(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 1 195 717 A2**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
10.04.2002 Bulletin 2002/15

(51) Int Cl.^{7}: **G06T 15/00**

(21) Application number: **01120048.2**

(22) Date of filing: **21.08.2001**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **04.10.2000 US 679315**

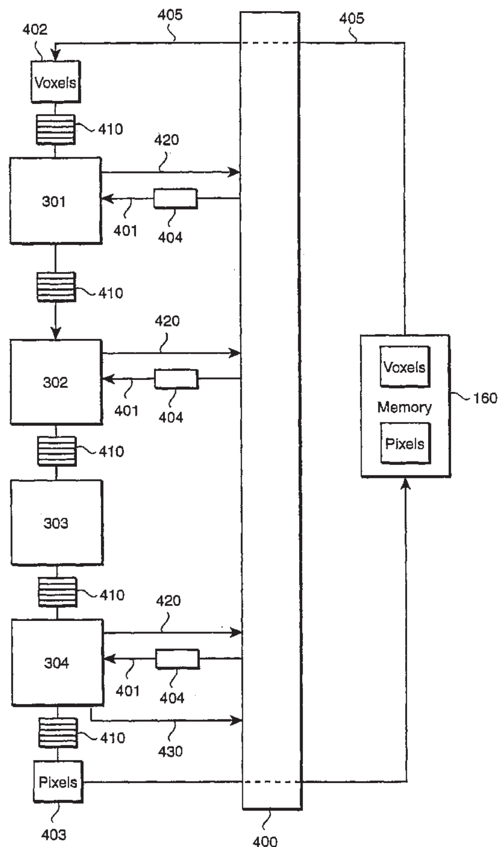
(71) Applicant: **TeraRecon, Inc.**
San Mateo, California 94403 (US)

(72) Inventors:
- **Seiler, Larry D.**
  Boylston, Massachusetts 01505 (US)
- **Burgess, Stephen R.**
  Medfield, MA 02052 (US)
- **Lussier, Jeffrey**
  Woburn, MA 01801 (US)
- **Bhatia, Vishal C.**
  Arlington, MA 02474 (US)

(74) Representative: **Gleiter, Hermann**
Pfenning, Meinig & Partner GbR
Mozartstrasse 17
80336 München (DE)

(54) **Controller for rendering pipelines**

(57) The invention provides a method and apparatus for rendering graphic data as an image. Graphic data that possibly contribute to the image is identified. The identified graphic data is read into a rendering pipeline. Samples are generated in the rendering pipeline only if they possibly contribute to the image for the identified graphic data. The identified graphic data and samples are processed in the rendering pipeline only as long as the identified graphic data and sample continue to contribute to the image. All other identified graphic data and samples are discarded from the pipeline.



DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

Description

Cross-Reference to Related Application

5 **[0001]** This is a continuation-in-part of U.S. Patent Application. 09/410,770 *"Voxel and Sample Pruning in a Parallel Pipelined Volume Rendering System,"* filed by Lauer et al. on October 1, 1999.

Field of the Invention

10 **[0002]** The present invention is related to the field of computer graphics, and in particular to rendering graphic data with a parallel pipelined rendering engine.

Background of the Invention

15 **[0003]** Volume rendering is often used in computer graphics applications where three-dimensional data need to be visualized. The volume data can be scans of physical or medical objects, or atmospheric, geophysical, or other scientific models where visualization of the data facilitates an understanding of the underlying real-world structures represented by the data.

[0004] With volume rendering, the internal structure, as well as the external surface features of physical objects and 20 models are visualized. Voxels are the fundamental data items used in volume rendering. A voxel is data that represent values at a particular three-dimensional portion of the object or model. The coordinates (x, y, z) of each voxel map the voxels to positions within the represented object or model.

[0005] A voxel represents one or more values related to a particular location in the object or model. For a given prior art volume, the values contained in a voxel can be one or more of a number of different parameters, such as, density, 25 tissue type, elasticity, or velocity. During rendering, the voxel values are converted to color and opacity (RGBα) values in a process called classification. These RGBα values can be blended and then projected onto a two-dimensional image plane for viewing.

[0006] One frequently used technique during rendering is ray-casting. There, a set of imaginary rays are cast through the array of voxels. The rays originate from some view point or image plane. Sample points are then defined along the 30 ray. The voxel values are interpolated to determine sample values, and the sample values along each ray are combined to form pixel values.

