```
/Helvetica findfont     % Find the font object "Helvetica"
                        % previously defined.
188 scalefont           % Make it 188 times as big—default font size is
                        % one point.
setfont                 % Make this the current font (stored as part of
                        % the "graphics state").
100 20 moveto           % Move to a point on the page.
67 rotate               % Rotate coordinates by 67 degrees.
(Hello) show            % Place the word "Hello" at the current
                        % point (100, 20) and render it on the page.
```

Fig. 19.70 A POSTSCRIPT program.

5. *Character and font operators*. This special class of operators is used for specifying, modifying, and selecting fonts. Since fonts are graphical entities like all others, the operators that place text into the current ''path'' are really path-construction operators; they are placed in a special class only because of the specialized nature of fonts.
6. *Device-setup and output operators*. The setup operators establish the correspondence of raster memory with the output device. The output operators control data transfer from memory to the device.

To exhibit the power of the POSTSCRIPT model, we give examples of the use of several of these operators. Each example includes running comments flagged by the ''%'' comment delimiter in POSTSCRIPT.

The first example, shown in Fig. 19.70, generates some text on the page, as shown in Fig. 19.71. The **show** operator is a special one: it both defines the object (a text string) to be shown and renders that object into raster memory. In all examples, a box has been drawn to show the outline of the page, although the instructions that drew the box are not included

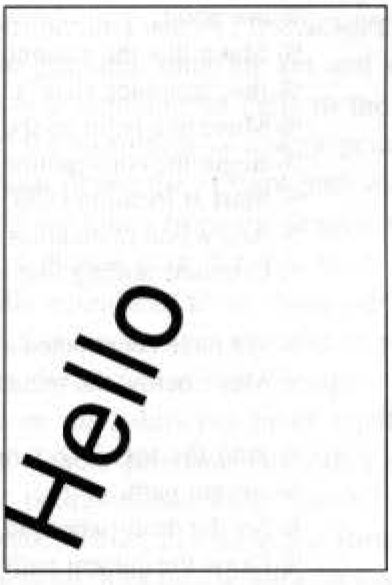


Fig. 19.71 The output of the first POSTSCRIPT program.

in the examples. A slash (/) preceding a name pushes that name onto the stack as a literal object.

In Fig. 19.72, we build a rectangle and its label, and draw both with a dotted line, as shown in Fig. 19.73. To do this, we build up the "current path" using path-construction operators. The first of these is the **lineto** operator, which adds a line from the current point to the specified point. The second is the **charpath** operator, which takes two arguments, a text string and a Boolean; if the Boolean is **false,** the outline of the text string in the current font is added to the current path. When the Boolean is **true,** the outline of the text string is converted to a special type of path suitable for use in clipping. (We shall see this outline text in the third example.) The **stroke** operator is used to render the current path and to begin a new current path. (The current path can also be rendered using the **fill** or **eofill** operator. Both of these require that the path be closed. The first fills according to the nonzero winding rule; the second fills by the even–odd fill rule.)

Figure 19.74 shows how text is used for clipping; the result is shown in Fig. 19.75. Clipping to text requires the use of the clip path, which is another part of the POSTSCRIPT graphics state. Initially, the clip path encloses everything. We can create clipped objects by defining a smaller clip path. Here we define a procedure for drawing three horizontal strips, and then display them through two different clipping regions: a cubic curve, which is closed off using the **closepath** operator, and later a large piece of text. Since the **clip** operator always reduces the clipping region to the intersection of the current clipping region and the current path, it is important to save the original clipping region so that we can restore it. To do this, we use the **gsave** and **grestore** operators, which save and restore the entire graphics state.

The procedure definition is somewhat arcane. The **def** operator expects two arguments on the stack: a name and a definition. When used to define a procedure, the name is pushed

```
/Helvetica findfont               % Look for an object named "Helvetica"
                                  % previously defined.
120 scalefont                     % Make it 120 times as big—default font size is
                                  % one point.
setfont                           % Make this the current font (stored as part of
                                  % the "graphics state").
0 0 moveto                        % Move to a point on the page.
newpath                           % Begin the construction of a path.
100 300 moveto                    % Start at location (100, 300).
500 300 lineto                    % Add a line to location (500, 300).
500 800 lineto                    % Continue adding line segments.
100 800 lineto
100 300 lineto                    % We have constructed a rectangle.
100 50 moveto                     % Move below the rectangle.
(Rectangle) false charpath
                                  % Add the text "Rectangle" to the
                                  % current path.
[9 3] 0 setdash                   % Set the dash-pattern to 9-on, 3-off, repeating
stroke                            % Draw the current path in this dash-pattern.
```

Fig. 19.72 A more complex POSTSCRIPT program.

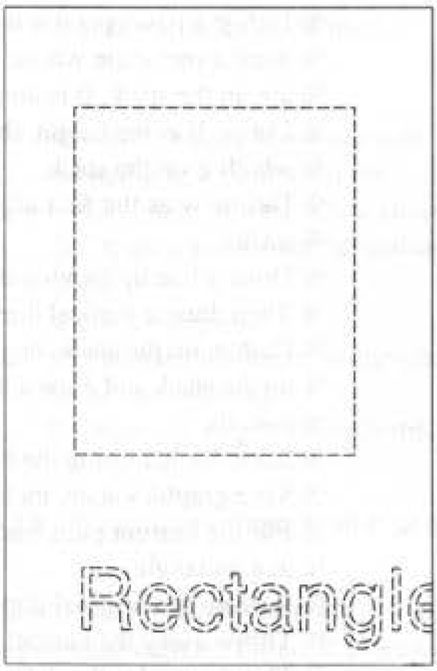


Fig. 19.73 The output of the second POSTSCRIPT program.

on the stack (with a preceding slash to prevent interpretation) and then the body of the procedure (which is a single item—all the material between matched braces). The **def** operator pops these two operands and defines the name to mean the body.

Often, the two operands to **def** are in the opposite order, as in the case where a procedure is called with its operands on the stack. To bind these operands to local variables, we push the variable name on the stack, then invoke the **exch** operator, which exchanges the top two elements on the stack, and then invoke **def.**

These examples give a good sense of the power of the POSTSCRIPT model. POSTSCRIPT is essentially like assembly language—it is powerful, but programming in it is not too pleasant. On the other hand, defining procedures is simple, which makes POSTSCRIPT far easier to use. In fact, the recommended use of POSTSCRIPT is in the form of a prologue, in which various attributes of the graphics state are set and various procedures are defined, followed by the script, which is a sequence of calls to the defined procedures.

The *current path* used in the examples is a very general object. It can be drawn in several ways: the **stroke** operator draws the current path as a sequence of lines and curves whose thickness and patterning are taken from the graphics state; the **fill** operator fills the current path (which must be a closed path for this to make sense) with the current fill pattern; the **eofill** operator fills according to an even–odd rule. As in the examples, the curves defining the outline of a character can be used as part of a path.

POSTSCRIPT also supports the notion of an ''image'' as a primitive, which can be a 1-bit-per-pixel representation or an *n*-bits-per-pixel representation. When an image is rendered into raster memory, it acts just like any other primitive; every pixel in raster memory covered by the image is drawn into with ''paint'' that can be white, black, or (in color implementations) any color. Thus, invoking the **image** operator draws an image on top of whatever is already present (it does no blending or compositing of the form described in Section 17.6). An alternate form, **imagemask,** is suitable for 1-bit images. Such a mask

```
/rect {                 % Define a new operator to
                        % draw a rectangle whose width and height
                        % are on the stack. It is invoked by "w h rect".
   /h exch def          % Define h as the height, the second argument
                        % which is on the stack.
   /w exch def          % Define w as the first argument from the
                        % stack.
   w 0 rlineto          % Draw a line by moving the pen by (w, 0).
   0 h rlineto          % Then draw a vertical line, moving by (0, h).
   w neg 0 rlineto      % Push w on the stack, negate it, push zero
                        % on the stack and draw a line, moving by
                        % (–w, 0).
   0 h neg rlineto      % Same for h, closing the box.
   gsave                % Save graphics state, including clip path
   fill                 % Fill the current path, reducing the clip region
                        % to a rectangle.
   grestore             % Restore the original clip path
   newpath              % Throw away the current path and start anew.
} def                   % This concludes the definition of rect.

/stripes {              % Define the "stripes" operator, which
                        % draws three stripes on the page. It
   newpath              % takes no arguments.
   100 300 moveto       % Go to a point on the screen.
   800 50 rect          % Draw a rectangle.
   100 200 moveto       % Move down a little and do it again.
   800 50 rect
   100 100 moveto
   800 50 rect          % Do it yet again.
} def                   % This concludes the definition of stripes.

0 0 moveto              % Start the current point at the origin.
.95 setgray             % Set the gray level to very pale.
stripes                 % Show the full stripes.
.4 setgray              % Set the gray shade a little darker.
gsave                   % Save the current graphics state
                        % (including the clip path).
newpath                 % Start a new path at 50,150.
50 150 moveto
100 250 300 275 250 175
curveto                 % Draw a Bezier curve with control points
                        % at (50, 150), (100, 250), (300, 275),
                        % and (250, 175).
closepath               % Close off the curve with a straight line.
clip                    % The new clip region is the intersection
                        % of the old clip region (everything) with
                        % the path just constructed.
stripes                 % Draw stripes through the new clipping
                        % region, slightly darker than last time.
grestore                % Restore the original clipping path.
```

Fig. 19.74 (*Cont.*)

```
.2 setgray                  % Darken the color further.
gsave
/Helvetica findfont
100 scalefont setfont       % Create a huge Helvetica font.
newpath                     % Start a new path to clip by.
200 80 moveto               % Get ready to write a few characters.
(ABC) true charpath         % Create a path from the outline of
                            % the text.
closepath                   % Close the path,
eoclip                      % and make this the new clip path, using
                            % the even-odd rule.
stripes                     % Draw the stripes through this.
grestore
```

Fig. 19.74 A complex POSTSCRIPT program.

draws (with the current gray-level) only where there are 1s in the image; in places where there are 0s, the raster memory is untouched. Thus, it effectively transfers the image as a pattern in **transparent** mode, except that this image can be transformed before the operation is performed. In fact, an image can be scaled and rotated, and any other operation (including clipping by the current clip path) can be applied to it.

This last remark brings us to an important question: How are POSTSCRIPT interpreters implemented? How would we draw thickened cubic splines, for example, or rotated images? The exact interpretation is not specified in the POSTSCRIPT *Language Reference Manual* [ADOB85b], because the implementation must necessarily be different for different classes of printers (and for displays). Still, certain aspects of the imaging model are given explicitly. Curves are drawn by being divided into very short line segments. These segments are then thickened to the appropriate width and are drawn, producing a thick curve. Thus, thick lines have small notches in their sides if they are very wide, and the polygonization of

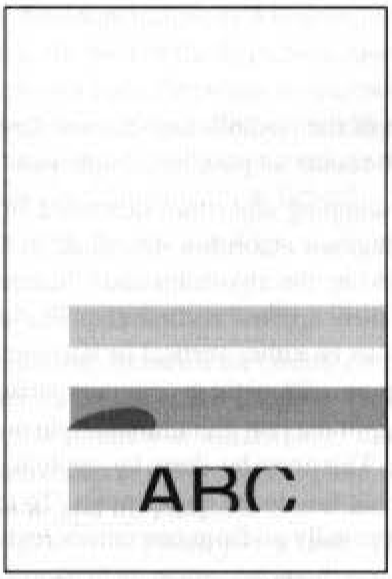


Fig. 19.75 POSTSCRIPT allows general clipping, even using text outlines as a clip path, as shown by this output from the third POSTSCRIPT program.

the curves is coarse (this, too, is user definable). Note that, in POSTSCRIPT, the line width is a geometric rather than cosmetic attribute, in the sense that it is defined in some coordinate system and can later be transformed. If we set the line width and then scale the current transformation by 2, all subsequent lines are drawn twice as thick.

POSTSCRIPT images apparently are transformed by affine maps using a technique similar to the two-pass transform. An image is defined, in the imaging model, as a grid of tiny squares, so a transformed image is drawn by rendering many small qudrilaterals. The method used for clipping by an arbitrary path is not stated so explicitly, but it appears that simple span arithmetic is performed in some implementations: The clipping region is represented by a collection of horizontal spans, each primitive is scan-converted row by row, and then only those pixels lying within the spans are actually painted. Much of the remaining implementation of the language amounts to a massive bookkeeping task; the current clip region, the current path, and the various fill styles must all be maintained, as well as the raster memory and the current transformation matrix.

19.10 SUMMARY

We have discussed some of the geometric and raster algorithms and data structures in use today, as well as their use in implementing various raster packages, including page-description languages. Despite the considerable sophistication of many of these algorithms, work continues to be done on these topics. Geometric clipping (and its extension to 3D, which is used in polyhedral constructive solid geometry) is a subject of active research, and new scan-conversion algorithms, especially with antialiasing, are still appearing frequently in the literature.

One lesson to be drawn from the algorithms is that simplicity may be more important than sophistication in many cases. Scissoring is a widely used clipping technique, the shape data structure makes many operations trivial, and the implementation of bitBlt we described has, as its core, the idea of making short and simple code that can be executed in a cache.

EXERCISES

19.1 Improve the implementation of the Nicholl–Lee–Nicholl line-clipping algorithm by making as much repeated use of intermediate results as possible. Implement all the cases.

19.2 The Liang–Barsky polygon-clipping algorithm described in the text may produce degenerate edges (so may the Sutherland–Hodgman algorithm described in Chapter 3). Develop an algorithm that takes the list of edges generated by the algorithm and "cancels" any two successive edges that have opposite directions, and is then applied recursively. The algorithm is made simple by the observation that such edge pairs must be either vertical or horizontal. Explain why this is true. Note that your algorithm should require no arithmetic except comparisons.

Also develop an algorithm to split the polygon into multiple output polygons if necessary, so that no two output polygons intersect. This may be done by applying the first algorithm to the output polygon, and then scanning the result for closed components. To test whether your algorithm works, apply it to several polygons that repeatedly go from one corner region to the adjacent or opposite one.

19.3 Reimplement the Liang–Barsky polygon-clipping algorithm without the use of infinity, by making special cases of horizontal and vertical lines.

19.4 Write an algorithm to convert a polygon in the plane, specified by a list of vertices, into the data structure for a contour described in the Weiler polgyon-clipping algorithm. Extend your algorithm to work for polygons with holes, specified by multiple paths.

19.5 Suppose that you are given a set and a partial ordering (called *contained in*) on the set: For any two elements, A and B, A is contained in B, B is contained in A, or neither A nor B is contained in the other. (This partial order has nothing to do with the *element-of* relation in set theory.)

Design an algorithm that constructs a binary tree whose nodes are labeled by the elements of the set, and that has the following two properties: The left child of a node is always contained in the node, whereas the right child of a node (which may be NULL) neither is contained in nor contains the parent. A good algorithm for constructing this tree can be used to improve the postprocessing step in the Weiler polygon algorithm, where the contained/coexists structure of the contours must be determined.

19.6 Show by example that finding an integer starting point for a line with actual endpoint (x_0, y_0) as done in the text can give a different choice from $(\text{Round}(x_0), \text{Round}(y_0))$. Which choice is better, and why?

19.7 Suppose we have chosen to allow quarter-pixel resolution for endpoints of line segments, and that we consider the line from (0, 0) to (40, 1). Consider the portion of the line between $x = 15$ and $x = 40$. Show that, if we clip first and then compute the line equation from the clipped endpoints (after rounding to quarter-integers) and scan convert, the result will be different from that obtained by clipping, then using the rounded value of one clipped end as a start point, and scan converting using the equation of the original line.

19.8 Complete the ellipse specification given in Section 19.2.6 as follows. Let

$$\begin{pmatrix} x \\ y \end{pmatrix} = \cos(t)\begin{pmatrix} P_x \\ P_y \end{pmatrix} + \sin(t)\begin{pmatrix} Q_x \\ Q_y \end{pmatrix}.$$

Solve this equation for $\cos(t)$ and for $\sin(t)$ in terms of x, y, P_x, P_y, Q_x, and Q_y. Now use the identity $\cos^2 t + \sin^2 t = 1$ to create a single equation in x, y, P_x, P_y, Q_x and Q_y. Express this equation in the form $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$, and thus derive formulae for the coefficients A, B, C, D, E, and F in terms of P_x, P_y, Q_x, and Q_y. Your answers for D and E should be zero.

19.9 Describe methods for specifying hyperbolas and parabolas, and for computing the coefficients of the conic from the specification. One way to specify a hyperbola is to describe a parallelogram. The long diagonal of the parallelogram is the axis of the hyperbola, the two vertices of the hyperbola lie at the endpoints of this diagonal (the *vertices* are the points on the two branches of the hyperbola that are closest to each other), and the asymptotes are parallel to the sides of the parallelogram. A parabola can be specified by giving a focus (a point), a directrix (a line), and one point on the parabola, but this is not very intuitive. Can you come up with something better?

19.10 Compute the table of increments for the Van Aken conic algorithm. These are the increments in the decision variables for square and diagonal moves in each drawing octant. Show that the code that handles the first and second drawing octants actually will work for every odd or even octant.

