coarser substitute texels from said clip-map.

10. The computer graphics system of claim **3**, wherein textured display images are output for display in real-time.

11. The computer graphics system of claim **3**, wherein said clip-map contains over 99% less texels than said texture map.

12. The computer graphics system of claim **3**, wherein said texture processor means comprises:

a texture generator; and

a texture memory manager coupled between said texture generator and said second texture memory;

wherein said texture generator includes a texture coordinate generator for generating texel coordinates for each pixel and a LOD generator for generating a LOD value for each pixel, said LOD value identifies the tile which covers the pixel and has texel dimensions closest in size to the pixel, said texel coordinates and LOD value being output for each pixel to said texture memory manager; and

wherein said texture memory manager retrieves texels from said clip-map for combining with said pixels to form a textured display image.

13. The system of claim **3**, wherein said second texture memory stores texels in said clip-map in a two-dimensional or a three-dimensional texel array.

14. A computer graphics processing system for providing texture from a texture pattern in a display image, comprising:

first texture memory for storing a texture map said texture map having multiple level of detail maps for storing texels representing the texture pattern filtered at successively coarser resolutions;

clip-map selecting means for selecting a clip-map within said texture map, said clip-map consisting of a set of tiles corresponding to regions of said level of detail maps substantially covering the display image;

second texture memory for storing said clip-map; and

texture processor means for retrieving texels from said clip-map which map to pixel data forming the display image, wherein said texture processor means comprises:

texture coordinate generator for generating at least one texture coordinate identifying where a pixel in the display image maps to the texture pattern,

LOD generator for generating a LOD value, and

a memory controller for retrieving at least one texel from said clip-map stored in said second texture memory based on said at least one texture coordinate and said LOD value, wherein, said LOD generator includes:

LOD identifying means for identifying an appropriate level of detail representing the level of detail map amongst said multiple level of detail maps where texel dimension is closest in size to said pixel,

texel determining means for determining whether a texel at said at least one texture coordinate is included in a first tile at said appropriate level of detail, said LOD value being set to said appropriate level of detail when said texel is included in said first tile, and

substitute texel determining means for determining a substitute texel in a second tile which includes said at least one texture coordinate, said LOD value being set to the level of detail of said second tile.

15. A computer graphics processing system for providing texture from a texture pattern in a display image, comprising:

first texture memory for storing a texture map, said texture map having multiple level of detail maps for storing texels representing the texture pattern filtered at successively coarser resolutions;

clip-map electing means for selecting a clip-map within said texture map, said clip-map consisting of a set of tiles corresponding to regions of said level of detail maps substantially covering the display image;

second texture memory for storing said clip-map; and

texture processor means for retrieving texels from said clip-map which map to pixel data forming the display image, wherein said set of tiles comprises a cubical part and a pyramidal part;

said one or more tiles in said cubical part comprising one or more arrays of texels, respectively, within said regions of said level of detail maps, and

said one or more tiles in said pyramidal part comprising one or more arrays of texels, respectively, said one or more arrays texels in said pyramidal part consisting of said level of detail maps which are equal to or smaller than said one or more tiles in said cubical part.

16. A method for providing texture from a texture pattern in a display image comprising the steps of:

storing a texture map in a first texture memory, said texture map having multiple level of detail maps for storing texels representing the texture pattern filtered at successively coarser resolutions;

selecting a clip-map within said texture map, said clip-map consisting of a set of tiles corresponding to regions of said level of detail maps, said set of tiles being centered in a field of view extending from an eyepoint location of the display image to substantially cover the display image;

storing said clip-map in a second texture memory;

retrieving texels from said clip-map stored in said second texture memory which map to pixels forming the display image; and

updating texels at edges of said tiles stored in said second texture memory to track changes in the location of the eyepoint location.

17. The method of claim **16**, wherein said updating step updates said clip-map to track a change in said eyepoint location for a new display image.

18. The method of **17**, further comprising the step of checking whether a texel to be accessed is located in a fringe portion of a tile, said fringe portion including the edges where texels are updated, and using a substitute texel when said texel is located in said fringe portion.

19. The method of claim **16**, wherein said updating step update texels at edges of said tiles stored in said second texture memory to track a change to a new eyepoint location for a new display image.

20. The method of claim **19**, wherein said updating step includes the steps of:

discarding texels from an edge of a tile located furthest from said new eyepoint location; and

loading texels from said first texture memory to said second texture memory, said texels being loaded next to an edge of a tile closer to the new eyepoint location wherein the number of texels loaded equals the number of texels discarded to maintain the size of said tile constant.

21. The method of claim **16**, further comprising the step of retrieving coarser substitute texels from said clip-map stored in said second texture memory.

22. The method of claim **16**, wherein said storing step stores texels in said clip-map in a two-dimensional or a three-dimensional texel array.