[0007] U.S. Patent Application Sn. 09/315,742, *"Volume rendering integrated circuit,"* filed on May 20, 1999 by Burgess et al., incorporated herein by reference describes a rendering system that uses parallel pipelines. The rendering system includes a host processor connected to a volume graphics board (VGB) by a bus. The VGB includes a voxel 35 memory and a pixel memory connected to a Volume Rendering Chip (VRC). The VRC includes all logic necessary for performing real-time interactive volume rendering operations. The VRC includes four interconnected rendering pipelines. In effect the VGB provides a rendering engine or "graphics accelerator."

[0008] During operation, application software executing in the host transfers volume data to the VGB for rendering. The application software also loads rendering registers accessible by the pipelines. These registers specify how the 40 rendering is to be performed. After all data have been loaded, the application issues a command to initiate the rendering operation. When the rendering operation is complete, the output image is moved from the pixel memory to the host or to a 3D graphics card for display.

[0009] One problem with prior art hardware rendering pipelines is that frequently "bubbles" appear in the pipelines. Bubbles are due to the fact that data are not available on a given clock cycle. Once a bubble is introduced, it has to 45 pass all the way through the pipeline. Consequently, bubbles waste time, and reduce the performance of the system.

[0010] Another problem with prior art hardware pipelines is that they typically process all voxels in a data set. It is well known that for a given visualization of volume data, that there are clusters of voxels that contribute useful information to the image and other clusters that are totally irrelevant. For example, in medical data sets, the percentage of voxels that do not contribute to the final image is typically in the range of 70-95%. Thus, eliminating unnecessary voxel/ 50 sample processing could eliminate up to 90% of the work.

[0011] Therefore, there is a need for a rendering system that can dynamically adapt to the complexities of the rendering data, and furthermore there is a need for a pipelined rendering system that does not process unnecessary data.

Summary of the Invention

55

[0012] The invention provides a method and apparatus for rendering graphic data as an image. Graphic data that possibly contribute to the image is identified. The identified graphic data is read into a rendering pipeline. Samples are generated in the rendering pipeline only if they possibly contribute to the image for the identified graphic data. The

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

identified graphic data and samples are processed in the rendering pipeline only as long as the identified graphic data and sample continue to contribute to the image. All other identified graphic data and samples are discarded from the pipeline.

Brief Description of the Drawings

[0013]

Figure 1 is a block diagram of a pipelined rendering system that uses a controller according to the invention;

Figure 2 is a block diagram of a rendering engine;

Figure 3 is a block diagram of stages of rendering pipelines;

Figure 4 is a block diagram of a controller connected to rendering pipelines;

Figure 5a-b are block diagrams of sample slices and voxel slabs;

Figures 6a-b are block diagrams of sample stamps and tiles;

Figures 7a-b, and 8 are block diagrams of rays passing through voxels;

Figure 9 is a block diagram of a controller according to the invention;

Figures 10-11 are block diagrams of controller state machines;

Figure 12 is a block diagram of instruction tags;

Figure 13 is a block diagram of stamp motion;

Figure 14 is a block diagram of section motion; and

Figure 15 is a block diagram of a controller execution unit.

Detailed Description of the Preferred Embodiment

Pipeline Organization

[0014] Figure 1 shows the overall organization of a volume rendering system 10 using a controller (CTRL) 400 according to our invention. The system includes a host computer 100 connected to rendering subsystem 200 by a bus 121. As an advantage, the rendering subsystem is fabricated as a single ASIC. The host includes a CPU 110 and a main memory 120.

[0015] As also shown in Figure 2, the principal modules of the rendering subsystem 200 are a memory interface 210, bus logic 220, a controller 400, and four parallel hardware pipelines 300. Except for shared slice buffers 250, which span all four pipelines, the pipelines (A, B, C, and D) operate independent of each other. The. pipelines form the core of our rendering engine.

Memory Interface

[0016] The memory interface 210 controls eight double data rate (DDR) synchronous DRAM channels that comprise an off-chip rendering memory 160. The rendering memory provides a unified storage for all data 211 needed for rendering volumes, i.e., voxels, pixels, depth values, look-up tables, and command queues. The memory interface 210 implements all accesses to the rendering memory 160, arbitrates the requests of the bus logic 220 and the controller 400, and distributes array data across the modules and the rendering memory 160 for high bandwidth access and operation.

Find authenticated court documents without watermarks at docketalarm.com.