19.11 Show that the three different error measures for circles give the same pixel selections when the radius and coordinates of the center of the circle are integers. This is simplest if you assume that the pixel under consideration satisfies $x \geq 0$, and $y \geq x$. (All other cases can be derived from this one by symmetry.) The radial error measure selects between (x, y) and $(x, y - 1)$ by evaluating $F(x, y) = x^2 + y^2 - r^2$ at each of the two points and choosing the one where the absolute value of F is smaller. The axial error measure uses $y' = \sqrt{r^2 - x^2}$; both $y - y'$ and $(y - 1) - y'$ are computed. The one with smaller absolute value is selected. The midpoint error measure selects (x, y) or $(x, y - 1)$, depending on whether the value of $F(x, y - 0.5)$ is negative or positive. What choice do you have to make to ensure that the three ambiguous cases agree (i.e., the cases where $F(x, y) = F(x, y - 1)$, where $y' = y - 0.5$, and where $F(x, y - 0.5) = 0$)? Or is there no consistent choice? At least show that, in the ambiguous cases, there is some pixel that minimizes all three measures of error.

19.12 Study the circle of radius $\sqrt{7}$ centered at the origin by scan converting it with any algorithm you choose. Is the result aesthetically pleasing? What happens if you move the center to $(\frac{1}{2}, 0)$? Are the sharp corners in the scan-converted version irritating? Should circle algorithms try to make curvature constant rather than minimizing distances? This circle, along with a few others whose squared radii are integers, is particularly troublesome. McIlroy [MCIL83] shows that the next case with such sharp corners is $r^2 = 7141$.

19.13 Complete the Van Aken conic algorithm by adding the portion that finishes the conic in the last drawing octant.

19.14 Improve the Van Aken conic algorithm by adding a check for octant jumps. DaSilva [DASI88] recommends using the decision variable to decide which pixel to draw next, but then checking to see whether that pixel has jumped over an octant. If it has, the alternate pixel is chosen instead.

19.15 Write a general line-drawing algorithm. Your algorithm should handle the case where the coordinates of the endpoints are rational numbers with a fixed denominator, p. Be certain that if you draw two lines that share an endpoint, the rasterized lines have the identical endpoints as well. Initializing the decision variable may be the most difficult part of the algorithm.

19.16 By considering the line from (0, 0) to (4, 2) and the line from (4, 2) to (0, 0), show that the midpoint algorithm described in Chapter 3 can draw different pixels, depending on the direction in which the line is specified. Find a way to examine the coordinates of the endpoints of a line and to adjust the comparison (from less than to less than or equal to) in the midpoint algorithm so that the same pixels are always drawn, regardless of the order in which the endpoints of a segment are specified.

19.17 Create a line-drawing algorithm to draw the portion of a line (specified by two endpoints) that lies between two vertical lines. That is, your procedure should look like TrimmedLine (point *start*, point *end*, **int** *xmin*, **int** *xmax*); and should draw the portion of the line segment between *start* and *end* that lies between *xmin* and *xmax*. Implement this algorithm with scissoring, and then with analytic clipping. When doing analytic clipping, be sure to derive the decision variable and increments from the original line, not from the rounded clipped endpoints.

19.18 Explain why a tightly bent curve is difficult to draw with forward-differencing techniques. Explain the problem as a form of aliasing.

19.19 Trace the recursive fill algorithm through a complete example to be sure you understand it. What are the subtle cases in the algorithm?

19.20 Implement the recursive fill algorithm. You can implement it and then execute it on a conventional video terminal, as long as it is cursor-addressable, by using characters to represent different pixel values. Implement both boundary fill and flood fill.

19.21 Show how to construct a mitered joint between two lines using the shape algebra; that is, describe a union, intersection, or difference of shapes that will produce a mitered joint. Do the same for a trimmed mitered joint.

19.22 Develop algorithms for determining the difference between two shapes and the union of two shapes by describing the high-level algorithm in terms of an '''Overlap'' function, and then describing the values returned by the overlap function.

19.23 Implement the improvements to the filling algorithms described in the text. What happens if stack processing is done on a first-in-first-out basis, instead of on a last-in-first-out one? Can you think of frame-buffer architectures in which the first approach is better or worse?

19.24 Consider the two filled circles defined by the equations $(x - 100)^2 + y^2 = 10000.9801$ and $(x + 100)^2 + y^2 = 10000.9801$. Show that, if we scan convert these circles, each generates a pixel at (0, 2); hence, the pixel (0, 2) is contained in the shape data structure for each. Show, on the other hand, that the intersection of these two circles contains only points between $y = -1$ and $y = 1$, and hence should not contain the pixel (0, 2). Is the shape algebra a good way to implement intersections despite this apparent contradiction? Explain the circumstances in which you would accept the shape intersection as a reasonable approximation of the true intersection of geometric primitives.

19.25 In Section 19.1, we talked about determining the side of a ray on which a given point lies. Suppose we have a ray from P to Q, and a point, R.

a. Show that a ray 90° clockwise from the ray from P to Q is given by $(-Q_y + P_y, Q_x - P_x)$.

b. Explain why R is to the left of the ray from P to Q only if the vector from P to R lies in the same halfplane as does the vector computed in part a.

c. Two vectors lie in the same halfplane only if their dot product is positive. Use this fact to show that R lies to the left of the ray from P to Q if and only if

$$(R_x - P_x)(-Q_y + P_y) + (R_y - P_y)(Q_x - P_x)$$

is positive.

19.26 Implement the simpler antialiasing algorithm for the circle described in Section 19.3.2. You will need some preliminary computations.

a. Show that, for large enough R and small values of s, a point (x, y) at a distance s outside a circle of radius R has a residual value, $F(x, y) = x^2 + y^2 - R^2$, which is approximately $2Rs$. (Hint: To say that the point is distance s outside the circle is the same as saying its distance to the origin is $R + s$.)

b. Compute a table of weighted overlap values for a disk of radius 1 and a halfplane; make the table have 32 entries, corresponding to distances between -1 and 1 pixel (a distance of -1 means the pixel center is one unit *inside* the halfplane).

c. Now write a scan-conversion algorithm as follows. For each pixel near the circle of radius R (determine these pixels using a midpoint-style algorithm), compute the residual and divide by $2R$ to get the distance. Use this to index into the table of overlap values; use the distance minus 1 to index into the table as well, and subtract to get a coverage value.

d. Try your algorithm on both large and small circles, and criticize the performance for small circles.

19.27 Suppose you are given a circle in the plane, with an equation of the form $(x - h)^2 + (y - k)^2 - R^2 = 0$. Describe how to detect whether a point (x, y) lies within the circle. Describe how to determine if a point lies within a distance of one from the circle. Now do the same two problems for an ellipse given by $Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F = 0$. The second problem is much harder in this case, because of the difficult formula for the distance. Suppose that instead you estimate the distance from a point (x, y) to the ellipse as follows: multiply the residual at the point (x, y) by $(AC - B^2) / F^2$, and check whether this value is between 0.99 and 1.01. Explain why this might make a reasonable measure of closeness to the ellipse.

19.28 a. Consider the set of points on the x-axis between $-\frac{1}{2}$ and $\frac{1}{2}$, together with the set of points on the y axis between $-\frac{1}{2}$ and $\frac{1}{2}$. We include only one of the two endpoints in each of these intervals. Since this has the shape of a "+" sign, we will call the shape P. Show that the midpoint algorithms we have discussed will draw the pixel at (0, 0) precisely if the primitive intersects P. (Note that P contains only two of its four possible endpoints. The decision of which two to include amounts to the decision to count an exact midpoint crossing of a horizontal line as contributing the left or the right pixel, and similarly for vertical midpoint crossings. Be sure that your choice of endpoints for P corresponds to the algorithm you are studying.)

b. Analogously, a pixel Q is drawn by a midpoint algorithm precisely if the primitive intersects a copy of P that has been translated to have its center at Q. Show that a circle of radius 0.49, centered at (0.5, 0.5), therefore causes no pixel to be drawn.

c. Alternative shapes have been proposed instead of P, including the convex hull of P, and a square box around a pixel. Criticize each of these choices.

19.29 The Posch-Fellner style algorithm for drawing thick curves can be improved somewhat. Instead of considering a circle swept along a curve, consider a *pen polygon* [HOBB89]. A pen polygon is characterized by two properties: opposite sides are parallel, and if the polygon is translated so that one vertex of an edge lies at the origin, then the line containing the opposite edge must pass through a point with integer coordinates. For example, an octagon with vertices $\pm(1, \frac{1}{2})$, $\pm(\frac{1}{2}, 1)$, $\pm(-\frac{1}{2}, 1)$, and $\pm(-1, \frac{1}{2})$ is a pen polygon. Compute the points on a curve through the origin with slope 0.935 using the Posch–Fellner algorithm (with a width of 2) and the pen-polygon algorithm, using the octagonal pen. In general, thick lines drawn with pen polygons have a more uniform appearance.

20 Advanced Modeling Techniques

Earlier chapters have concentrated on geometric models, including transforming and rendering them. In a world made entirely of simple geometric objects, these models would suffice. But many natural phenomena are not efficiently represented by geometric models, at least not on a large scale. Fog, for example, is made up of tiny drops of water, but using a model in which each drop must be individually placed is out of the question. Furthermore, this water-drop model does not accurately represent our perception of fog: We see fog as a blur in the air in front of us, not as millions of drops. Our visual perception of fog is based on how fog alters the light reaching our eyes, not on the shape or placement of the individual drops. Thus, to model the perceptual effect of fog efficiently, we need a different model. In the same way, the shape of a leaf of a tree may be modeled with polygons and its stem may be modeled with a spline tube, but to place explicitly every limb, branch, twig, and leaf of a tree would be impossibly time consuming and cumbersome.

It is not only natural phenomena that resist geometric modeling. Giving an explicit description of the Brooklyn Bridge is also challenging. Much of the detail in the bridge is in rivets, nuts, and bolts. These are not placed in the same location on every strut or cable, so primitive instancing cannot be used; rather, they are placed in ways that can be determined from the locations of the struts or cables (for example, two cables whose ends abut need a coupler between them).

The advanced modeling techniques in this chapter attempt to go beyond geometric modeling, to allow simple modeling of complex phenomena. Typically, this means representing a large class of objects by a single model with easily adjusted and intuitive parameters. Thus, the list of parameters becomes the data from which the model is generated. This technique has been called *database amplification* [SMIT84], a term accurately describing our desire to model elaborate entities that are quite uniform at high levels of detail (e.g., fog, bridges, fire, plants).

Some of the techniques lie somewhere between the extremes of explicit modeling and database amplification, such as the hierarchical splines of [FORS88], and the blobby objects of [BLIN82b] and soft objects of [WYVI86]. *Hierarchical splines* are surface patches having varying densities in their control-point meshes. In regions where the object to be modeled has few features, a coarse mesh is used. In regions with lots of detail, finer meshes subdivide a single rectangle in the coarser mesh. Thus, a modest amount of information (the control points and subdivision hierarchy) describes the shape of the full surface. The *blobby* and *soft objects* are again controlled by locating a few objects and then "blending" them together to form a complex object. The few objects (spheres or blobs of soft material) still required must be placed, but their intersections are smoothed out automatically, freeing the user from having to define fillets explicitly at each object intersection.

Some of the techniques presented here are widely applicable; procedural models, fractals, grammar-based models, and particle systems have all been used to model a wide variety of things. Some of them are special purpose; for example, the ocean-wave models. Nonetheless, many of the models presented here give good pictures without being faithful to the underlying science. The clouds modeled as textured quadrics by Gardner [GARD84] are based not on atmospheric science, but rather on appearances; the water modeled by blobby objects has no basis in the physics of surface tension and the dynamics of water molecules. It is important to recognize the difference between these approaches to modeling and modeling based on the underlying science.

For each modeling technique, new methods of rendering may be required. In discussing a new modeling technique, we therefore introduce any new rendering techniques developed for its use. In one case, volume rendering, the modeling (which consists of the assignment of a value to each point in a 3D region) is not new at all. Scalar fields have been used in physics and mathematics for hundred of years, but only recently have attempts been made to render these fields on 2D images.

Finally, many of the methods presented here have been developed for dynamic models rather than for static ones, and for modeling growth and change as well as form. The individual images produced from these models are of interest, but the full power of the modeling technique comes out when several still images are combined in an animated sequence. Some of the topics covered in this chapter thus concern both *object modeling* and *animation*.

20.1 EXTENSIONS OF PREVIOUS TECHNIQUES

Before starting our survey of new techniques for modeling natural and artificial objects, we discuss two extensions to our previous modeling techniques: hierarchical splines and noise-based pattern mapping. Neither of these handles any shapes or characteristics that we could not model before, but each makes the modeling a great deal simpler. Hierarchical splines make it unnecessary to place many control points in regions with little detail (as would have to be done if a uniformly fine control mesh were used), and noise-based patterns are simply particularly interesting patterns to map onto conventional objects.

20.1.1 Advanced Modeling with Splines

With the tensor-product spline-patch surfaces defined in Chapter 11, more control vertices must be added to gain a higher level of detail. The Oslo algorithm and its descendents [COHE80; BART87] can be applied to the control mesh of such splines to produce new control meshes with more vertices but identical resulting surfaces. This refinement of the control mesh is shown for a line spline in Fig. 20.1. The circled black dots control the shape of the thickened segment in the figure; if one of these is moved, the shape of the thickened segment changes. But how many control vertices do we need to redraw the arc in its changed form? For the portions outside the thickened segment, we can use the (comparatively few) white vertices; for the portion inside the thickened segment, we can use the circled black vertices. This localization of detail is the fundamental notion of hierarchical B-spline modeling developed by Forsey and Bartels [FORS88].

Two problems arise: maintaining a data structure for the hierarchical spline, and altering the large-scale spline without damaging the small-scale one. These two problems can be solved together. We wish to alter the large-scale spline so that the small-scale spline follows the alteration. We do this alteration by describing the locations of the (adjustable) control vertices for the small-scale spline in a coordinate system based on the larger spline. This prescribes the data structure for the splines as well—a tree in which the control vertices of each spline are specified in coordinates based on its parent node in the tree. The initial position of each control vertex defines the origin of its coordinate system, and the displacements along the normal to the large spline and in the directions tangent to the coordinate curves of the large spline determine a basis for this coordinate system. Thus, when the large spline is altered, the origin and basis vectors for the displacement coordinate system are moved as well.

Figure 20.2 shows how this procedure works for a line spline. Color Plate IV.4(a) is an example of the impressive results this technique can yield. Notice that the final object is just a union of portions of various spline patches, so the conventional rendering techniques that can handle splines (including polygonal and ray-tracing renderers) can be adapted to render these objects. About 500 control nodes, each of which contains a parametric position, a level of overlay (i.e., depth in the heirarchy) and an offset, were used to define the dragon's head. By defining the offsets of control vertices relative to particular segments in a skeletal

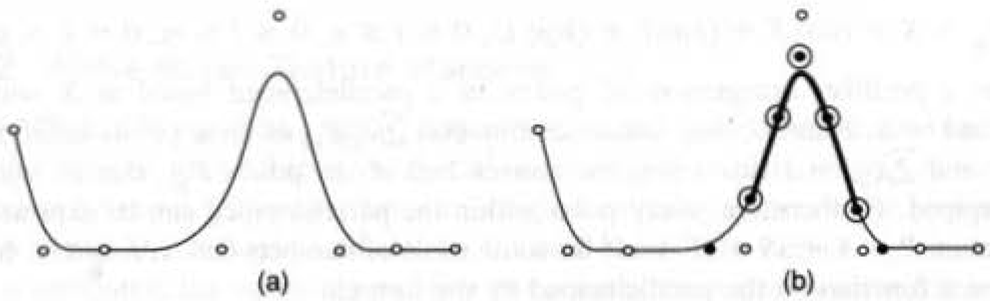


Fig. 20.1 The spline curve in (a) is generated from the control vertices shown there. This collection of control vertices can be refined as shown in (b). The black dots in (b) are the new control vertices; the circled black dots are the ones that contribute to the thickened segment.

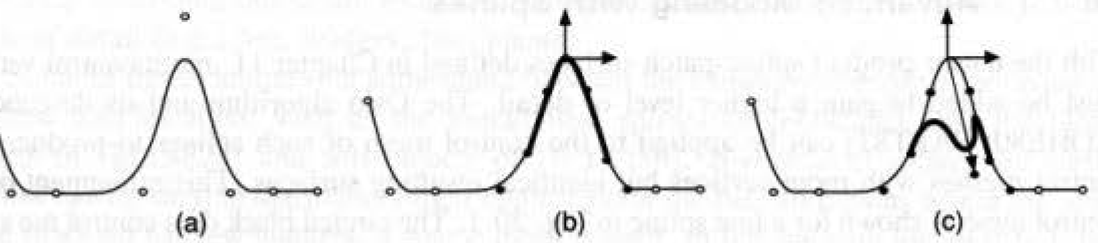


Fig. 20.2 (a) A line spline with its control points. (b) The same spline with subdivided control points in the central segment. Only the middle control point can be moved if continuity is to be maintained. The coordinate system for the displacement of the middle control point is shown as well. (c) The middle control point after it is moved, along with the displacement vector in the new coordinate system.

model (instead of relative to the parent surface), Forsey and Bartels have extended this technique to automate the production of skin, as shown in Color Plate IV.4(b).