23. A method for providing texture from a texture pattern in a display image comprising the steps of:

storing a texture map in a first texture memory, said texture map having multiple level of detail maps for storing texels representing the texture pattern filtered at successively coarser resolutions;

selecting a clip-map within said texture map, said clip-map consisting of a set of tiles corresponding to regions of said level of detail maps, said set of tiles being centered in a field of view extending from an eyepoint location of the display image to substantially cover the display image;

storing said clip-map in a second texture memory; and

retrieving texels from said clip-map stored in said second texture memory which map to pixels forming the display image, wherein said retrieving step includes the following steps:

generating at least one texture coordinate identifying where a pixel in the display image maps to the texture pattern,

generating a LOD value, and

retrieving at least one texel from said clip-map stored in said second texture memory based on said at least one texture coordinate and said LOD value, wherein, said LOD value generating step includes:

identifying an appropriate level of detail representing the level of detail map amongst said multiple level of detail maps where texel dimension is closest in size to said pixel,

determining whether a texel at said at least one texture coordinate is included in a first tile at said appropriate level of detail and setting said LOD value to said appropriate level of detail when said texel is included in said first tile, and

when said texel in not included in said first tile, determining a substitute texel in a coarser second tile which includes said at least one texture coordinate and setting said LOD value to the level of detail of said second tile.

24. A texture processor for mapping texels from a clip-map to corresponding pixels, wherein the clip-map consists a set of tiles representing a selected portion of a texture MIP-map which substantially covers the pixels, said texture processor comprising:

a texture memory storing the clip-map;

a texture generator; and

a texture memory manager coupled between said texture generator and said texture memory, wherein

said texture generator includes a texture coordinate generator for generating texel coordinates for each pixel and a LOD generator for generating a LOD value for each pixel, said LOD value identifies the tile which covers the pixel and has texel dimensions closest in size to the pixel, said texel coordinates and LOD value being output for each pixel to said texture memory manager; and

said texture memory manager retrieves texels from the clip-map stored in said texture memory for combining with said pixels to form a textured display image.

25. The texture processor of claim **24**, wherein said texture memory stores said texels in two-dimensional or three-dimensional texel arrays.

26. The texture processor of claim **24**, wherein said texture memory manager includes an address generator for subtracting at least one of a tile offset and an update offset.

27. The texture processor of claim **24**, wherein 2×2 groups of pixels are processed as pixel quads.

28. The texture processor of claim **27**, wherein said texture generator includes a quad coordinate compression block for compressing texel coordinates for each pixel quad sent to said texture memory manager, and said texture memory manager includes a quad coordinate decompression block to decompress the compressed texel coordinates for each pixel quad.

* * * * *

Appendix CC - Claim Chart Showing Teachings of Fuller and Hornbacker Pertinent to Challenged Claims of U.S. Patent No. 8,924,506

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
1.Preamble: A method of retrieving large-scale images over network communications channels for display on a limited communication bandwidth computer device, said method comprising:	Fuller at Abstract; 15, 18-19, 21, 25, Fig. 1. See Ground 1 for relevant teachings of Hornbacker for this claim element.
1.A: issuing, from a limited communication bandwidth computer device to a remote computer, a request for an update data parcel	Fuller at Abstract, 15, 17, 18, 25, Figs. 1, 3. See Ground 1 for relevant teachings of Hornbacker for this claim element.
1.B: wherein the update data parcel is selected based on an operator controlled image viewpoint on the computer device relative to a predetermined image and	Fuller at 17-19, Figs. 3-5. Hornbacker at 5:16-25; 7:11-25; 13:11-14:16; Fig. 2.
1.C: the update data parcel contains data that is used to generate a display on the limited communication bandwidth computer device;	Fuller at 18. Hornbacker at Abstract; 8:7-15.
1.D: processing, on the remote computer, source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Fuller at 17, 25; Fig. 3. See Ground 1 for relevant teachings of Hornbacker for this claim element.
1.E: wherein series image K_0 being subdivided into a regular array	Fuller at 17; Fig. 3. Hornbacker at 6:13-19; 7:11-15; 8:30-