Bus Logic

[0017] The bus logic 220 provides an interface with the host computer system 100. If the host is a personel computer (PC) or workstation, then the bus can be a 64-bit, 66 MHz PCI bus 121 conforming to version 2.2 of the PCI specification. The bus logic also controls direct memory access (DMA) operation for transfering data to and from the rendering memory 160 via the memory interface 210. The DMA operations are burst-mode data transfers.

[0018] The bus logic also provides access to internal register files 221 of the controller 400. These accesses are direct reads and/or writes of individual registers initiated by the host computer 100 or by some other device on the PCI bus. The bus logic 220 also interprets access commands for efficient control of data transfers. The bus logic also sends register values directly to the controller 400 for controlling rendering operations and receives status back from the controller.

Controller

[0019] The controller 400 controls the operation of the volume rendering engine 300 using control signals 401. Note, the controller is coupled to the pipelines in a parallel manner. The controller determines what data to fetch from the memory, dispatches that data to the four pipelines, sends control information, such as interpolation weights, to the individual pipeline stages at the right time, and receives output data and status from rendering operations.

[0020] A major function of the controller is to discard as much data as possible. By discarding data that are not needed rendering can be greatly accelerated.

[0021] The controller, in part, is implemented as a finite state machine controlled by a large number of registers. These are typically written by the bus logic 220 in response to load register commands of a command queue. Internally, the controller maintains the counters needed to step through sample space one section at a time, to convert sample coordinates to voxel coordinates, and to generate the control information needed by the stages of the pipelines. The controller 400 is described in greater detail below.

[0022] The controller is designed to operate, time-wise, well in advance of the pipelines 300. Thus, the controller can determine what samples and voxels are needed, and those that can be discarded. Recall, as many as 90% of the voxels in certain classes of volume data do not affect the resulting image. Not reading voxels saves memory bandwidth, and not processing samples saves pipeline cycles. In effect, the controller attempts to dynamically "prune" the volume data to a bare minimum.

[0023] Some samples and voxels may enter early stages of the pipeline, before this determination can be made. In that case, the samples and voxels are discarded in later stages, perhaps causing "bubbles." However, because the various stages of the pipeline are buffered and may operate at different rates, bubbles can sometimes be squeezed out to greatly decrease the amount of time it takes to render a volume. Because the peak rate at which the controller produces commands is faster than the pipelines can process commands, bubbles can be replaced with good data so that the performance of the pipelines is maximized.

[0024] As an additional feature, the controller can operate asynchronously with respect to the pipelines. This greatly simplifies the timing relationship. In fact, the pipelines can be though of as having variable lengths (in terms of cycles). For some operations the pipelines are (time-wise) shorter than for others. The controller is capable of time-aligning the control signals with the data even though the control signals are generated well in advance. Even though the controller does not know in advance how many clock cycles it will take for certain data to reach a particular stage in any of the pipelines. The signals are buffered so that they arrive at the stage when they are needed by the data.

Pipelines, Miniblocks and Stamps

[0025] Figure 3 shows the four rendering pipelines of the rendering engine in greater detail, and it also shows how data and rendering operations are distributed among the piplines. Each pipeline includes a gradient estimation stage 301, a classifier-interpolator stage 302, an illuminator stage 303, and a compositor stage 304.

[0026] Voxels are stored in the rendering memory 160 as miniblocks 310, that is, small cubic arrays of $2 \times 2 \times 2$ voxels each. During rendering, the controller 400 causes the memory interface to read streams of miniblocks. The miniblocks are presented to the pipelines at the rate of one miniblock per clock cycle. In actual fact, the mini-blocks are passed to the pipelines via the controller 400.

[0027] Miniblocks are read from the volume data set in *x*-*y*-*z*-order. That is, they are read sequentially in the *x*-direction to fill up a row of a section, and row-by-row in the *y*-direction to fill a slice, and slice-by-slice in the *z*-direction to render the entire section. Each miniblock is decomposed into four $1 \times 1 \times 2$ arrays of voxels 320, that is, four pairs of voxels (A, B, C, and D) aligned in the *z*-direction. One pair 320 of voxels is forwarded to each pipeline as shown in Figure 3.

[0028] Each pair of voxels is passed through the gradient estimation stage 301 to obtain gradient values at each

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

voxel. As a result of a central difference filter used to obtain gradients, the output voxels and gradients are offset by one unit in each dimension from the inputs. This requires a small amount of data exchange between pipelines.