Sederberg and Parry describe a different approach to altering spline models [SEDE86] that can be used to alter arbitrary models, although it is based on spline deformations of 3-space. A function from 3-space to 3-space may be thought of as a way to assign new positions to each point in space. Suppose we have such a function and it leaves most points unmoved, but deforms some region of space. (Figure 20.3 shows an analogous deformation in a region of the plane.) If a solid is described by the coordinates of its points in the original space, then applying the function to the coordinates of every point of the solid yields a solid that has been deformed by the function, just as the shaded area was deformed in Fig. 20.3.

Sederberg and Parry use functions from 3-space to 3-space that are based on *Bernstein polynomials*, which have the form[1]

$$Q_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad 0 \le t \le 1.$$

These polynomials have the property that $\sum_{i=0}^{n} Q_{n,i}(t) = 1$ (see Exercise 20.1) and that the individual Qs are always between 0 and 1.

If we establish a coordinate system in 3-space defined by an origin X and three basis vectors S, T, and U, we can form an $(n+1) \times (m+1) \times (p+1)$ lattice of points in 3-space by considering all points

$$P_{ijk} = X + (i/n)\,S + (j/m)T + (k/p)\,U,\ 0 \le i \le n,\ 0 \le j \le m,\ 0 \le k \le p,$$

which is a gridlike arrangement of points in a parallelepiped based at X with sides determined by S, T and U. Any linear combination $\sum c_{ijk} P_{ijk}$ of these points satisfying $0 \le c_{ijk} \le 1$ and $\sum c_{ijk} = 1$ lies within the convex hull of the points P_{ijk}, that is, within the parallelepiped. Furthermore, every point within the parallelepiped can be expressed as a combination $P = X + sS + tT + uU$ for some triple of numbers $0 \le s, t, u \le 1$. Suppose we define a function on the parallelepiped by the formula

[1]The notation $\binom{n}{i}$ means $n!/(i!(n-i)!)$.

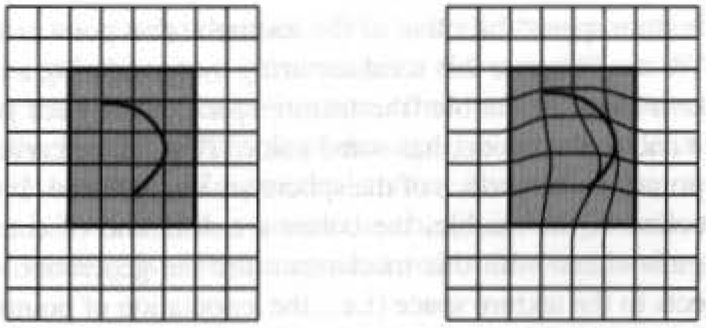


Fig. 20.3 The plane is deformed so that the region inside the rectangle is distorted, but the rest of the plane remains fixed. A figure drawn within the rectangle is also deformed.

$$F(X + sS + tT + uU) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p}\binom{n}{i}\binom{m}{j}\binom{p}{k} t^i (1-t)^{n-i} s^j (1-s)^{m-j} u^k (1-u)^{p-k} P_{ijk}$$
$$= \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p} Q_{n,i}(t)\, Q_{m,j}(s)\, Q_{p,k}(u)\, P_{ijk}.$$

Then because of the convexity property of the Bernstein polynomials, the parallelepiped maps to itself, with the boundary going to the boundary. In fact, if the P_{ijk} are left in the positions defined previously, F sends each point in the parallelepiped to itself. If the P_{ijk}s are moved, the map is no longer the identity. As long as only internal P_{ijk}s are moved, however, the map will remain the identity on the boundary of the parallelepiped. Thus, the P_{ijk}s provide shape control over any item within the box. Color Plate IV.5 shows an example of adjusting a free-form shape; Color Plate IV.6 shows a hammer modeled using free-form deformations.

Computing vertices of a polygonal object after transformation by a trivariate Bernstein polynomial is simple (we just express each vertex as a linear combination $X + sS + tT + uU$ and substitute in the formula for F), so the method is suited for use with all polygonal renderers. It is less clear how to ray-trace these deformed objects, although the specialized methods described for another class of deformations by Barr [BARR84] might be applied.

20.1.2 Noise-Based Texture Mapping

Peachey [PEAC85] and Perlin [PERL85] have extended traditional texture mapping by using *solid textures*. Recall that in traditional bump mapping or pattern mapping the texture was extracted from a 2D image that was mapped onto the surface to be rendered (see Chapters 14 and 16). Described differently, for each point of the surface, a point in the 2D texture is computed, the values surrounding this point are averaged by some filter, and the resulting texture value is assigned to the surface point.

This mechanism is altered slightly for solid textures. A texture value is assigned to each point in a 3D *texture space*. To each point of the object to be textured, there is associated some point in the texture space; the value of the texture at that point is also associated with the surface point. We can illustrate this solid texturing by considering an analogous physical example. If we take a block of marble (the texture space), then each point of the marble, both on the surface and in the inside, has some color. Thus, if we carve a sphere from this marble block, the points on the surface of the sphere are also colored. If we carve the sphere from a different section of the marble, the colors are different, of course.

The two tasks associated with this mechanism are the generation of textures and the mapping from objects to the texture space (i.e., the association of points on the object with points in the texture space). The mapping to texture space is easy in systems where the object is modeled in some space, then is transformed into "world space" before rendering. In this case, the natural choice for texture space is modeling space. During rendering, a 3D point on the object is transformed by the inverse of the modeling transformation to give a point in modeling space whose coordinates provide the index into the solid texture (this situation corresponds to our carving in marble). When this is done, changing the world-space position of the object does not affect its pattern. Using world-space coordinates as indices into the solid texture can provide interesting special effects. If the marble sphere is translated in the course of an animation, the texture slides through it, and it appears to be continuously recarved from new marble. In other systems, some coordinate system (or else some map from world space to texture space) must be chosen to associate each point on the object with a texture value.

Generating textures is a different matter. Before discussing it, let us reconsider the function of texture mapping. When a texture map is used for environment mapping onto a reflective surface, there is no underlying solid texture. The same is true when a texture map is used, say, to put a label on a box, or when bump mapping is applied to an object to simulate architectural details such as regularly spaced ceiling tiles, or to generate surface characteristics such as the directional reflections on brushed metals. But when we simulate the texture of a material such as concrete, wood, or marble, the internal structure of the underlying material determines the resulting appearance of the object. In such cases, solid textures are most applicable.

One type of intermediate case, too, that is handled nicely by solid textures is surface characteristics, such as the texture of stucco, that should be statistically independent of their surface position. Here ordinary pattern mapping tends to produce an orientation because of the coordinate system in which the pattern map is defined, and because of the transformation from the mapping space onto the object, which tends to compress or expand the pattern in some places (e.g., when mapping onto a sphere with standard coordinates, one tends to compress the pattern near the poles). Solid textures handle this problem by associating values that can be made effectively independent of the shape of the surface (see Color Plate IV.7c).

Generating a solid texture requires associating one or more numbers with each point in some volume. We can specify these numbers by generating them at each point of a 3D lattice (this is sometimes called a *3D image*), and then interpolating to give intermediate values, or simply by giving one or more real-valued functions on a region in 3-space.

Most of the functions used by Perlin are based on noise functions. He defines a function $Noise(x, y, z)$ with certain properties: statistical invariance under rigid motions and band limiting in the frequency domain. The first of these means that any statistical property, such as the average value or the variance over a region, is about the same as the value measured over a congruent region in some other location and orientation. The second condition says that the Fourier transform of the signal is zero outside of a narrow range of frequencies (see Section 14.10). In practical terms, this means that the function has no sudden changes, but has no locations where the change is too gradual, either. One way of expressing the band limiting is that, for any unit vector (a, b, c) and any point (x_0, y_0, z_0), the integral

$$\int_0^\infty \text{Noise}(x_0 + ta,\ y_0 + tb,\ z_0 + tc) f(mt)\, dt$$

is zero when $f(t) = \sin(t)$ or $\cos(t)$, and m is outside some small range of values. Essentially, this says that the noise along a parameterized line in the (a, b, c) direction has no periodic character with period m.

Such a noise function can be generated in a number of ways, including direct Fourier synthesis, but Perlin has a quick and easily implemented method. For each point in the integer lattice (i.e., for each point (x_0, y_0, z_0) with x_0 y_0 and z_0 all integers) we compute and store four pseudorandom[2] real numbers (a, b, c, d). Compute $d' = d - (ax_0 + by_0 + cz_0)$. Notice that if we substitute the point (x_0, y_0, z_0) into the formula $ax + by + cz + d'$ we get the value d. We now define the *Noise* function at an arbitrary point (x, y, z) by the two rules: If (x, y, z) is a point of the integer lattice, then $Noise(x, y, z)$ = the d value at that lattice point $= ax_0 + by_0 + cz_0 + d'$. For any point not on the lattice, the values of a, b, c, and d' are interpolated from the values at the nearby lattice points (Perlin recommends a cubic interpolation—first in x, then in y, then in z) to give values for a, b, c, and d' at the point (x, y, z). Now $Noise(x, y, z)$ is computed: $Noise(x, y, z) = ax + by + cz + d'$.

Since the coefficients a, b, c, and d' are interpolated by cubics on the integer lattice, it is clear that there are no discontinuities in their values (in fact, they will all be differentiable functions with well-behaved derivatives). Hence, the value of $Noise(x, y, z)$ is also well behaved and has no high-frequency components (i.e., sudden changes).

Noise functions can be used to generate textures by altering colors, normal vectors, and so on [PERL85]. For example, a random gray-scale value can be assigned to a point by setting its color to $(r, g, b) = Noise(x, y, z) * (1.0, 1.0, 1.0)$ (assuming that the *Noise*() function has been scaled so that its values lie between 0 and 1). A random color can be assigned to a point by $(r, g, b) = (NoiseA(x, y, z), NoiseB(x, y, z), NoiseC(x, y, z))$, where *NoiseA*(), *NoiseB*() and *NoiseC*() are all different instances of the *Noise*() function. An alternate way to assign random colors is to use the gradient of the noise function:

$$Dnoise(x, y, z) = (dNoise/dx,\ dNoise/dy,\ dNoise/dz),$$

which generates a vector of three values at each point. These values can be mapped to color values.

[2]Pseudorandom-number generation is provided by the *Random*() function in many systems. See also [KNUT69].

If an object has sufficiently great extent, it may not be practical to generate a texture for its entire bounding box. Instead, as in Chapter 16, we generate the texture on a finite box (perhaps 256 by 256 by 256) and use the low-order bits of the point's coordinates to index into this array (using modular arithmetic to wrap around from 255 to 0). We can use this finite texture array to generate another type of noise by defining $Noise2(x, y, z) = Noise(2x, 2y, 2z)$. *Noise*2 will have features that are one-half of the size of those generated by *Noise*(). By generating a combination of such multiples of *Noise*(), we can create a number of fascinating textures; see Exercises 20.2 and 20.3. Perlin has extended solid textures to allow the modification of geometry as well [PERL89]. Some examples of the results are shown in Color Plates IV.8 and IV.9.

Peachey [PEAC85] uses somewhat different mechanisms for specifying solid textures. One of the most interesting is what he calls *projection textures*, although the term "extrusion textures" might apply as well. In such textures, the value of the texture function is constant along certain parallel lines in the volume. For example, such a texture might be constant along each line parallel to the z axis, while on any (x, y)-plane cross-section it might look like a conventional 2D texture. The effect is like that of a (nonperspective) slide projector: When someone walks in front of the screen, the image is mapped onto the person instead of onto the screen. These textures are most interesting when several are combined. If the textures are constant along different lines, the results can effectively simulate completely random textures. The textures in Color Plate IV.10 are all based on projection textures.

20.2 PROCEDURAL MODELS

Procedural models describe objects that can interact with external events to modify themselves. Thus, a model of a sphere that generates a polygonal representation of the sphere at a requested fineness of subdivision is procedural: The actual model is determined by the fineness parameter. A model that determines the origin of its coordinate system by requesting information from nearby entities is also procedural. A collection of polygons specified by their vertices is *not* a procedural model.

Procedural models have been in use for a long time. One of their best features is that they save space: It is far easier to say "sphere with 120 polygons" than to list the 120 polygons explicitly. Magnenat-Thalman and Thalman [MAGN85] describe a procedural model for bridges in which a bridge consists of a road, a superstructure, piers, and parapets, and is specified by giving descriptions of these along with an orientation to determine the bridge's position. Each of the pieces (road, piers, etc.) is specified by a number of parameters (length of the road, number of joints in the road, height of the pier, etc.) and the procedure then generates the model from these. This is akin to the primitive instancing of Chapter 12, but differs in that the geometric or topological nature of the object may be influenced by the parameters. Also, the model generated does not need to consist of a collection of solids; it might be a collection of point light sources used to exhibit the bridge in a night scene, for instance. In any case, specifying a few parameters leads to the creation of a very large model. In the case of the bridge, the only things created are various sorts of bridges. In subsequent procedural models, such as particle systems, however, highly variable classes of objects are supported under a single class of procedures.

One important aspect of procedural models is their ability to interact with their environment. Amburn, Grant, and Whitted introduce two extensions to standard procedural models: a communication method through which independent procedures can influence one another's behaviors, and a generalization of the notion of subdivision to include a change of representation [AMBU86].

Interobject communication can be used to control the shapes of objects defined by procedures. Amburn, Grant, and Whitted use as an example a road passing through wooded terrain. The terrain is generated by stochastic subdivision of triangles (see Section 20.3), the trees are generated using grammar-based models (see Section 20.4), and the road is generated by extrusion of a line along a spline path. At the top level, the road must follow the geometry of the terrain. At a finer level of detail, however, the terrain is bulldozed to let the road be smooth. Each of these objects thus must control the other. The bases of the trees must be placed on the terrain, but not too close to the road. To execute this interobject control, each of the subdivision procedures proceeds for a few steps, then checks its progress against that of the others.

This interobject checking can be extremely expensive; the road may be modeled with hundreds of rectangles and the terrain with thousands of triangles. Checking for intersections among these and establishing communications between each pair is prohibitively laborious. Instead, during the construction of the road, bounding boxes for the road, for each pair of control points for the road, and for each segment of the road were constructed. Similar bounding boxes were maintained during the subdivision of the triangles. As soon as the bounding box of a child triangle no longer intersected that of the road, communications between the two were severed. Thus, there were only a few overlaps at the finest level of subdivision.

These subdivisions were also subject to changes of representation. At some point in a subdivision process, the current model representation may no longer seem adequate to the modeler; and the modeler (or some other procedure in the model) may request that some procedural object change its representation. Thus, a shape that is initially modeled with Bezier spline patches, recursively subdivided, may at some point be altered to implement further changes using stochastic subdivision to make a "crinkly" material of some specific overall shape. Amburn, Grant, and Whitted store these changes of representation in a script associated either with the individual object or with the class of the object; the script might say, for example, "At the third level of subdivision, change from Bezier to stochastic. At the fifth level, change to a particle system representation." The human modeler is also allowed to interact with the objects as the procedural modifications take place. Our hope is that, in the future, such interactions will no longer be necessary, and that the models will be able to determine for themselves the best possible representation.

Most of the remaining models in this chapter are procedural in some way. Many of them are generated by repeated subdivision or repeated spawning of smaller objects. The subdivision terminates at a level determined by the modeler, the model, or (depending on implementation) the renderer, which can request that no subpixel artifacts be generated, for example. The power of these models is manifested in how they amplify the modeler's effort: Very small changes in specifications can result in drastic changes of form. (Of course, this can be a drawback in some cases, if the modeler cannot direct a tiny change in the result.)

20.3 FRACTAL MODELS

Fractals have recently attracted much attention [VOSS87; MAND82; PEIT86]. The images resulting from them are spectacular, and many different approaches to generating fractals have been developed. The term *fractal* has been generalized by the computer graphics community to include objects outside Mandelbrot's original definition. It has come to mean anything which has a substantial measure of exact or statistical self-similarity, and that is how we use it here, although its precise mathematical definition requires statistical self-similarity at all resolutions. Thus, only fractals generated by infinitely recursive processes are true fractal objects. On the other hand, those generated by finite processes may exhibit no visible change in detail after some stage, so they are adequate approximations of the ideal. What we mean by *self-similarity* is best illustrated by an example, the von Koch snowflake. Starting with a line segment with a bump on it, as shown in Fig. 20.4, we replace each segment of the line by a figure exactly like the original line. This process is repeated: Each segment in part (b) of the figure is replaced by a shape exactly like the entire figure. (It makes no difference whether the replacement is by the shape shown in part (a) or by the shape shown in part (b); if the one in part (a) is used, the result after 2^n steps is the same as the result after n steps if each segment of the current figure is replaced by the entire current figure at each stage.) If this process is repeated infinitely many times, the result is said to be *self-similar:* The entire object is similar (i.e., can be translated, rotated, and scaled) to a subportion of itself.