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
	9:28; 10:7-10.
1.F: wherein each resulting image parcel of the array has a predetermined pixel resolution	Fuller at 17, 21, Fig. 3. Hornbacker at 6:20-7:25, 8:30-9:28, 10:3-10, 11:19-28, 12:21-13:10, 13:26-14:6.
1.G: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Fuller at 21. Hornbacker, 6:13-7:25, 8:7-15, 10:11-23; 12:2-16.
1.H: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Fuller at Fig. 3. Hornbacker at 6:13-7:25, 8:7-15.
1.I: said array subdivision being related by a factor of two	Fuller at Fig. 3. Hornbacker at 6:13-7:25, 8:7-15.
1.J: such that each image parcel being of a fixed byte size;	See Ground 1 for relevant teachings of Hornbacker for this claim element.
1.K: receiving said update data parcel from the data parcel stored in the remote computer over a communications channel; and	Fuller at 15, 17, 19-21, Fig. 1. Hornbacker at Abstract, 3:10-27, 5:3-6:19.
1.L: displaying on the limited communication bandwidth computer device using the update data parcel that is a part of said predetermined image, an image wherein said update data parcel uniquely forms a discrete	Fuller at 17, 18, Figs. 3, 4. See Ground 1 for relevant teachings of Hornbacker for this claim element.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
portion of said predetermined image.	
2. The method of claim 1, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See discussions for claim element 1.D. Fuller at 17, 25. Hornbacker at 3:22-27; 5:16-6:19; 7:26-8:6; 10:3-11:28.
3. The method of claim 2, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	Fuller at 17.
4. The method of claim 1, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	Fuller at 25.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
5. The method of claim 1, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See discussions for claim 5 in Ground 1.
6. The method of claim 1, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	Fuller at Abstract, 17-19, Fig. 5.
7.A: The method of claim 1, wherein the processing further comprises compressing each data parcel and	See discussions for claim 5 in this Ground and discussions for claim element 7.A in Ground 1.
7.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Fuller at 17, 19; Figs. 3, 5. See discussions for Hornbacker for this claim element in Ground 1.
8.Preamble: A display system for displaying a large-scale image retrieved over a limited bandwidth	See discussions for claim element 1.Preamble.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
communications channel, said display system comprising:	
8.A: a display of defined screen resolution for displaying a defined image;	Fuller at 18, 21. See Ground 1 for relevant teachings of Hornbacker for this claim element.
8.B: a memory providing for the storage of a plurality of image parcels	Fuller at 18. See Ground 1 for relevant teachings of Hornbacker for this claim element.
8.C: displayable over respective portions of a mesh corresponding to said defined image;	See discussions for claim element 1.E.
8.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel;	Fuller at 16; Figs 1 and 2. See Ground 1 for relevant teachings of Hornbacker for this claim element.
8.E: a processor coupled between said display, memory and communications channel interface,	*See generally* Fuller and Hornbacker.
8.F: said processor operative to select said defined data parcel,	See discussions for claim element 1.B.
8.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	See discussions for claim element 8.B.
8.H: render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution	Fuller at 19, Figs. 3, 5.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
enhancement of said defined image on said display; and	Hornbacker at 12:24-13:10.
8.I: wherein a remote computer, coupled to the limited bandwidth communications channel, delivers the defined data parcel	See discussions for claim element 1.K.
8.J: wherein delivering the defined data parcel further comprises processing source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	See discussions for claim element 1.D.
8.K: wherein series image K_0 being subdivided into a regular array	See discussions for claim element 1.E.
8.L: wherein each resulting image parcel of the array has a predetermined pixel resolution	See discussions for claim element 1.F.
8.M: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See discussions for claim element 1.G.
8.N: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	See discussions for claim element 1.H.
8.O: said array subdivision being related by a factor of two	See discussions for claim element 1.I.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
8.P: such that each image parcel being of a fixed byte size.	See discussions for claim element 1.J.
9. The display system of claim 8, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See discussions for claim 2.
10. The display system of claim 9, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	See discussions for claim 3.
11. The display system of claim 8, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	See discussions for claim 4.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
12. The display system of claim 8, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See discussions for claim 5.
13. The display system of claim 8, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	See discussions for claim 6.
14.A: The display system of claim 8, wherein the processing may further comprises compressing each data parcel and	See discussions for claim element 7.A.
14.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	See discussions for claim element 7.B.
15.Preamble: A remote computer for delivering large-scale images over network communications channels for display on a limited communication	See discussions for claim elements 1.Preamble, 8.Preamble, and 8.I.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
bandwidth computer device that has a display system for displaying a large-scale image retrieved over a limited bandwidth communications channel,	
15.A: a display of defined screen resolution for displaying a defined image,	See discussions for claim element 8.A.
15.B: a memory providing for the storage of a plurality of image parcels	See discussions for claim element 8.B.
15.C: displayable over respective portions of a mesh corresponding to said defined image,	See discussions for claim element 8.C.
15.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel and	See discussions for claim element 8.D.
15.E: a processor coupled between said display, memory and communications channel interface,	See discussions for claim element 8.E.
15.F: said processor operative to select said defined data parcel,	See discussions for claim element 8.F.
15.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	See discussions for claim element 8.G.
15.H: render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display, the remote computer	See discussions for claim element 8.H.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
comprises:	
15.I: a parcel processing unit that processes a piece of source image data and	See discussions for claim element 8.J.
15.J: delivers the defined data parcel to the limited communication bandwidth computer device; and	See discussions for claim element 8.I.
15.K: wherein the parcel processing unit further comprises a parcel processing control that processes source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	See discussions for claim element 8.J.
15.L: wherein series image K_0 being subdivided into a regular array	See discussions for claim element 8.K.
15.M: wherein each resulting image parcel of the array has a predetermined pixel resolution	See discussions for claim element 8.L.
15.N: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See discussions for claim element 8.M.
15.O: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	See discussions for claim element 8.N.
15.P: said array subdivision being related by a factor of two	See discussions for claim element 8.O.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
15.Q: such that each image parcel being of a fixed byte size.	See discussions for claim element 8.P.
16. The remote computer of claim 15, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See discussions for claim 2.
17. The remote computer of claim 16, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	See discussions for claim 3.
18. The remote computer of claim 15, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	See discussions for claim 4.

8,924,506 Patent Claim Language	**Teachings of *The MAGIC Project: From Vision to Reality* ("Fuller") in view of WO 99/41675 ("Hornbacker")**
19. The remote computer of claim 15, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See discussions for claim 5.
20. The remote computer of claim 15, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	See discussions for claim 6.
21.A The remote computer of claim 15, wherein processing further comprises compressing each data parcel and	See discussions for claim element 7.A.
21.B storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	See discussions for claim element 7.B.