[0029] From the gradient estimation stage, the voxels and gradients are passed to the classifier-interpolator 302. In this stage, voxel fields are converted to RGBα values and, along with gradients, are interpolated to values at sample points along rays. The interpolator first performs interpolation in the Z-direction, and then in the Y and X directions. The classification and interpolation steps can occur in either order. Note that the classifier-interpolator has one pair of slice buffers 250 that are shared among all four pipelines, as well as unshared buffers that store the voxel data used for Z interpolation.

[0030] The output of the four classifier-interpolators of the four pipelines is an array of RGBα values and gradients at a 2×2 array of points in sample space called a *stamp.* The points of a stamp always lie in a plane that is parallel to the voxel slab, at XY positions corresponding to the intersection of the slice with four of the rays being cast through the volume. When the rays are defined so as to pass through pixels on the image plane, we call it *xy-image order,* because the *x*- and *y*-coordinates of the rays are the same as those of image space. Ordinary image order, as known in the prior art, selects points in sample space on planes that are parallel to the image plane, rather than on planes that are parallel to the *xy* planes in the volume.

[0031] The stamp of RGBα values and gradients is next passed to the four illuminators 303. These apply the well known Phong lighting using reflectance maps. The illuminator of each pipeline is independent of those of the other pipelines, in the sense that they do not exchange data during rendering. The pipelines all operate synchronously according to the same clock.

[0032] The gradients are consumed in the illuminator stages, except when the rendering operation specifies the output of gradients. In this case, the three gradient components are substituted for the red, green, and blue color components in the pipelines.

[0033] The output of the illuminator stage of each pipeline is an illuminated RGBα value representing the color contribution of its sample point. The RGBα value is passed to the compositor stage 304. The compositor accumulates the RGBα values of the rays into an on-chip buffer. At the end of rendering a section, the outputs of the four compositor stages are read out, a stamp at a time, for storage in the rendering memory 160 as, for example, pixel values.

Controller-Pipeline Interface

[0034] Figure 4 shows how the controller 400 is connected in parallel to the various stages 301-304 of the pipelines 300. For clarity, the interconnects between the controller and the pipeline are shown at an abstract level. The actual implementation includes a large number of parallel interconnect lines and more separate interconnects, see Figure 9 for a next level of detail.

[0035] The raw input data, e.g., voxels 402 from the rendering memory 160 pass through the controller 400 on the way into the pipelines 300 via bus 405. The stages 301-304 convert voxel values to sample values, and combine sample values to pixel values 403. The pixels are written back to the rendering memory via the controller.

[0036] In contrast with the prior art, the present rendering engine 300 is adaptively elastic. The controller 400 issues output control signals 401 to the pipelines 300. The output control signals are transferred to the pipelines via queues 404. These are first-in-first-out (FIFO) queues. The output control signals are used to control the operation of the pipeline stages 301-304. Input control signals 420 are received from the pipeline stages. The input control signals indicate when each corresponding queue 404 is about to become full, so that the controller should stop sending data.

[0037] The output control signals 401, individually or as sets, include *tags* described in greater detail below. The tags indicate the beginnings and ends of the various types of data structures into which the volume data are organized, such as sections, slices, and slabs, described in further detail below. The tags also mark types of data processed inside the controller, including stacks, tiles, stamps, etc., also described in further detail below.

[0038] The purpose of the tags in the queues 404 is to time-align the output control signals 401 with the data in the various stages of the pipelines. Buffers 410 provide elasticity in the pipeline. For clarity, the buffers 410 are shown between the stages, but in the preferred embodiment, some of the stages, such as the interpolator, have internal buffers. The buffers provide a place to store data when a next stage is not yet ready to accept the data. It is the buffers, in part, that give the pipelines a variable length or elasticity. The preferred implementation can save gates by eliminating the buffers between some of the stages, in particular stages where buffers do not help eliminate bubbles, e.g., between the classifier/ interpolator and the illumination stages.

[0039] During operation, depending on unknown dynamics of data availability, bus loads, and computation complexity, the various stages can process the data at different rates. Thus, if a down stream stage is still busy, then an upstream stage can continue to process and write its output to one of the buffers 410. Then, when the downstream stage completes the previous task, the input data that the downstream stage needs will be readily available.

[0040] The tags ensure that the data are always synchronized with respect to each other, even when the stages operate asynchronously with respect to each other, and with respect to the controller 400. Additional input control lines

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.