An object that is not exactly self-similar may still seem fractal; that is, it may be substantially self-similar. The precise definition of statistical self-similarity is not necessary here—we need only to note that objects that "look like" themselves when scaled down are still called fractal.

Associated with this notion of self-similarity is the notion of *fractal dimension*. To define fractal dimension, we shall examine some properties of objects whose dimension we know. A line segment is 1D; if we divide a line into N equal parts, the parts each look like the original line scaled down by a factor of $N = N^{1/1}$. A square is 2D: if we divide it into N parts, each part looks like the original scaled down by a factor of $\sqrt{N} = N^{1/2}$. (For example, a square divides nicely into nine subsquares; each one looks like the original scaled by a factor of $\frac{1}{3}$.) What about the von Koch snowflake? When it is divided into four pieces (the pieces associated with the original four segments in Fig. 20.4a), each resulting piece looks like the original scaled down by a factor of 3. We would like to say it has a dimension d, where $4^{1/d} = 3$. The value of d must be $\log(4)/\log(3) = 1.26 \ldots$. This is the definition of fractal dimension.

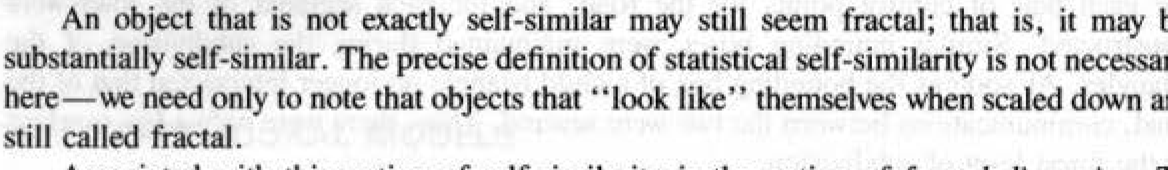


Fig. 20.4 Construction of the von Koch snowflake: each segment in (a) is replaced by an exact copy of the entire figure, shrunk by a factor of 3. The same process is applied to the segments in (b) to generate those in (c).

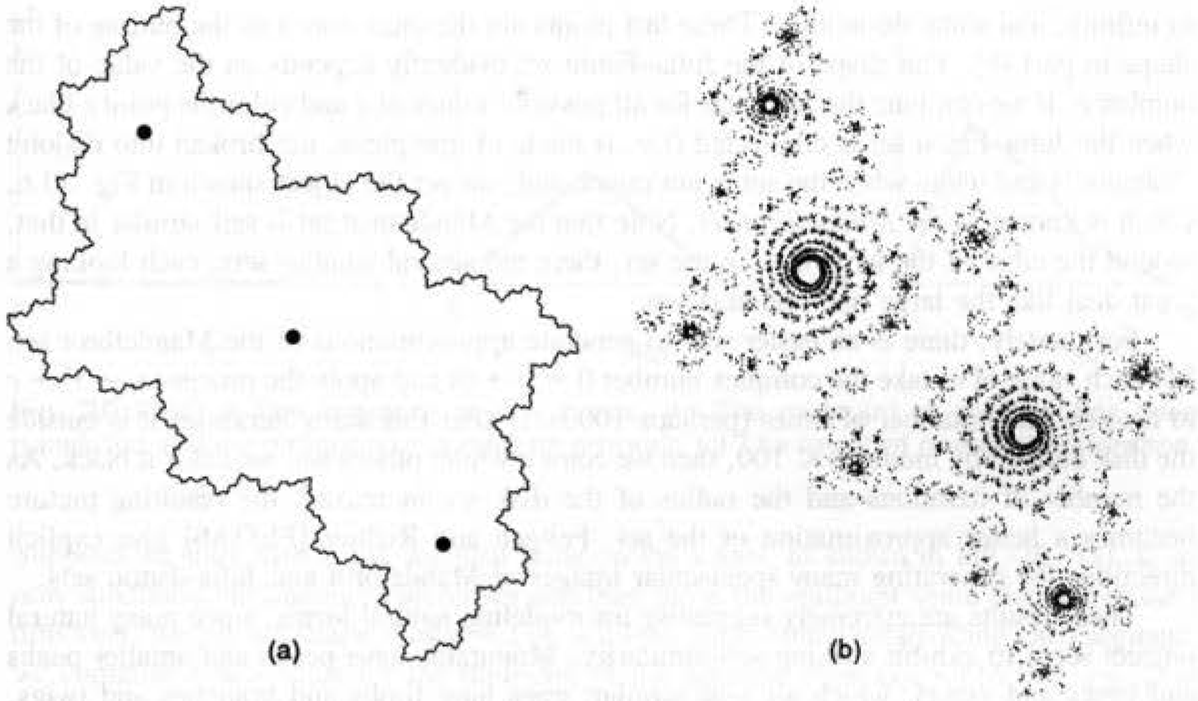


Fig. 20.5 (a) The Julia–Fatou set for $c = -0.12375 + 0.056805i$; (b) the Julia–Fatou set for $c = -0.012 + 0.74i$.

The most famous two fractal objects deserve mention here: the Julia–Fatou set and the Mandelbrot set. These objects are generated from the study of the rule $x \rightarrow x^2 + c$ (and many other rules as well—this is the simplest and best known). Here x is a *complex number*,[3] $x = a + bi$. If a complex number has modulus < 1, then squaring it repeatedly makes it go toward zero. If it has a modulus > 1, repeated squaring makes it grow larger and larger. Numbers with modulus 1 still have modulus 1 after repeated squarings. Thus, some complex numbers "fall toward zero" when they are repeatedly squared, some "fall toward infinity," and some do neither—the last group forms the boundary between the numbers attracted to zero and those attracted to infinity.

Suppose we repeatedly apply the mapping $x \rightarrow x^2 + c$ to each complex number x for some nonzero value of c, such as $c = -0.12375 + 0.056805i$; some complex numbers will be attracted to infinity, some will be attracted to finite numbers, and some will go toward neither. Drawing the set of points that go toward neither, we get the Julia–Fatou set shown in Fig. 20.5(a).

Notice that the region in Fig. 20.5 (b) is not as well connected as is that in part (a) of the figure. In part (b), some points fall toward each of the three black dots shown, some go

[3]If you are unfamiliar with complex numbers, it suffices to treat i as a special symbol and merely to know the definitions of addition and multiplication of complex numbers. If $z = c + di$ is a second complex number, then $x + z$ is defined to be $(a + c) + (b + d)i$, and xz is defined to be $(ac - bd) + (ad + bc)i$. We can represent complex numbers as points in the plane by identifying the point (a, b) with the complex number $(a + bi)$. The *modulus* of the number $a + bi$ is the real number $(a^2 + b^2)^{1/2}$, which gives a measure of the "size" of the complex number.

to infinity, and some do neither. These last points are the ones drawn as the outline of the shape in part (b). The shape of the Julia–Fatou set evidently depends on the value of the number c. If we compute the Julia sets for all possible values of c and color the point c black when the Julia–Fatou set is connected (i.e, is made of one piece, not broken into disjoint "islands") and white when the set is not connected, we get the object shown in Fig. 20.6, which is known as the *Mandelbrot set*. Note that the Mandelbrot set is self-similar in that, around the edge of the large disk in the set, there are several smaller sets, each looking a great deal like the large one scaled down.

Fortunately, there is an easier way to generate approximations of the Mandelbrot set: For each value of c, take the complex number $0 = 0 + 0i$ and apply the process $x \rightarrow x^2 + c$ to it some finite number of times (perhaps 1000). If after this many iterations it is outside the disk defined by modulus < 100, then we color c white; otherwise, we color it black. As the number of iterations and the radius of the disk are increased, the resulting picture becomes a better approximation of the set. Peitgen and Richter [PEIT86] give explicit directions for generating many spectacular images of Mandelbrot and Julia–Fatou sets.

These results are extremely suggestive for modeling natural forms, since many natural objects seem to exhibit striking self-similarity. Mountains have peaks and smaller peaks and rocks and gravel, which all look similar; trees have limbs and branches and twigs, which all look similar; coastlines have bays and inlets and estuaries and rivulets and drainage ditches, which all look similar. Hence, modeling self-similarity at some scale seems to be a way to generate appealing-looking models of natural phenomena. The scale at which the self-similarity breaks down is not particularly important here, since the intent is modeling rather than mathematics. Thus, when an object has been generated recursively through enough steps that all further changes happen at well below pixel resolution, there is no need to continue.

Fournier, Fussell, and Carpenter [FOUR82] developed a mechanism for generating a class of fractal mountains based on recursive subdivision. It is easiest to explain in 1D.

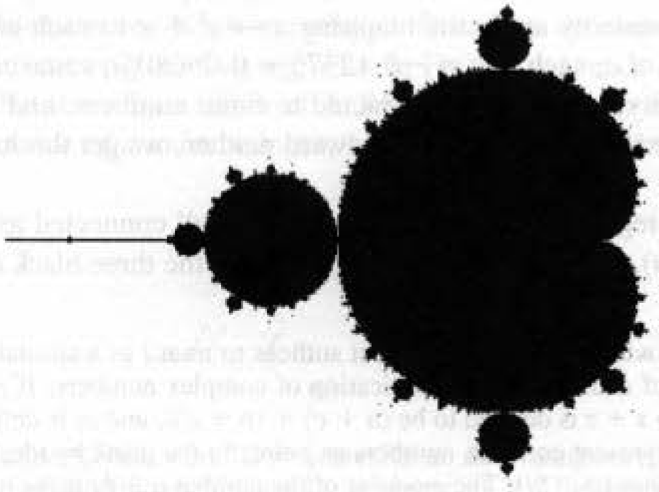


Fig. 20.6 The Mandelbrot set. Each point c in the complex plane is colored black if the Julia set for the process $x \rightarrow x^2 + c$ is connected.

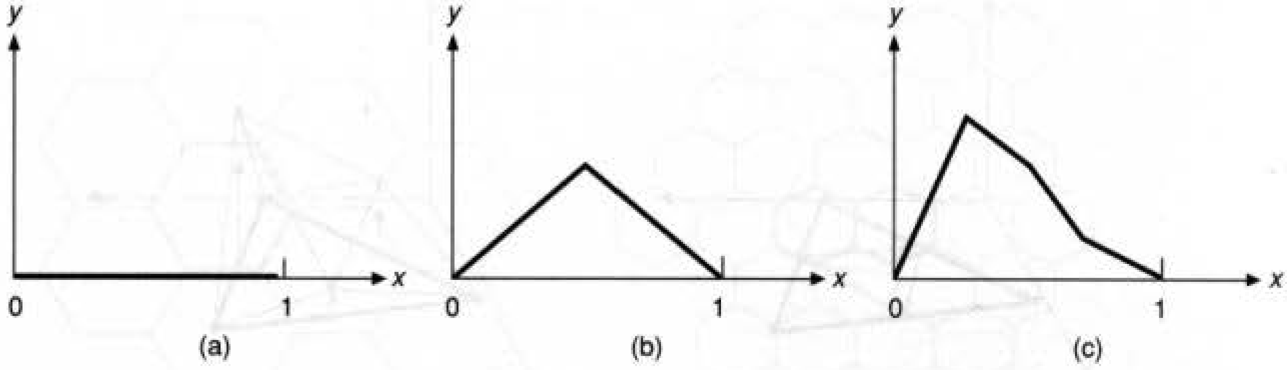


Fig. 20.7 (a) A line segment on the x axis. (b) The midpoint of the line has been translated in the y direction by a random amount. (c) The result of one further iteration.

Suppose we start with a line segment lying on the x axis, as shown in Fig. 20.7(a). If we now subdivide the line into two halves and then move the midpoint some distance in the y direction, we get the shape shown in Fig. 20.7(b). To continue subdividing each segment, we compute a new value for the midpoint of the segment from (x_i, y_i) to (x_{i+1}, y_{i+1}) as follows: $x_{\text{new}} = \frac{1}{2}(x_i + x_{i+1})$, $y_{\text{new}} = \frac{1}{2}(y_i + y_{i+1}) + P(x_{i+1} - x_i)\ R(x_{\text{new}})$, where $P()$ is a function determining the extent of the perturbation in terms of the size of the line being perturbed, and $R()$ is a random number[4] between 0 and 1 selected on the basis of x_{new} (see Fig. 20.7c). If $P(s) = s$, then the first point cannot be displaced by more than 1, each of the next two points (which are at most at height $\frac{1}{2}$ already) cannot be displaced by more than $\frac{1}{2}$, and so on. Hence, all the resulting points fit in the unit square. For $P(s) = s^a$, the shape of the result depends on the value of a; smaller values of a yield larger perturbations, and vice versa. Of course, other functions, such as $P(s) = 2^{-s}$, can be used as well.

Fournier, Fussell, and Carpenter use this process to modify 2D shapes in the following fashion. They start with a triangle, mark the midpoint of each edge, and connect the three midpoints, as shown in Fig. 20.8 (a). The y coordinate of each midpoint is then modified in the manner we have described, so that the resulting set of four triangles looks like Fig. 20.8 (b). This process, when iterated, produces quite realistic-looking mountains, as shown in Color Plate IV.11 (although, in an overhead view, one perceives a very regular polygonal structure).

Notice that we can start with an arrangement of triangles that have a certain shape, then apply this process to generate the finer detail. This ability is particularly important in some modeling applications, in which the layout of objects in a scene may be stochastic at a low level but ordered at a high level: The foliage in an ornamental garden may be generated by a stochastic mechanism, but its arrangement in the garden must follow strict rules. On the other hand, the fact that the high-level structure of the initial triangle arrangement persists in the iterated subdivisions may be inappropriate in some applications (in particular, the fractal so generated does not have all the statistical self-similarities present in fractals based

[4]$R()$ is actually a *random variable*, a function taking real numbers and producing randomly distributed numbers between 0 and 1. If this is implemented by a pseudorandom-number generator, it has the advantage that the fractals are repeatable: We can generate them again by supplying the same seed to the pseudorandom-number generator.

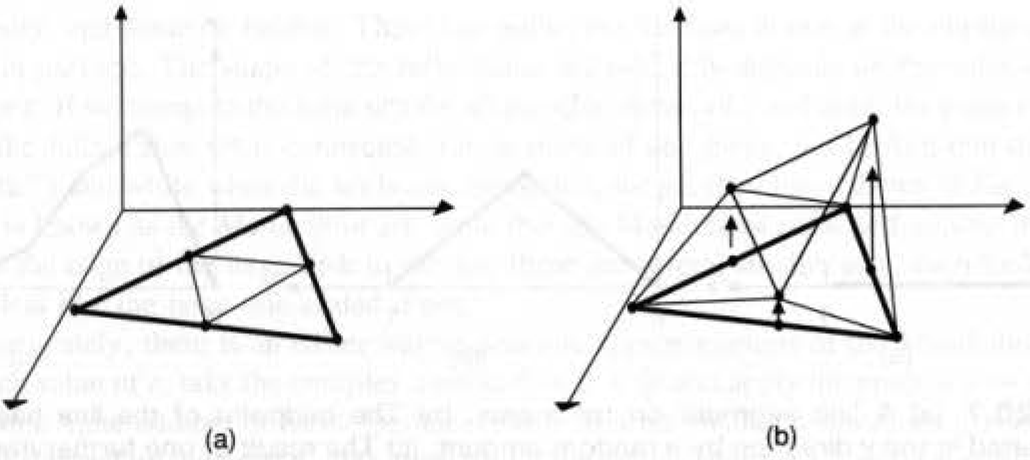


Fig. 20.8 (a) The subdivision of a triangle into four smaller triangles. The midpoints of the original triangle are perturbed in the *y* direction to yield the shape in (b).

on Brownian motion [MAND82]). Also, since the position of any vertex is adjusted only once and is stationary thereafter, creases tend to develop in the surface along the edges between the original triangles, and these may appear unnatural.

Voss [VOSS85] describes a modified version of this algorithm in which stage $n + 1$ of a model is created by adding a random displacement to every vertex of the model at stage n, together with the midpoints of the edges at that stage. This method removes many of the artifacts of the original subdivision algorithm but lacks the control provided by that algorithm. Voss also discusses methods that produce models with even greater statistical invariance under scaling and have other mathematical properties more consistent with the original definition of fractals [VOSS85]. In particular, the Weierstrass–Mandelbrot random fractal function gives a computationally tractable mechanism for generating fractal functions of one variable, and can doubtless be extended to two or more.