Appendix DD - Claim Chart Showing Teachings of Yap and Rabinovich Pertinent to Challenged Claims of U.S. Patent No. 8,924,506

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
1.Preamble: A method of retrieving large-scale images over network communications channels for display on a limited communication bandwidth computer device, said method comprising:	Yap at Abstract; 1:8-11. Rabinovich at 1.
1.A: issuing, from a limited communication bandwidth computer device to a remote computer, a request for an update data parcel	Yap at 9:1-11. Rabinovich at p.2, System Overview.
1.B: wherein the update data parcel is selected based on an operator controlled image viewpoint on the computer device relative to a predetermined image and	Yap at 8:55-9:5. Rabinovich at Fig. 3.
1.C: the update data parcel contains data that is used to generate a display on the limited communication bandwidth computer device;	Yap at 10:45-48. Rabinovich at pp.3-4.
1.D: processing, on the remote computer, source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	Yap at 4:21-24; 7:6-8:7. Rabinovich at p. 2, System Overview; p. 3, Texture Processing; Fig. 4.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
1.E: wherein series image K_0 being subdivided into a regular array	Yap at 7:10-15. Rabinovich at p. 2, § 2, System Overview; Fig. 4.
1.F: wherein each resulting image parcel of the array has a predetermined pixel resolution	Yap at 4:21-24; 7:6-8:7. Rabinovich at p. 2, System Overview; p. 3, Texture Processing; Fig. 4.
1.G: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	Yap at 7:6-10; 8:4-7. Rabinovich at p. 4, Experimental Results; Fig. 5.
1.H: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	Yap at 7:36-40; Fig. 2A. Rabinovich at Fig. 4.
1.I: said array subdivision being related by a factor of two	Yap at 7:36-40; Fig. 2A. Rabinovich at Fig. 4.
1.J: such that each image parcel being of a fixed byte size;	Yap at 8:4-7. Rabinovich at 2, § 2, System Overview.
1.K: receiving said update data parcel from the data parcel stored in the remote computer over a communications channel; and	Yap at 10:25-44; *see also* Yap at Abstract. Rabinovich at p.3, Caching; p.4, Experimental Results.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
1.L: displaying on the limited communication bandwidth computer device using the update data parcel that is a part of said predetermined image, an image wherein said update data parcel uniquely forms a discrete portion of said predetermined image.	Yap at 10:45-48. Rabinovich at pp. 3-4, Texture Processing.
2. The method of claim 1, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See discussions for claim element 1.D. Yap at 3:31-34; 3:41-52; 4:5-12; 4:25-33.
3. The method of claim 2, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	Yap at 3:31-38; 4:25-33. Rabinovich at Abstract; p. 2, § 2, System Overview; p. 3, § 3.4, Caching.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
4. The method of claim 1, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	See discussions in claim element 1.Preamble.
5. The method of claim 1, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	Yap at 8:3-4. Rabinovich at 2, § 2, System Overview.
6. The method of claim 1, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	Yap at 9:1-11; 9:12-10:9; Fig. 5. Rabinovich at p.2, Data Reduction; p.2, System Overview; p.3, Continuous Resolution; Fig. 3.
7.A: The method of claim 1, wherein the processing further	See discussions for claim 5.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
comprises compressing each data parcel and	
7.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	Rabinovich at Fig. 4.
8.Preamble: A display system for displaying a large-scale image retrieved over a limited bandwidth communications channel, said display system comprising:	See discussions for claim element 1.Preamble.
8.A: a display of defined screen resolution for displaying a defined image;	Yap at 5:58-67. Rabinovich at 4, Experimental Results.
8.B: a memory providing for the storage of a plurality of image parcels	Yap at 10:25-44; *see also* Yap at Abstract. Rabinovich at p.3, Caching; p.4, Experimental Results.
8.C: displayable over respective portions of a mesh corresponding to	See discussions for claim element 1.E.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
said defined image;	
8.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel;	Yap at Fig. 1.
8.E: a processor coupled between said display, memory and communications channel interface,	Yap at 6:5-10. Rabinovich at p. 1, § 1, Introduction.
8.F: said processor operative to select said defined data parcel,	See discussions for claim element 1.B.
8.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	See discussions for claim element 8.B.
8.H: render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display; and	Yap at 3:31-34; 10:45-48; 10:65-67.
8.I: wherein a remote computer, coupled to the limited bandwidth communications channel, delivers the defined data parcel	See discussions for claim element 1.K.
8.J: wherein delivering the defined data parcel further comprises processing source image data to	See discussions for claim element 1.D.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
obtain a series K_{1-N} of derivative images of progressively lower image resolution and	
8.K: wherein series image K_0 being subdivided into a regular array	See discussions for claim element 1.E.
8.L: wherein each resulting image parcel of the array has a predetermined pixel resolution	See discussions for claim element 1.F.
8.M: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See discussions for claim element 1.G.
8.N: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	See discussions for claim element 1.H.
8.O: said array subdivision being related by a factor of two	See discussions for claim element 1.I.
8.P: such that each image parcel being of a fixed byte size.	See discussions for claim element 1.J.
9. The display system of claim 8, wherein processing the source image data further comprises one of	See discussions for claim 2.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	
10. The display system of claim 9, wherein receiving the update data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	See discussions for claim 3.
11. The display system of claim 8, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	See discussions for claim 4.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
12. The display system of claim 8, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See discussions for claim 5.
13. The display system of claim 8, wherein processing the source image data further comprises queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	See discussions for claim 6.
14.A: The display system of claim 8, wherein the processing may further comprises compressing each data parcel and	See discussions for claim element 7.A.
14.B: storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	See discussions for claim element 7.B.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
15.Preamble: A remote computer for delivering large-scale images over network communications channels for display on a limited communication bandwidth computer device that has a display system for displaying a large-scale image retrieved over a limited bandwidth communications channel,	See discussions for claim elements 1.Preamble, 8.Preamble, and 8.I.
15.A: a display of defined screen resolution for displaying a defined image,	See discussions for claim element 8.A.
15.B: a memory providing for the storage of a plurality of image parcels	See discussions for claim element 8.B.
15.C: displayable over respective portions of a mesh corresponding to said defined image,	See discussions for claim element 8.C.
15.D: a communications channel interface supporting the retrieval of a defined data parcel over a limited bandwidth communications channel and	See discussions for claim element 8.D.
15.E: a processor coupled between said display, memory and communications channel interface,	See discussions for claim element 8.E.
15.F: said processor operative to	See discussions for claim element 8.F.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
select said defined data parcel,	
15.G: retrieve said defined data parcel via said limited bandwidth communications channel interface for storage in said memory, and	See discussions for claim element 8.G.
15.H: render said defined data parcel over a discrete portion of said mesh to provide for a progressive resolution enhancement of said defined image on said display, the remote computer comprises:	See discussions for claim element 8.H.
15.I: a parcel processing unit that processes a piece of source image data and	See discussions for claim element 8.J.
15.J: delivers the defined data parcel to the limited communication bandwidth computer device; and	See discussions for claim element 8.I.
15.K: wherein the parcel processing unit further comprises a parcel processing control that processes source image data to obtain a series K_{1-N} of derivative images of progressively lower image resolution and	See discussions for claim element 8.J.
15.L: wherein series image K_0 being subdivided into a regular array	See discussions for claim element 8.K.
15.M: wherein each resulting image parcel of the array has a	See discussions for claim element 8.L.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
predetermined pixel resolution	
15.N: wherein image data has a color or bit per pixel depth representing a data parcel size of a predetermined number of bytes,	See discussions for claim element 8.M.
15.O: resolution of the series K_{1-N} of derivative images being related to that of the source image data or predecessor image in the series by a factor of two, and	See discussions for claim element 8.N.
15.P: said array subdivision being related by a factor of two	See discussions for claim element 8.O.
15.Q: such that each image parcel being of a fixed byte size.	See discussions for claim element 8.P.
16. The remote computer of claim 15, wherein processing the source image data further comprises one of pre-processing the source image data on the remote computer and processing the source image data in real-time on-demand based on the request for the updated image parcel.	See discussions for claim 2.
17. The remote computer of claim 16, wherein receiving the update	See discussions for claim 3.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
data parcel over a communications channel further comprises streaming the update data parcel over a communications channel to the limited communication bandwidth computer device.	
18. The remote computer of claim 15, wherein the limited communication bandwidth computer device further comprises one of a mobile computer system, a cellular computer system, an embedded computer system, a handheld computer system, a personal digital assistants and an internet-capable digital phone and a television.	See discussions for claim 4.
19. The remote computer of claim 15, wherein a size of the data parcel on the remote computer is different from the update data parcel on the limited communication bandwidth computer device.	See discussions for claim 5.
20. The remote computer of claim 15, wherein processing the source image data further comprises	See discussions for claim 6.