Mandlebrot has developed another improvement of the midpoint-displacement algorithm [PEIT88]. His first observation is that the displacements in the original midpoint-displacement algorithm are symmetric, so when a fractal mountain of this sort is inverted, it has the same statistical properties as when upright. Real mountains look very different from inverted valleys, and Mandlebrot models this asymmetry by choosing the displacements from a nonsymmetric distribution, such as a binomial distribution. He also relieves some of the "creasing" of the midpoint model by choosing a different subdivision method. Rather than starting with an initial mesh of triangles, he starts from an initial mesh of hexagons. Noting that height values need to be associated with only the vertices in a mesh, he changes the topology of the mesh during subdivisions so that the initial edges of the hexagon are no longer edges in the subdivision. Instead, he replaces the hexagon with three smaller hexagons, as shown in Fig. 20.9. The central vertex has its height computed as in the triangle algorithm—as an average of the neighboring vertices in the original hexagon, plus a displacement. The other six new vertices are given heights that are weighted averages of the vertices of the hexagon. Mandlebrot says that different choices of weights give substantially different results. The principle feature of this subdivision is that the edges of

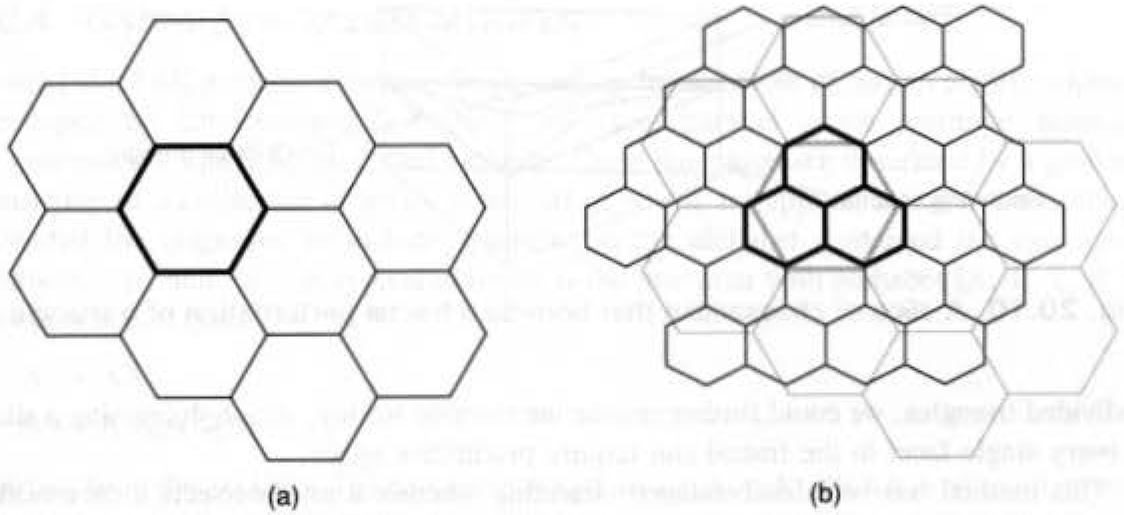


Fig. 20.9 (a) The initial hexagon mesh. One hexagon has been drawn in heavier lines. (b) The subdivided hexagon mesh, with the descendants of the outlined hexagon drawn in heavy lines.

the original hexagons, along which creases might have formed, are now distorted into multiple edges, so that the creases will be far less apparent. The fractals so generated are extremely impressive.

For further fractal algorithms, see [VOSS85; PEIT86].

Other iterative processes can be used to generate a great many interesting images. The grammar-based models and particle systems described in the following sections give some sense of the power of this approach. The changes in those models at deep levels of recursion illustrate a deficiency of the self-similarity model for natural objects. The structure of a tree may be self-similar at certain levels—branches and twigs look a lot alike—but the leaves of a tree do not really look much like a tree.

Rendering fractals can be difficult. If the fractals are rendered into a *z*-buffer, displaying the entire object takes a long time because of the huge number of polygons involved. In scan-line rendering, it is expensive to sort all the polygons so that only those intersecting the scan line are considered. But ray tracing fractals is extremely difficult, since each ray must be checked for intersection with each of the possibly millions of polygons involved. Kajiya [KAJI83] gave a method for ray tracing fractal objects of the class described in [FOUR82], and Bouville [BOUV85] improves this algorithm by finding a better bounding volume for the objects.

Kajiya points out that, if one starts with a triangle and displaces points within it in the vertical direction, as described in [FOUR82], the resulting object lies within a triangular prism of infinite extent, whose cross-section is the original triangle. If the displacements of the points of the triangle are small enough, then their sum remains finite, and the shape based at the triangle is contained in a truncated triangular prism (''slice of cheesecake''; see Fig. 20.10). We could thus ray trace a fractal mountain by first checking whether a ray hits a cheesecake slice for each of the original triangles; if not, no further checking of that triangle's descendants is necessary. By creating additional slices of cheesecake for further

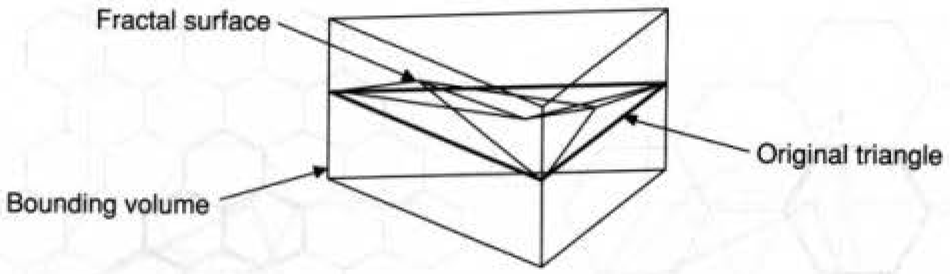


Fig. 20.10 A slice of cheesecake that bounds a fractal perturbation of a triangle.

subdivided triangles, we could further reduce intersection testing, although creating a slice for every single facet in the fractal can require prohibitive space.

This method has two disadvantages: detecting whether a ray intersects a cheesecake slice requires computing intersections with several planes (i.e., solving several algebraic equations), and the slice of cheesecake is not a tight bounding volume—lots of rays hit the cheesecake but never hit the fractal. Bouville observed that, when a triangle is subdivided and interior vertices are displaced, the original vertices remain fixed [BOUV85]. He therefore proposed fitting an ellipsoid around the subdivided triangle so that the original three vertices lay on an equator of the ellipsoid, and the displaced internal vertices all lay within the ellipsoid. In fact, as long as the displaced internal vertices lie within the ellipsoid with high probability, the results are attractive (determining this probability requires artistic rather than scientific judgment). If the ellipsoid is made so large as to be certain to contain all possible displaced vertices, it may be a bad bounding region, in the sense that many rays hit the ellipsoid but not the fractal object within. Notice that testing ray–ellipsoid intersection is easy: It amounts to solving one quadratic equation. This makes the Bouville method far faster than is the slice-of-cheescake method. Furthermore, the ellipsoids include much less extraneous volume than do the slices of cheesecake, so fewer levels of recursion are expected.

One other form of fractal modeling deserves mention, and that is the iterated function systems (IFSs) described in Chapter 17. The IFSs described there differ from all the other forms of modeling in this chapter, in that they model the *image* rather than the objects in the image. That is, a specification of a collection of contractive affine maps, associated probabilities, and a coloring algorithm, as described in Section 17.7.2, simply provides a compact description of a pixmap. For example, in the scene shown in Color Plate IV.1, altering a single affine map might distort the image substantially, shearing a limb away from every tree (and a branch away from every limb, and a twig away from every branch). It might also cause a branch to appear where one was not wanted.

IFSs can be used to generate images with great complexity. Since images of this sort are often desirable for pattern mapping, we can expect to see IFSs become a standard part of the modeler's toolkit.

A careful study of IFSs reveals that the technique does not actually depend on the dimension, so IFS models of 3D objects can be made as well. In some sense, the grammar-based models discussed next are quite similar: New parts of a model are generated by transformation of old parts to smaller-sized copies of some or all of the original parts.

20.4 GRAMMAR-BASED MODELS

Smith [SMIT84] presents a method for describing the structure of certain plants, originally developed by Lindenmayer [LIND68], by using parallel graph grammar languages (*L-grammars*), which Smith called *graftals*. These languages are described by a grammar consisting of a collection of productions, all of which are applied at once. Lindenmayer extended the languages to include brackets, so the alphabet contained the two special symbols, "[" and "]." A typical example is the grammar with alphabet {A, B, [,]} and two production rules:

1. A → AA
2. B → A[B]AA[B]

Starting from the axiom A, the first few generations are A, AA, AAAA, and so on; starting from the axiom B, the first few generations are

0. B
1. A[B]AA[B]
2. AA[A[B]AA[B]]AAAA[A[B]AA[B]]

and so on. If we say that a word in the language represents a sequence of segments in a graph structure and that bracketed portions represent portions that branch from the symbol preceding them, then the figures associated with these three levels are as shown in Fig. 20.11.

This set of pictures has a pleasing branching structure, but a somewhat more balanced tree would be appealing. If we add the parentheses symbols, "(" and ")," to the language and alter the second production to be A[B]AA(B), then the second generation becomes

2. AA[A[B]AA(B)]AAAA(A[B]AA(B))

If we say that square brackets denote a left branch and parentheses denote a right branch, then the associated pictures are as shown in Fig. 20.12. By progressing to later generations in such a language, we get graph structures representing extremely complex patterns. These graph structures have a sort of self-similarity, in that the pattern described by the

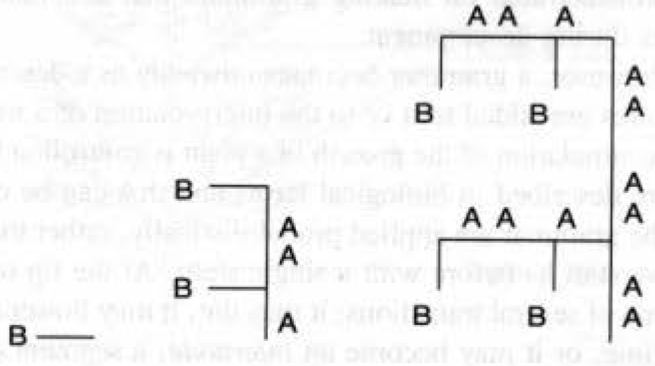


Fig. 20.11 Tree representations of the first three words of the language. All branches are drawn to the left of the current main axis.

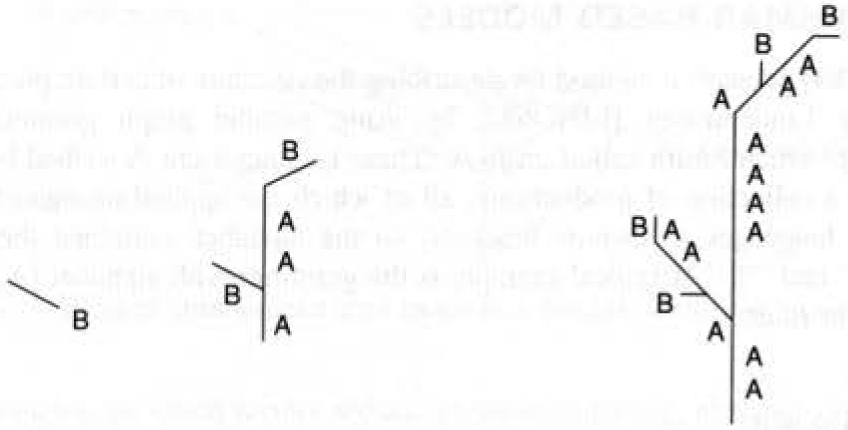


Fig. 20.12 Tree representations of the first three words, but in the language with two-sided branching. We have made each segment of the tree shorter as we progress into further generations.

*n*th-generation word is contained (repeatedly, in this case) in the $(n + 1)$th-generation word.

Generating an object from such a word is a process separate from that of generating the word itself. Here, the segments of the tree have been drawn at successively smaller lengths, the branching angles have all been 45°, and the branches go to the left or to the right. Choosing varying branching angles for different depth branches, and varying thicknesses for the lines (or even cylinders) representing the segments gives different results; drawing a "flower" or "leaf" at each terminal node of the tree further enhances the picture. The grammar itself has no inherent geometric content, so using a grammar-based model requires both a grammar and a geometric interpretation of the language.

This sort of enhancement of the languages and the interpretation of words in the language (i.e., pictures generated from words) has been carried out by several researchers [REFF88; PRUS88]. The grammars have been enriched to allow us to keep track of the "age" of a letter in a word, so that the old and young letters are transformed differently (this recording of ages can be done with rules of the form $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, . . . , $Q \rightarrow QG[Q]$, so that no interesting transitions occur until the plant has "aged"). Much of the work has been concentrated on making grammars that accurately represent the actual biology of plants during development.

At some point, however, a grammar becomes unwieldy as a descriptor for plants: Too many additional features are added to it or to the interpretation of a word in it. In Reffye's model [REFF88], the simulation of the growth of a plant is controlled by a small collection of parameters that are described in biological terms and that can be cast in an algorithm. The productions of the grammar are applied probabilistically, rather than deterministically.

In this model, we start as before with a single stem. At the tip of this stem is a *bud*, which can undergo one of several transitions: it may die, it may flower and die, it may sleep for some period of time, or it may become an *internode*, a segment of the plant between buds. The process of becoming an internode has three stages: the original bud may generate one or more *axillary buds* (buds on one side of the joint between internodes) a process that is called *ramification*; the internode is added; and the end of the new internode becomes

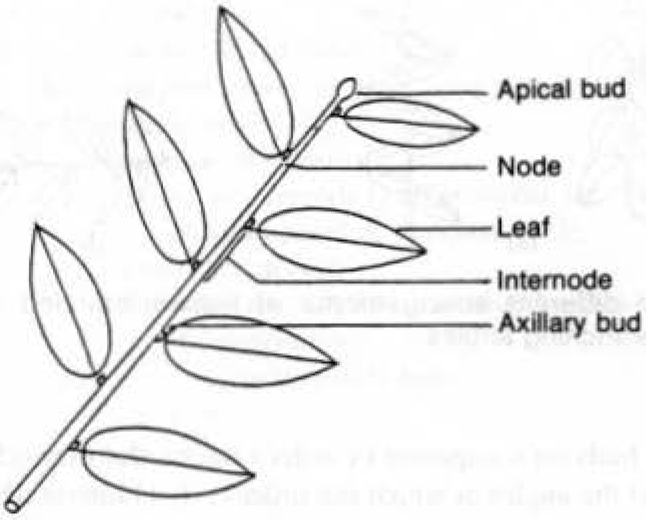


Fig. 20.13 The bud at the tip of a segment of the plant can become an internode; in so doing, it creates a new bud (the *axillary bud*), a new segment (*the internode*), and a new bud at the tip (the *apical bud*).

an *apical bud* (a bud at the very end of a sequence of internodes). Figure 20.13 shows examples of the transition from bud to internode.

Each of the buds in the resulting object can then undergo similar transitions. If we say the initial segment of the tree is of *order 1*, we can define the order of all other internodes inductively: Internodes generated from the apical bud of an order-i internode are also of order-i; those generated from axillary buds of an order-i internode are of order $(i + 1)$. Thus, the entire trunk of a tree is order 1, the limbs are order 2, the branches on those limbs are order 3, and so on. Figure 20.14 shows a more complicated plant and the orders of various internodes in the plant.

The discussion so far describes the topology of the plant, but does not describe the shape at all—whether the branches point up, down, or sideways has not been recorded. The

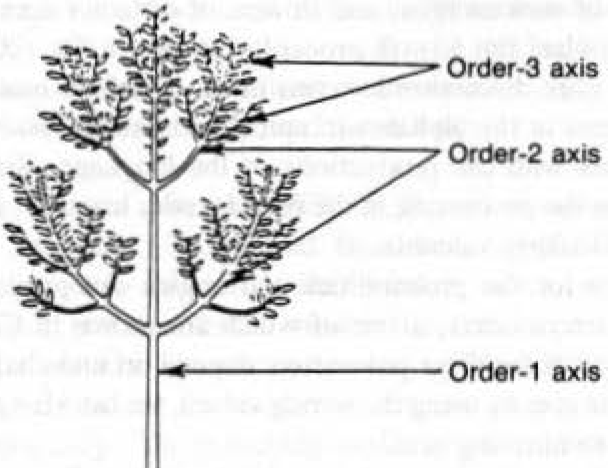


Fig. 20.14 A more complex plant (see Fig. 20.13), with orders attached to the various internodes.

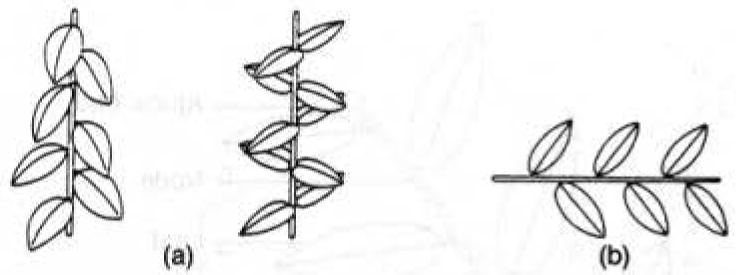


Fig. 20.15 (a) Two different arrangements of leaves: spiraled and distic. (b) The effects of different branching angles.

placement of axillary buds on a sequence of order-i internodes may occur in different ways (see Fig. 20.15a), and the angles at which the order-$(i + 1)$ internodes (if any) branch out from the order-i axillary buds also determine the shape of the plant (see Fig. 20.15b). There are also some anomalies in tree growth, in which the behavior of a collection of order-$(i + 1)$ internodes is not standard, but instead resembles that of some lower order (this is called *reiteration*), and this too must be modeled.

Finally, converting this description into an actual image of a tree requires a model for the shapes of its various components: an order-1 internode may be a large tapered cylinder, and an order-7 internode may be a small green line, for example. The sole requirement is that there must be a leaf at each axillary node (although the leaf may fall at some time).

To simulate the growth of a plant in this model, then, we need the following biological information: the current age of the model, the growth rate of each order of internode, the number of axillary buds at the start of each internode (as a function of the order of the internode), and the probabilities of death, pause, ramification, and reiteration as functions of age, dimension, and order. We also need certain geometric information: the shape of each internode (as a function of order and age), the branching angles for each order and age, and the tropism of each axis (whether each sequence of order-i internodes is a straight line, or curves toward the horizontal or vertical). To draw an image of the plant, we need still more information: the color and texture of each of the entities to be drawn—internodes of various orders, leaves of various ages, and flowers of different ages.

Pseudocode to simulate the growth process is shown in Fig. 20.16.

We can cast this entire discussion in terms of the grammar models that inspired it by assigning different letters in the alphabet to apical and axillary buds of various ages, and associating probabilities with the productions of the language. Since the application of productions amounts to the processing in the pseudocode, however, it is not clear that such a reformulation is particularly valuable.

Varying the values for the probabilities and angles can produce a wide variety of extremely convincing tree models, a few of which are shown in Color Plates IV.12 and IV.13. The correct choices for these parameters depend on knowledge of plant biology or on the modeler's artistic eye; by using the wrong values, we can also generate plants bearing no resemblance at all to anything real.

These plant models are the most spectacular examples of grammar-based modeling. The method has been used in other applications as well, including architecture [STIN78]. In any domain in which the objects being modeled exhibit sufficient regularity, there may be an opportunity to develop a grammar-based model.

```
for (each clock time) {
    for (each bud that is still alive) {
        determine from order, age, etc., what happens to bud;
        if (bud does not die)
            if (bud does not sleep) {
                create an internode (with geometric information
                    about its position, direction, etc.);
                create apical bud;
                for (each possible bud at old bud location)
                    if (ramification)
                        create axillary buds;
            } /* if */
    } /* for */
} /* for */
```

Fig. 20.16 Pseudocode for the plant-growth algorithm. Adapted from [REFF88].

20.5 PARTICLE SYSTEMS

Particle systems are an intriguing approach to modeling objects whose behavior over time cannot easily be described in terms of the surface of the objects (i.e., objects whose topology may change) [REEV83; REEV85]. A *particle system* is defined by a collection of particles that evolves over time. The evolution is determined by applying certain probabilistic rules to the particles: they may generate new particles, they may gain new attributes depending on their age, or they may die (disappear from the object). They also may move according to either deterministic or stochastic laws of motion. Particle systems have been used to model fire, fog, smoke, fireworks, trees, and grass.

Particles have been used for years as elementary entities in graphics modeling, especially in early video games, where they denoted bullets or exploding spaceships. These particles however, were deterministic and had to be placed individually. The *effects* of large collections of particles have also been used before and since to model the transmission and reflection of light in fog and in other diffuse media [BLIN82a; NISH87; RUSH87]. The essence of particle systems is that the positions of the particles are generated automatically, their evolution is controlled automatically, and the individual particles affect the final image directly.

In his first paper on particle systems [REEV83], Reeves describes their use in modeling fire, explosions, and fireworks. Reeves and Blau went on [REEV85] to use them in modeling the grass and trees in a forest. In this context, the particle systems look a great deal like the probabilistic grammar-based models described in the previous section. For example, the trees are modeled as particle systems in which each branch is a particle, each of which is placed randomly along the trunk's length; and each branch may fork or extend according to some probability. The branching angles or the various segments are selected from a distribution, as is the length of the branch (depending on its position in the tree). The particles in this system are like the letters of the alphabet in the grammar-based approach, and the rules for particle birth, death, and transformation correspond to the productions in the grammar.

The modeling of fire in [REEV83] is quite different. Here, the particles have a tree structure (particles have child particles), but the tree structure is not incorporated into the resulting image. Two levels of particle systems were used in modeling the Genesis effect in Color Plate IV.14. The first generated a collection of particles on circles of varying radii centered at a single point on the planet's surface; the particles were distributed about these circles at random positions selected from a probability distribution. Each of these particles was then used as the starting location for a new particle system of a different type (an *explosion* particle system).

In the *Genesis effect*, an explosion particle system is used to model a small burst of sparks from a region on the planet's surface (such systems can also be used to model fireworks and similar phenomena.) The particles of the system are generated in a small disk on the planet's surface with an initial direction of motion that is upward from the surface but may have some horizontal component as well (see Fig. 20.17). The position of each particle at subsequent times is computed by adding its velocity vector to its current position; the velocity vector may be updated by an acceleration vector (which may include gravity) as well. The placement of the particles in the disk, the rate at which they are generated, the initial velocities, and the lifetimes of the particles are all randomly chosen. In each such choice, the value of the property is chosen by a rule of the form

$$\mathit{property} = \mathit{centralValueForProperty} + \mathit{Random}() * \mathit{VarianceOfProperty},$$

so the central value and variance of the property must be specified as well.

The colors of the particles are initially set to red, with some green and a little blue, and alter over time to fade away, with the red component lasting longer than the green or blue, to simulate the cooling of a white-hot material.

Rendering particle systems is a different matter altogether. Ray tracing a particle system would be impossible, since computing the intersection of each ray with even a bounding box for each of several million particles would be immensely time consuming. To render the fire in the Genesis sequence, Reeves simply took each particle as a small point of light and computed the contribution of this light source to the final image. Since the particles were moving, he actually computed a short line segment representing the path of the particle during the frame being rendered, and then rendered this line segment (antialiased) into the final pixmap.[5] Each pixel value was computed by accumulating the values from each particle, so some pixels that were affected by many particles became clamped to the maximum values of red, green, and blue (especially red, since that was the dominant particle color). Particles that were actually behind other particles still contributed to the image, so no occluding of particles was done at all. Two tricks were used for the numerous fires burning on the planet. First, all the particles on the hidden side of the planet were rendered, then the planet was rendered, and then the front particles were rendered. These were composited together in the order back particles–planet–front particles to prevent the particles on the back from showing through the planet's surface (i.e., no z information was stored with the rendered images of the particles). Also, the particle systems contributed light only to the screen image, whereas actual fires would illuminate the nearby portions of

[5]This constitutes *motion blur*, which is discussed in Chapters 14 and 21.

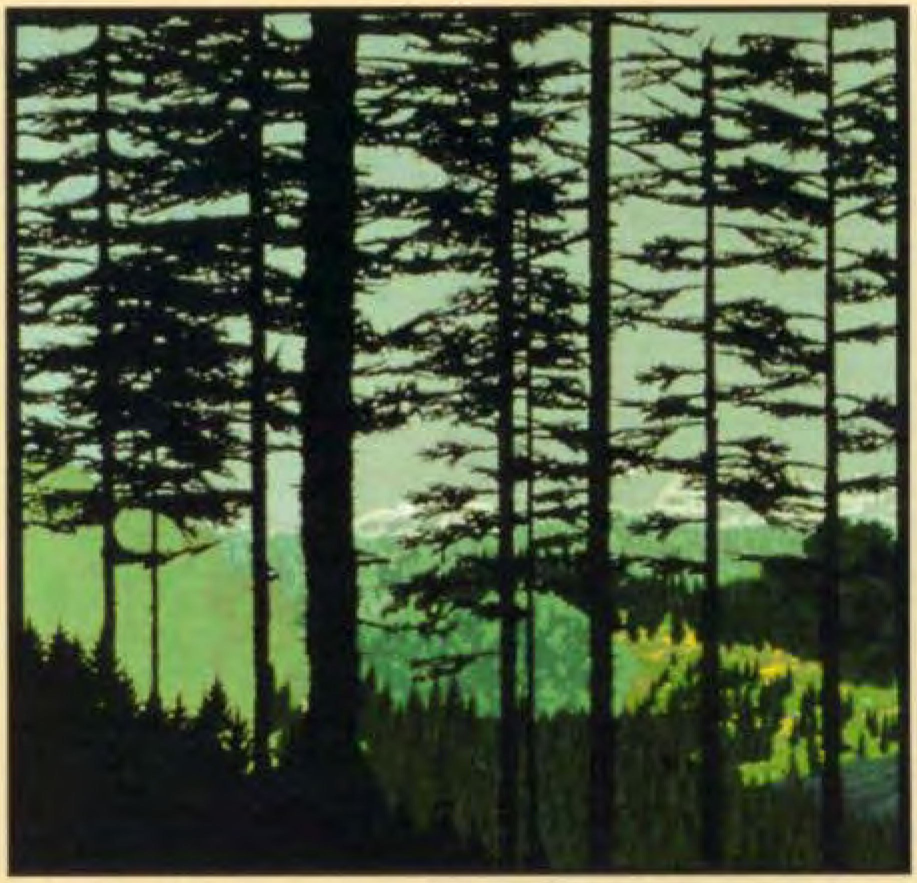
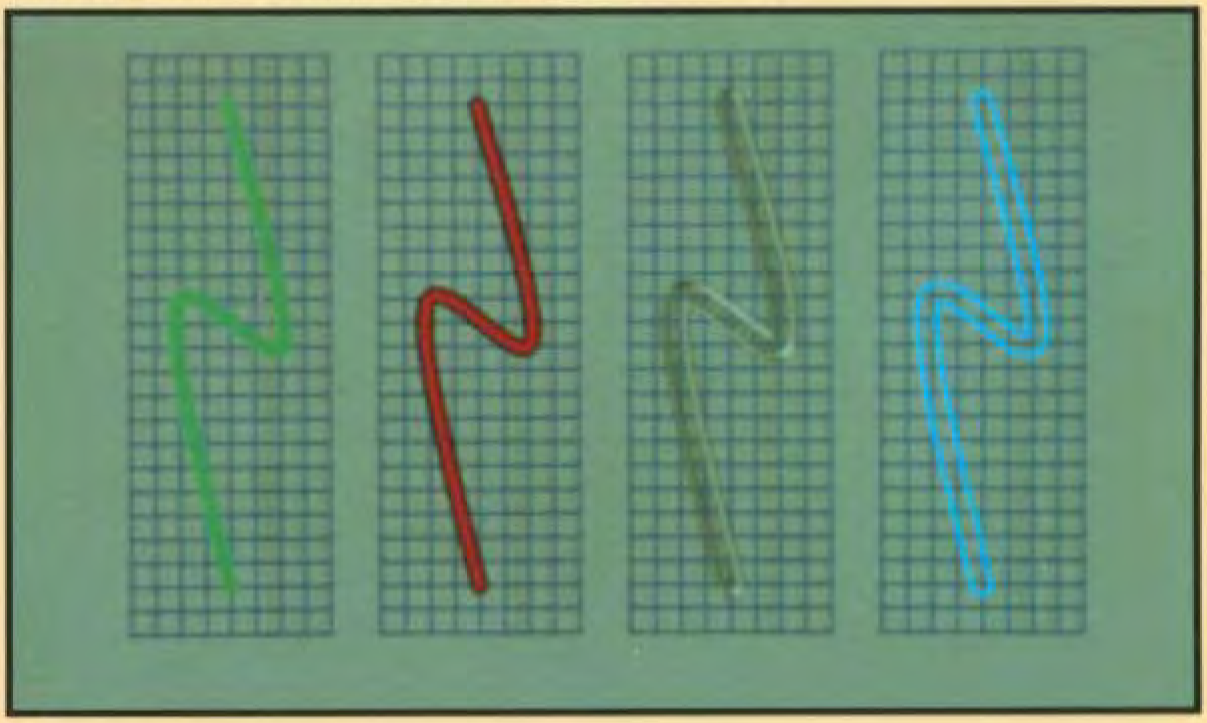


Plate IV.1 An image generated with an iterated function system. The function system contains fewer than 120 affine maps. (Courtesy of Michael Barnsley, Arnaud Jacquin, François Malassenet, Laurie Reuter, and Alan Sloan.)

Plate IV.2 Strokes drawn with an antialiased brush. (Courtesy of Turner Whitted, Bell Laboratories.)



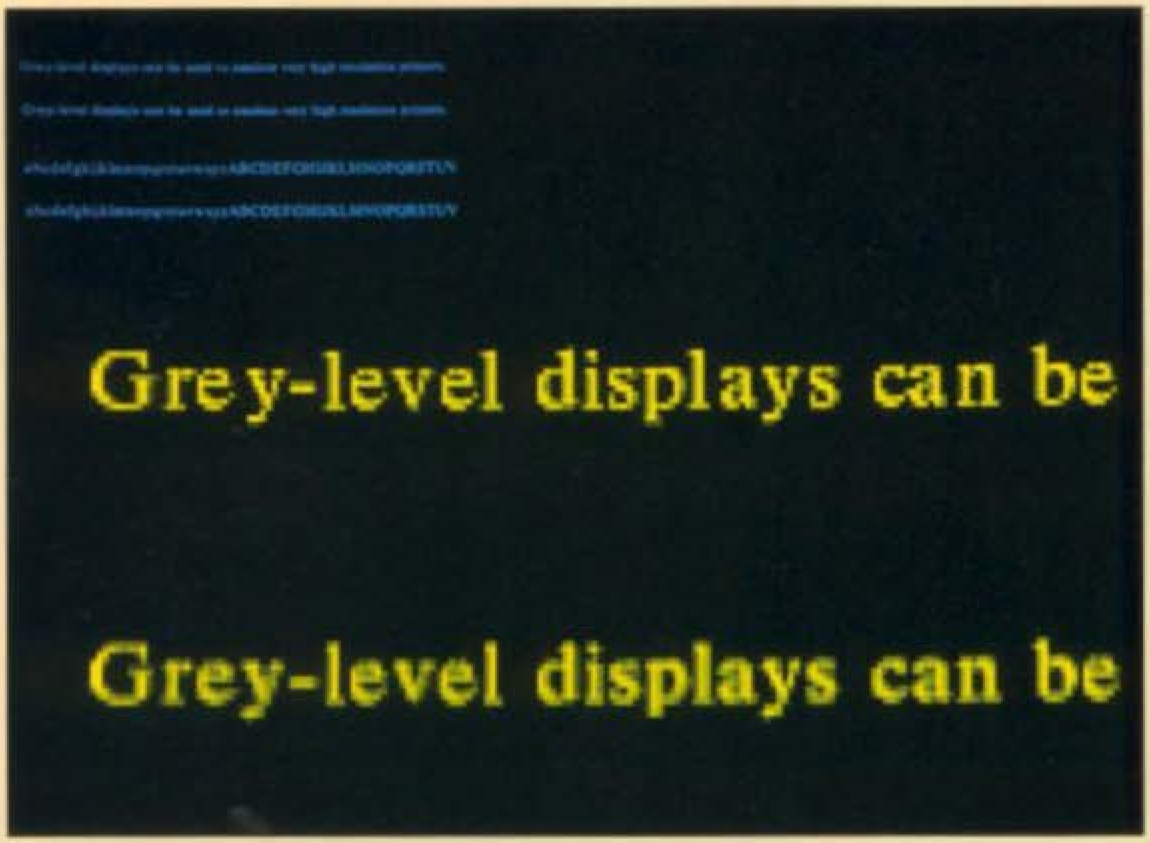


Plate IV.3 Antialiased text as displayed by the YODA display. (Courtesy of Satish Gupta, IBM T. J. Watson Research Center.)

Plate IV.4(a) A dragon modeled with hierarchical splines. (Plates (a) and (b) courtesy of David Forsey, Computer Graphics Laboratory, University of Waterloo.)