8,924,506 Patent Claim Language	**Teachings of U.S. 6,182,114 ("Yap") in view of Boris Rabinovich et al., Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization '97 ("Rabinovich")**
queuing the update data parcels on the remote computer based on an importance of the update data parcel as determined by the remote computer.	
21.A The remote computer of claim 15, wherein processing further comprises compressing each data parcel and	See discussions for claim element 7.A.
21.B storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value that represents the data set resolution index D and corresponding image array coordinate.	See discussions for claim element 7.B.

APPENDIX EE

This report appeared in a special issue of the International Journal of Geographic Information Sciences, issue 4 of volume 13, in 1999.

A Commentary on GeoVRML: A Tool for 3D Representation of GeoReferenced Data on the Web

Theresa-Marie Rhyne
ACM SIGGRAPH Director at Large
Lockheed Martin Technical Services
US EPA Scientific Visualization Center
86 Alexander Drive
Research Triangle Park, North Carolina 27711
(trhyne@vislab.epa.gov)

Abstract:

GeoVRML techniques have the potential to provide functional and transparent communication between geographic information and 3D Web visualization tools. This report outlines recommended practices and modifications to the VRML 97 standard to consider pre-existing cartographic projections and georeferenced data. The concepts outlined for incorporating georeferenced coordinate systems in VRML worlds have generic applicability to 3D Web technologies like MPEG-4, Java3D and Chrome.

Introduction:

The interactive three dimensional (3D) representation of georeferenced data on the World Wide Web (Web) is achieved with tools like the Virtual Reality Modeling Language (VRML). VRML97 is the approved International Standard (ISO/IEC 14772) file format for describing interactive multimedia on the Internet. In general, a VRML file is also called a "world". Users explore these "worlds" with Web browsers that support the viewing of VRML files. More information on VRML can be found at the Web3D Consortium Web pages, see: (http://www.web3d.org).

The VRML97 standard was designed primarily by the computer graphics community. Typical computer graphics imagery focuses on locally bounded regions and small screen sizes where maximum pixel ranges are approximately 1600 by 1280 pixels. As a result, VRML97 relies on single-precision (32 bit) IEEE floating point data values. The coordinate system for VRML97 is based on the simple Cartesian local (X,Y,Z) coordinate system with the orgin being at (0,0,0) and Y representing up. This coordinate system is often sufficient for many computer graphics problems.

These two parameters of the VRML 97 standard provide limitations for the representation of geographic and cartographic data as well as georeferenced computational modeling simulations in VRML. For example, since the earth's diameter approximates 12 million meters, it is not possible to present geographic data resolutions greater than 10 to 100 meters with single-precision data values. This means that data obtained from global positioning systems (GPS) with absolute locations within 1

APPENDIX EE

meter resolution cannot be accurately presented in VRML97. The heavy reliance on Cartesian coordinates also poses difficulties with data in Geodetic (GDC or lattitude/longitude), Universal Transverse Mercator (UTM), Lambert Conformal Conic (LCC) or other pre-existing cartographic projections. In February 1998, the VRML Consortium approved the formation of the GeoVRML Working Group to discuss and develop tools, recommended practices and standards necessary to generate, display and exchange georeferenced data in VRML, (Iverson & GeoVRML, 1998). In December 1998, the VRML Consortium expanded its charter and renamed itself as the "Web 3D Consortium".

This report reviews the major recommended practices and modifications to the VRML standard under consideration and development by the GeoVRML Working Group. Additional emerging 3D Web technologies and their relation to geospatial data visualization will also be highlighted. GeoVRML techniques have the potential to provide functional and transparent communication between geographic information and 3D visualization tools, (Rhyne, 1997).

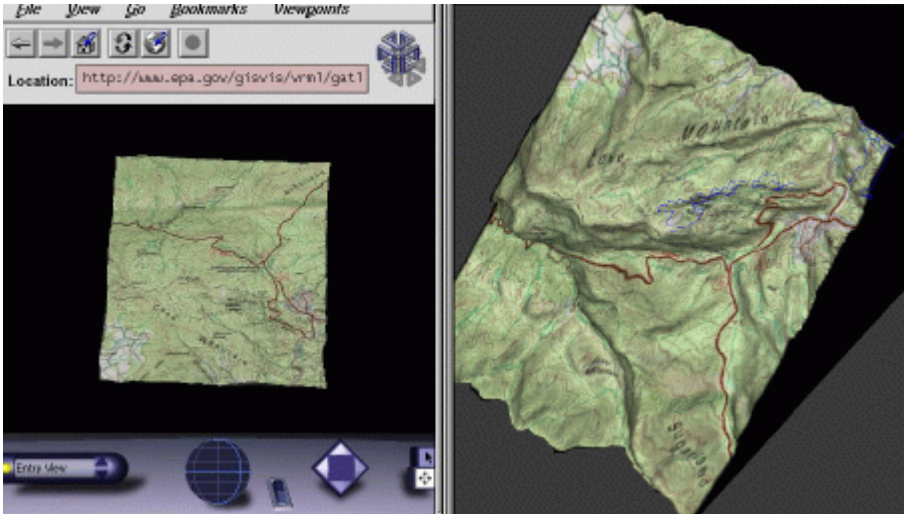


Figure #1: Example VRML world with a TIFF image of a USGS map drapped over the 3D surface is shown on the left. On the right is a similar image made with a visualization toolkit package. Notice how the map is inverted in the VRML browser. We hope to improve this situation with GeoVRML Coordinate systems. Images developed by Theresa-Marie Rhyne and Thomas Fowler of Lockheed Martin Technical Services at the United States Environmental Protection Agency's Scientific Visualization Center. See: (http://www.epa.gov/gisvis).

Moving towards GeoVRML Coordinate Systems:

The geographic information systems, cartographic and military simulation communities have developed a number of standards for the representation of geospatial information and georeferencing of arbitrary data, (Rhyne, 1998). The Open GIS Consortium is presently moving forward with efforts to support the full integration of geospatial data into mainstream computing and the widespread usage of interoperable commercial geoprocessing software, see: (http://www.opengis.org/). There are also International Organization for Standardization (ISO) efforts in the Geographic information/Geomatics arenas, see: (http://www.iso.ch/meme/TC211.html).