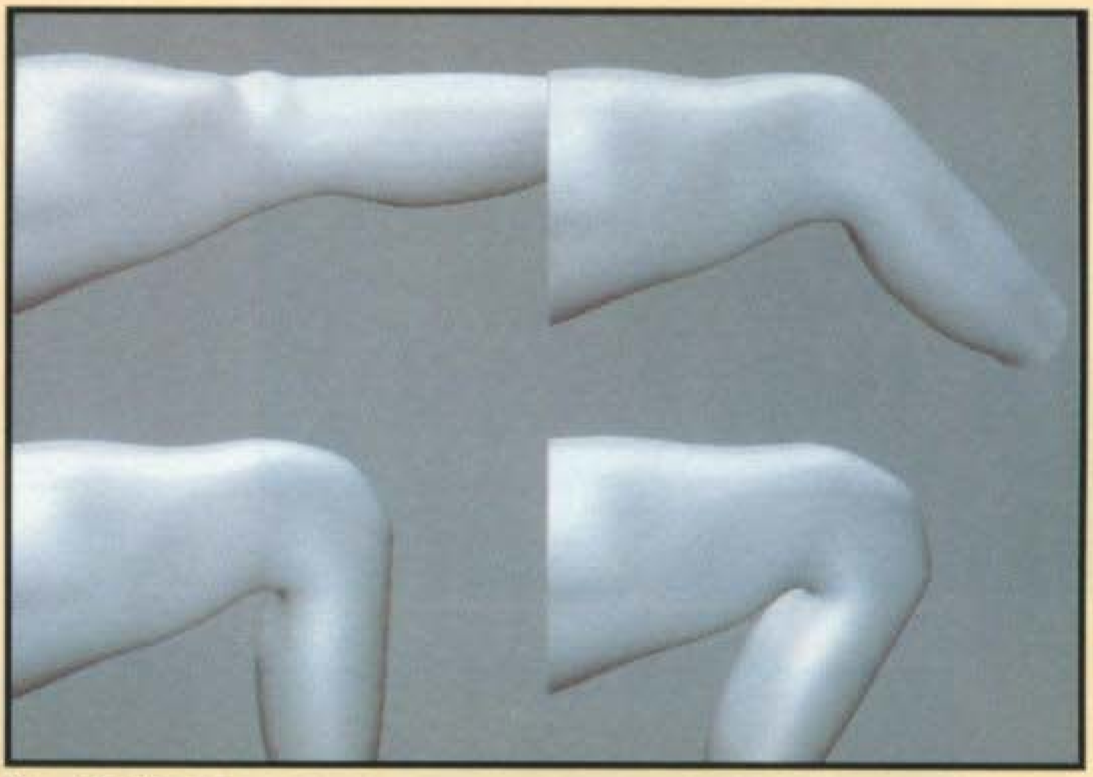


Plate IV.4(b) Skin modeled by defining hierarchical spline offsets relative to a skeletal model.

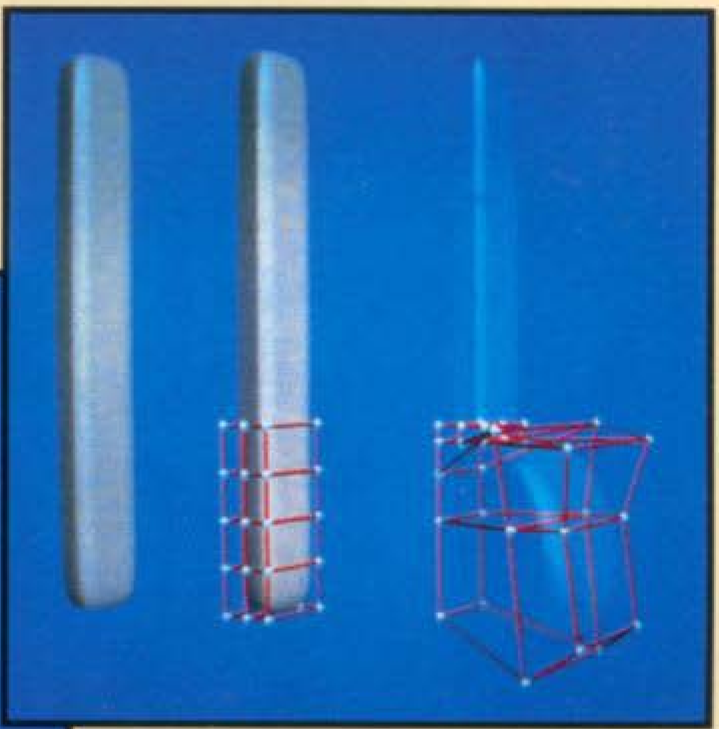


Plate IV.5 The end of a surface is placed within a box and the control points for the box are adjusted. The surface within the box is dragged along as well to form a new shape. (Courtesy of Thomas Sederberg and Scott Parry.)

Plate IV.6 A hammer modeled using free-form deformations. (Courtesy of Thomas Sederberg and Alan Zundel.)

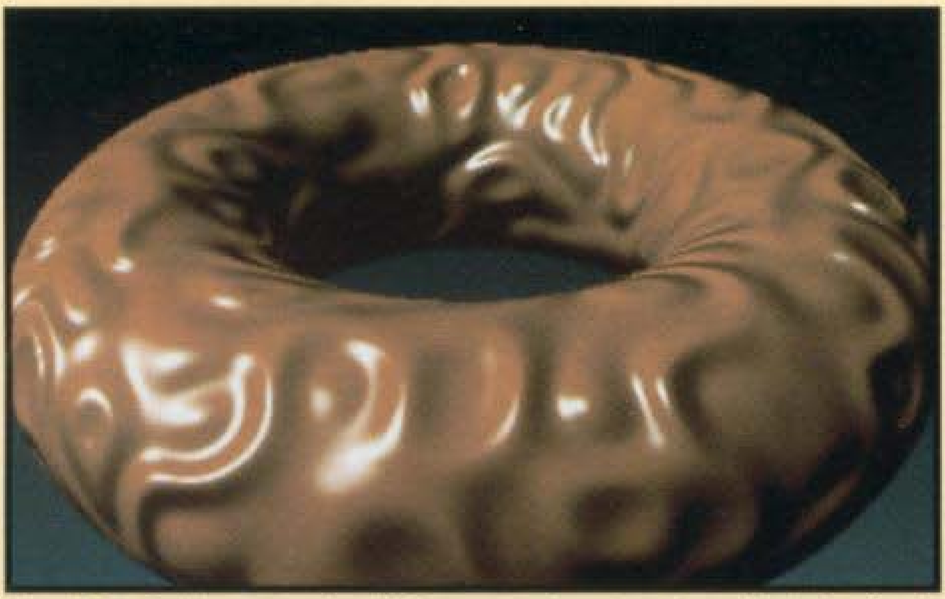
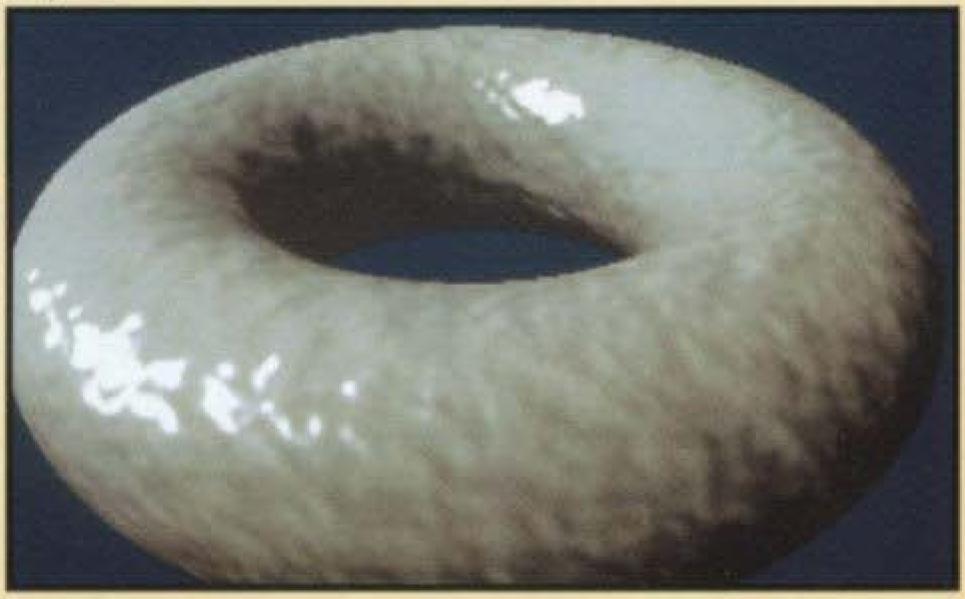
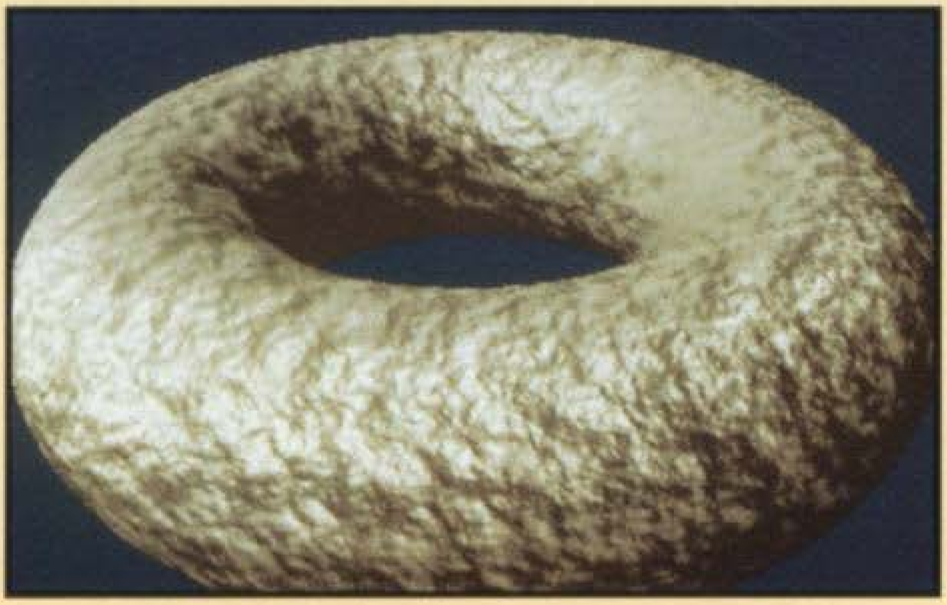

(a)

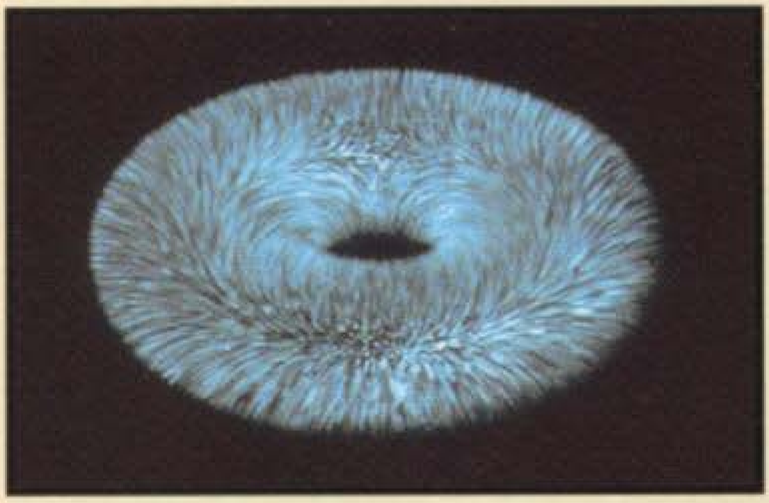
Plate IV.7 Solid textures (a–d). The stucco doughnut is particularly effective. (Courtesy of Ken Perlin.)

(b)




(c)

(d)



(a)

(b)

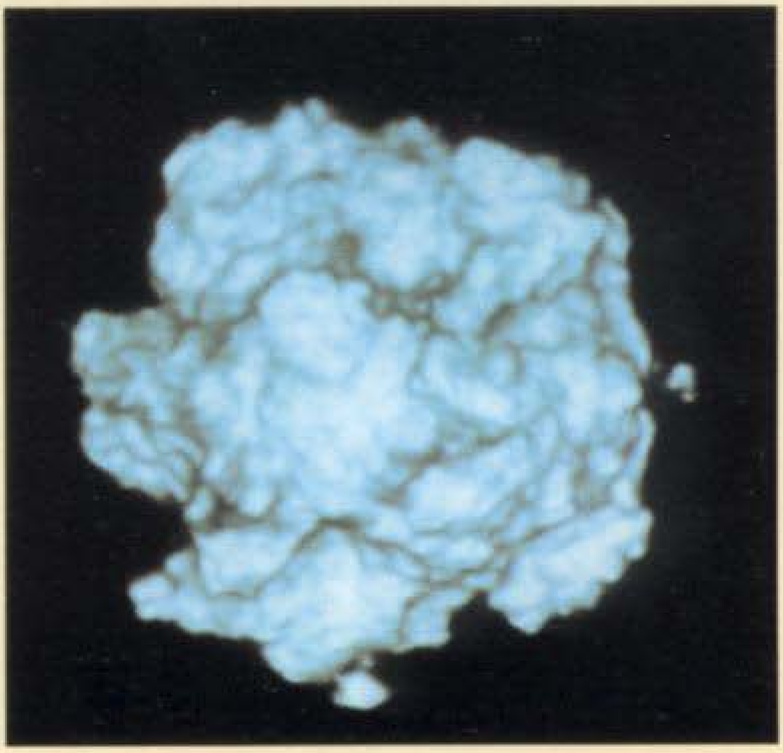


Plate IV.8 (a) A hairy donut modelled with hypertextures. (b) A hypertextured blob. (Courtesy of Ken Perlin.)

Plate IV.9 A hypertextured cube, showing how the texturing affects the geometry as well as the colors and normals of the surface. (Courtesy of Ken Perlin.)

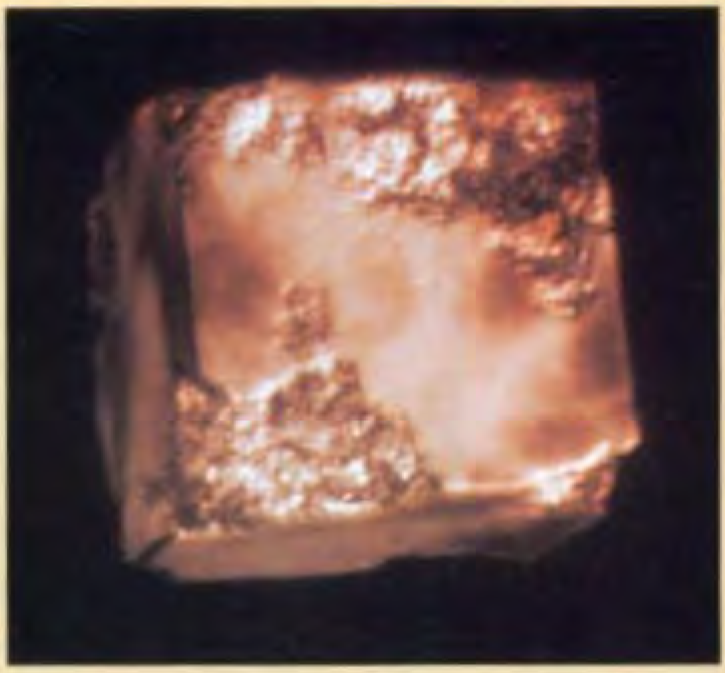


Plate IV.11 "Vol Libre Ridge": ▶ Fractal mountains generated with the Fournier–Fussell–Carpenter algorithm. (Courtesy of Loren Carpenter.)