In order to attempt to include a methodology for supporting georeferenced data in the upcoming (1999) revision to the VRML97 standard, the GeoVRML Working Group decided to base its efforts on a currently existing reference model and software package entitled the SEDRIS Geographic

APPENDIX EE

Reference Model (GRM). The Synthetic Environment Data Representation & Interchange Specification (SEDRIS) is a project funded by the United States' Defense Modeling and Simulation Office. The SEDRIS GRM supports twelve different coordinate systems and provides tools to automatically convert reference marks between them. The software source for GRM is publically available and is currently implemented in the C programming language. More information on the SEDRIS GRM can be found on the SEDRIS Web site at: (http://www.sedris.org/).

The GeoVRML Working Group is thus recommending a Level 1 practice whereby geographical coordinates based on the SEDRIS GRM are converted into a local Cartesian coordinate system for improved level of detail in GeoVRML visualizations. The GeoVRML Working Group is also exploring a Level 2 practice whereby geo-referenced data is transparently and seamlessly converted from a wider and multiple variety of sources, (Iverson & GeoVRML, 1998).

Researchers at the SRI International - Artificial Intelligence Center, have recently developed, for public release, the GeoTransform Java class file hierarchy based on the SEDRIS GRM. With the GeoTransform Java package, it is possible to perform efficient and accurate geographic coordinate transformations for the Geodetic Coordinate System (GDC), GeoCentric Coordinate System (GCC), and Universal Transverse Mercator (UTM) System. GeoTransform allows for authoring VRML worlds that read coordinates in any of these systems and tranparently convert the geographic data into Cartesian Coordinates for display in a VRML browser. More information on the GeoTranform Java package can be found at: (http://www.ai.sri.com/~reddy/geovrml/geotransform/) .

Defining the GeoOrigin Node:

In order to build a georeferenced VRML world, a GeoOrigin node is defined in the VRML file. This GeoOrigin allows for converting coordinates from cartographic earth-based coordinate systems into the existing VRML97 Cartesian reference frame, (Iverson & GeoVRML, 1998). A single GeoOrigin node, representing a single georeferenced point, becomes the reference frame identified with the VRML world's zero-based (0,0,0) origin.

GeoOrigin

```
EXTERNPROTO GeoOrigin [
  field MFString geoSystem ["GDC"]
  field SFString geoCoords ""
] "urn:geovrml:protos#GeoOrigin"
```

The geoSystem field selects a geographic reference system from the naming conventions based on the SEDRIS GRM. Some of these georeference coordinate systems require additional arguments to fully designate the coordinates. As an example, the Geodetic (GDC) system involves the selection of ellipsoid, geoid, and datum references. Additional strings in the geoSystem field support this requirement.

The geoCoords field is a sequence of 64-bit precision values seperated by spaces that define an absolute location using the coordinate system selected in the geoSystem field. Optional strings in the geoSystem field determine the interpretation of the geoCoords field. As an example, "DMS" can specify that the geoCoords string will include degree, minute and second fields for each latitute and longitude value in a GDC coordinate. Every geospatial location determined by a geoSystem and geoCoords pair defines an implicit orthogonal Cartesian reference frame indexed by x,y,z in meters

Microsoft Corp. Exhibit 1009

with the designated geospatial location at the origin and with y being the up direction. This allows for conformance with the VRML97 standard.

A more detailed discussion about the GeoOrigin node can be found in the Request for Comment document on GeoVRML Coordinate Systems. This discussion is located at the GeoVRML Working Group's Web site at: (http://www.ai.sri.com/~leei/geovrml/).

During the past year, SRI International developed a series of VRML97 nodes for improved support of terrain visualization. These contributions were developed as part of the GeoVRML Working Group and are in the public domain. A comprehensive discussion of these efforts can be found in the SRI International - Artificial Intelligence Center Report No. 559, which is cited the the references below, (Reddy, et. al., 1998). These new GeoVRML nodes can be accessed on the Web at: (http://www.ai.sri.com/geovrml/protos) .

Integrating Spatial Data Repositories and GeoVRML Visualizations:

There are a number of efforts underway to examine the use of the Virtual Reality Modeling Language (VRML) for the interactive exploration of geospatial data repositories (Rhyne & Fowler, 1996). In the United States, some of this work is being done in conjunction with the Federal Geographic Data Committee (FGDC) 's National Geospatial Data Clearinghouse (see: (http://fgdc.er.usgs.gov/)). In addition to the use of intelligent agents, data mining techniques are being employed to assist with the retrieval of spatial data. The development of GeoOrgin nodes in VRML will support the use of agent and data mining technology for rapid creation of interactive web-based visualizations. This will greatly facilitate visual information retrieval of geospatial data.

Reaching out to other Interactive 3D Web Technologies:

In addition to VRML, there are other 3D Web technologies under development. Three examples include (a) the development of the MPEG-4 standard; (b) Java 3D and (c) Chrome. In early 1998, the International Organization for Standardization (ISO) announced that it will use Apple Computer's QuickTime file format as the basis for a unified digital media storage format for the MPEG-4 standard for graphics content on the Web. The VRML Consortium has established a Working Group to examine MPEG-4 and VRML integration. Java3D, from Sun Microsystems, supports the development of 3D computer graphics applications in the Java programming language. This includes the development of VRML browsers with Java 3D. Another emerging 3D Web technology is Chrome from Microsoft Corporation. Chrome is a Windows 98 add-on that uses the Extensible Markup Language (XML) to access Windows 98 multimedia capabilities for creating 3D content on the Web. The concepts outlined above for incorporating georeferenced coordinate systems in VRML worlds have generic applicability to 3D Web technologies like MPEG-4, Java3D and Chrome. Details about the QuickTime file format and its adoption by ISO as the starting point for MPEG-4 can be found at the Apple Computer web site, see: (http://www.apple.com/quicktime/). More information on Java and Java 3D can be found at the Javasoft Web site, see: (http://www.javasoft.com/products/java-media/3D/). Additional information on Chrome can be found by searching the Microsoft web site at: (http://www.microsoft.com).

Concluding Remarks:

The use of VRML for cartographic and geographic presentation is currently being examined by research groups participating in the International Cartographic Association's Commission on Visualization, (Fairburn and Parsley, 1997). Preliminary definitions of the needs for geofunctions in

virtual reality and VRML were done at Leicester University in July 1997, (Moore, et. al.). The Commission has also explored other multimedia and web-based technologies for developing mapping products, (Cartwright, 1998) and (Andrienko & Andrienko, 1998). The Association for Computing Machinery's Special Interest Group on Graphics (ACM - SIGGRAPH)'s collaboration with the ICA Commission on Visualization has attempted to examine how computer graphics technology can be effectively adapted to meet cartographic needs and requirements. This project, entitled the ACM SIGGRAPH Carto Project, is pleased that the VRML Consortium chose to create the GeoVRML Working Group to actualize effective exchange of georeferenced data in VRML. We anticipate GeoVRML techniques expanding to address many 3D Web Technologies as the VRML Consortium redefines itself as the Web 3D Consortium. The issues discussed here are important steps toward functional integration of geographic information and 3D visualization tools. We hope similar efforts will continue to emerge in the future.

Acknowledgements:

We would like to acknowledge the efforts of Lee Iverson, founding Chair of the GeoVRML Working Group of the Web 3D Consortium, Don Brutzman, Vice President for Technology of the Web 3D Consortium, and Martin Reddy (who built many of the new GeoVRML nodes for VRML97). We are also appreciative to Judy Brown, Past Chair of Special Projects for ACM SIGGRAPH, for all the encouragement she provided during the first two years of the ACM SIGGRAPH Carto Project.

References:

ACM SIGGRAPH Carto Project Web Site: (http://www.siggraph.org/~rhyne/carto/).

Andrienko & Andrienko. 1998, Descartes -Intelligent Mapping and Visual Data Exploration on the Internet, Proceedings of the 1998 Polish Spatial Information Association Conference, May 1998, Warsaw Poland, : 339 - 340.

Cartwright, W. 1997. New media and their application to the production of map products. Computers &; Geosciences, special issue on Exploratory Cartographic Visualization 23(4) : 447-456.

Fairbairn, D. and Parsley, S. 1997. The use of VRML for cartographic presentation. Computers &; Geosciences, special issue on Exploratory Cartographic Visualization 23(4): 475-482.

GeoVRML Working Group of the VRML Consortium Web Site: (http://www.ai.sri.com/geovrml/).

Iverson, Lee & the GeoVRML Working Group of the VRML Consortium. 1998, GeoVRML RFC1: Coordinate Systems, (http://www.ai.sri.com/geovrml/rfc1.html).

ICA Commission on Visualization Web Site: (http://www.geog.psu.edu/ica/ICAvis.html).

Moore, K., Dykes, J., Wood, J., Bastin, L., Fisher, P. 1997, VR Geofunctions, (http://www.geog.le.ac.uk/mek/VRGeoFunctions.html).

Reddy, M., Leclerc, Y. G., Iverson, L., Bletter, N., and Vidimce, K. 1998, Modeling the Digital Earth in VRML, AIC Technical Report No. 559. SRI International, Menlo Park, CA. November 1998.

Rhyne, T.-M. and Fowler, T. 1996, Examining Dynamically Linked Geographic Visualization, Proceedings of the 1996 Computing in Environmental Resource Management Speciality Conference

APPENDIX EE

sponsored by the Air & Waste Management Association, Dec. 1996, Research Triangle Park, North Carolina (USA), : 571 - 573.

Rhyne, T.-M. 1997. Going virtual with geographic information and scientific visualization. Computers & Geosciences, special issue on Exploratory Cartographic Visualization 23(4): 489-492.

Rhyne, T.-M. 1998, Open Spatial Data Standards for the Information Highway (Examining Dynamically Linked Geographic Visualization), Proceedings of the 1998 Polish Spatial Information Association Conference, May 1998, Warsaw Poland, : 297 - 299.

Biography of the Author:

Theresa-Marie Rhyne is a Director at Large of the ACM SIGGRAPH Executive Committee and is the Project Director of the ACM SIGGRAPH Carto Project. She is a lead scientific visualization researcher for Lockheed Martin Technical Services at the United States Environmental Protection Agency's Scientific Visualization Center.