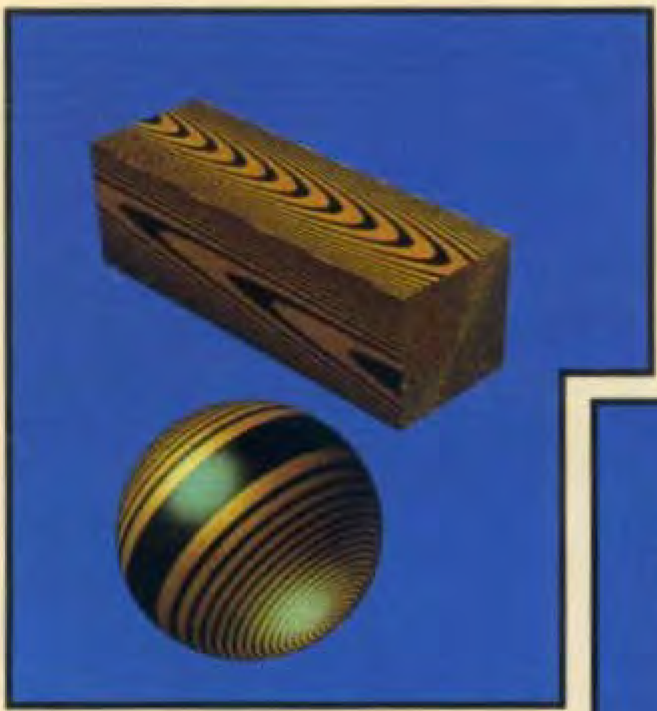
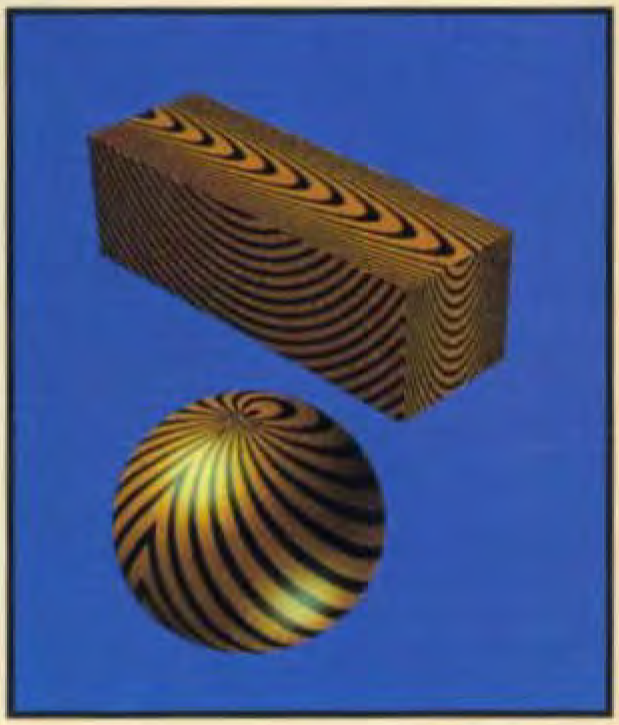
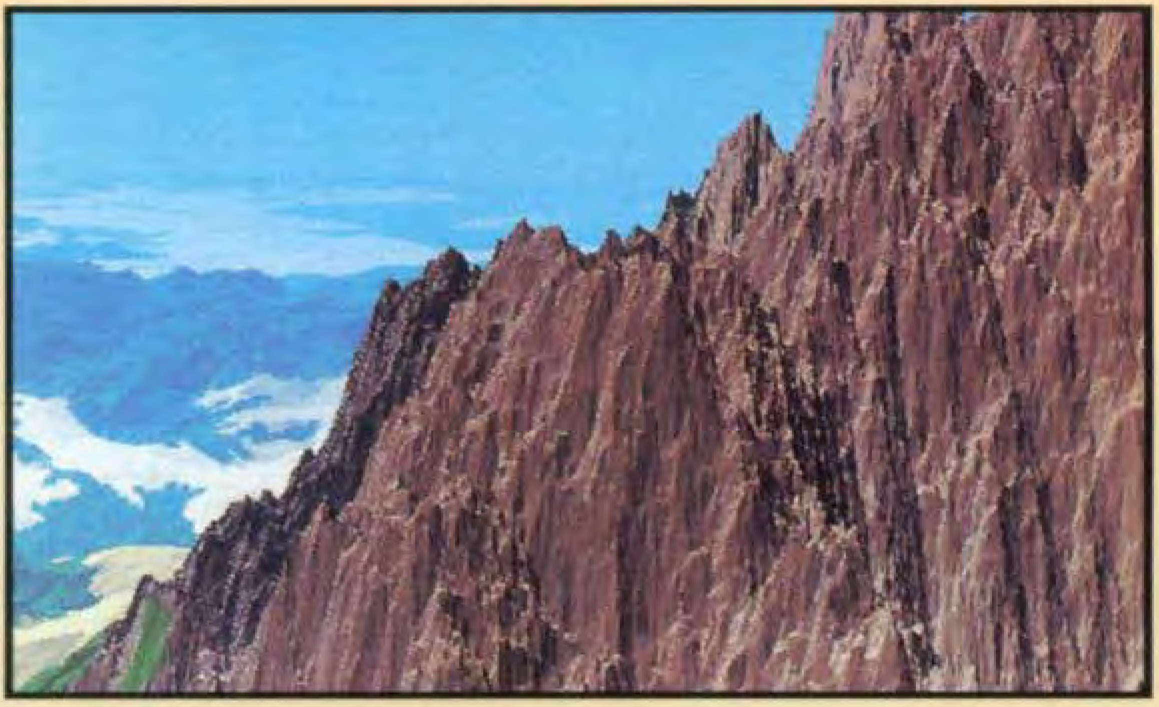


Plate IV.10 Solid textures generated using projection textures. Note that the patterns differ along each of the three principal axes. (Copyright © 1985 by Darwyn Peachey, Univ. of Saskatchewan, reprinted from [PEAC85]).





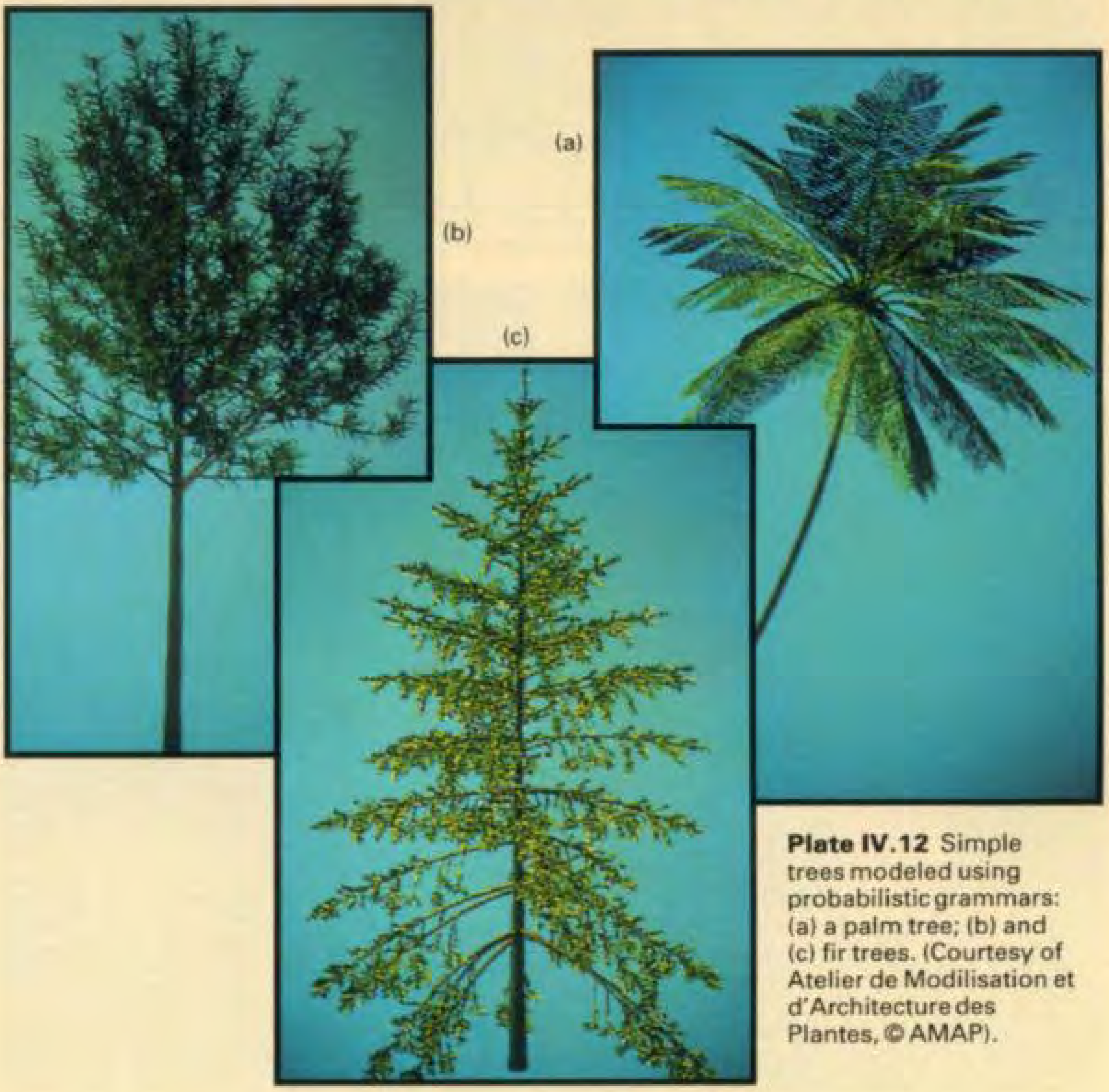
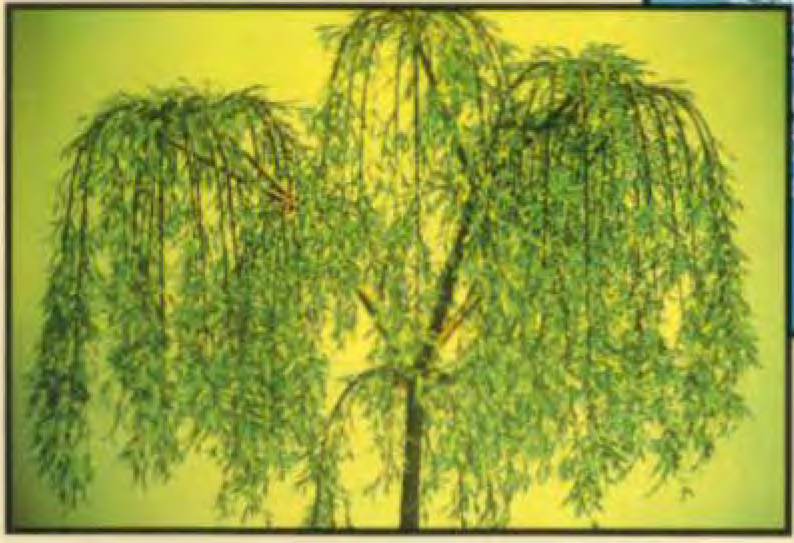
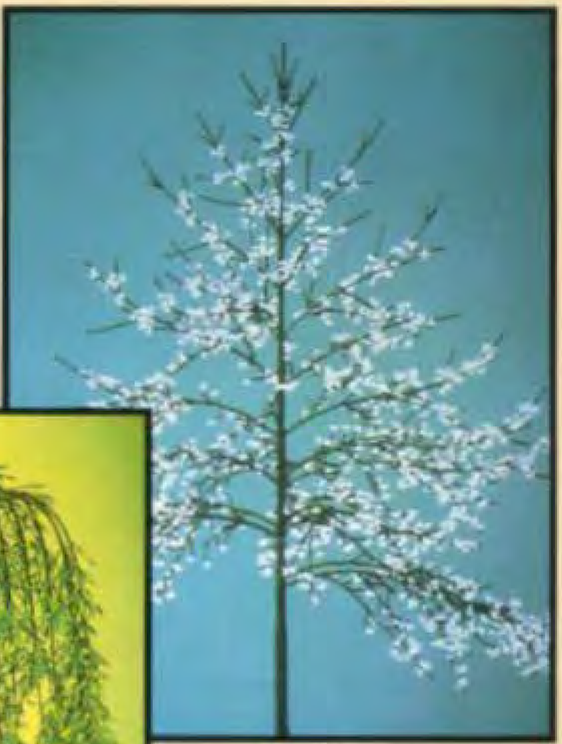


Plate IV.12 Simple trees modeled using probabilistic grammars: (a) a palm tree; (b) and (c) fir trees. (Courtesy of Atelier de Modilisation et d'Architecture des Plantes, © AMAP).

Plate IV.13 More complex trees modeled with the same techniques as in Color Plate IV.12, but with different parameters. (a) A willow tree; (b) a fruit tree in Spring. (Courtesy of Atelier de Modilisation et d'Architecture des Plantes, © AMAP).

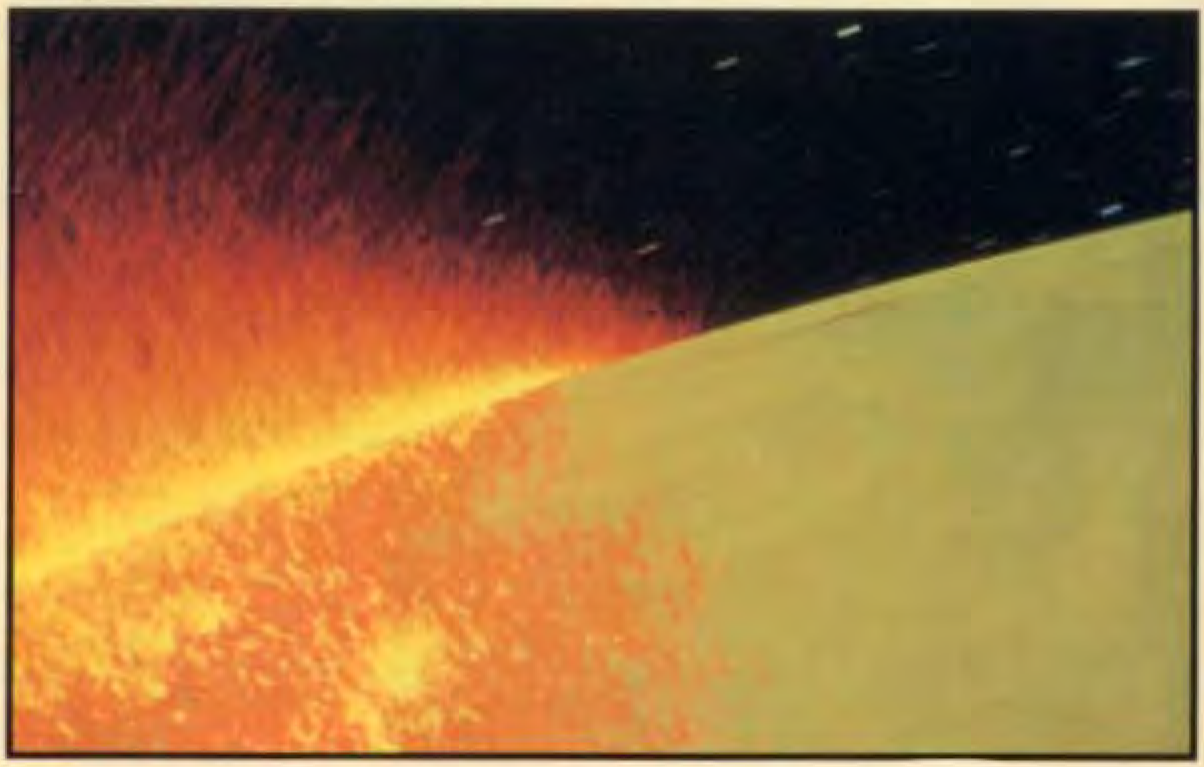


(a)



(b)

Plate IV.14 From *Star Trek II: The Wrath of Khan:* One frame from the Genesis effect particle-system animation of an explosion expanding into a wall of fire that engulfs a planet. (Courtesy of Ralph Guggenheim, Pixar. © 1982 Paramount Pictures Corporation, all rights reserved.)



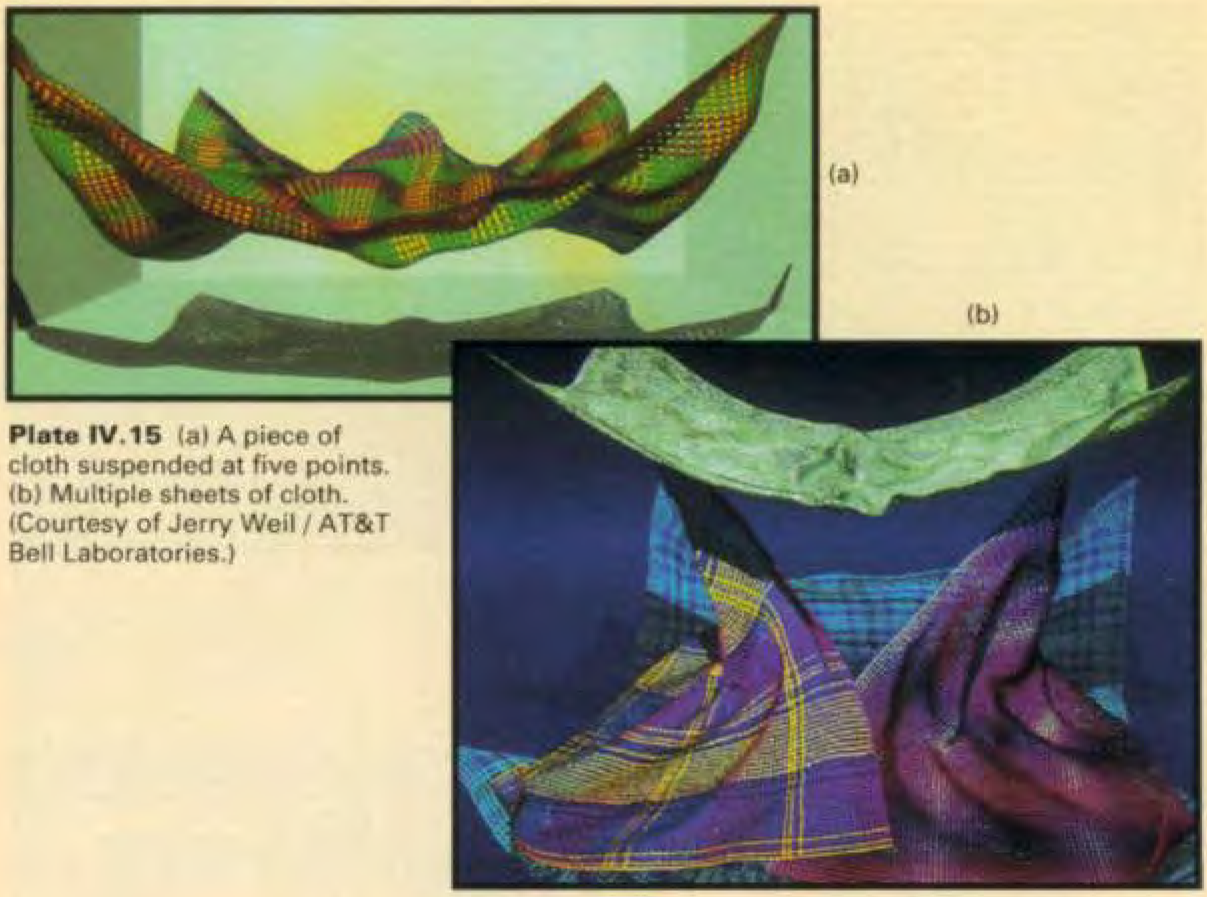
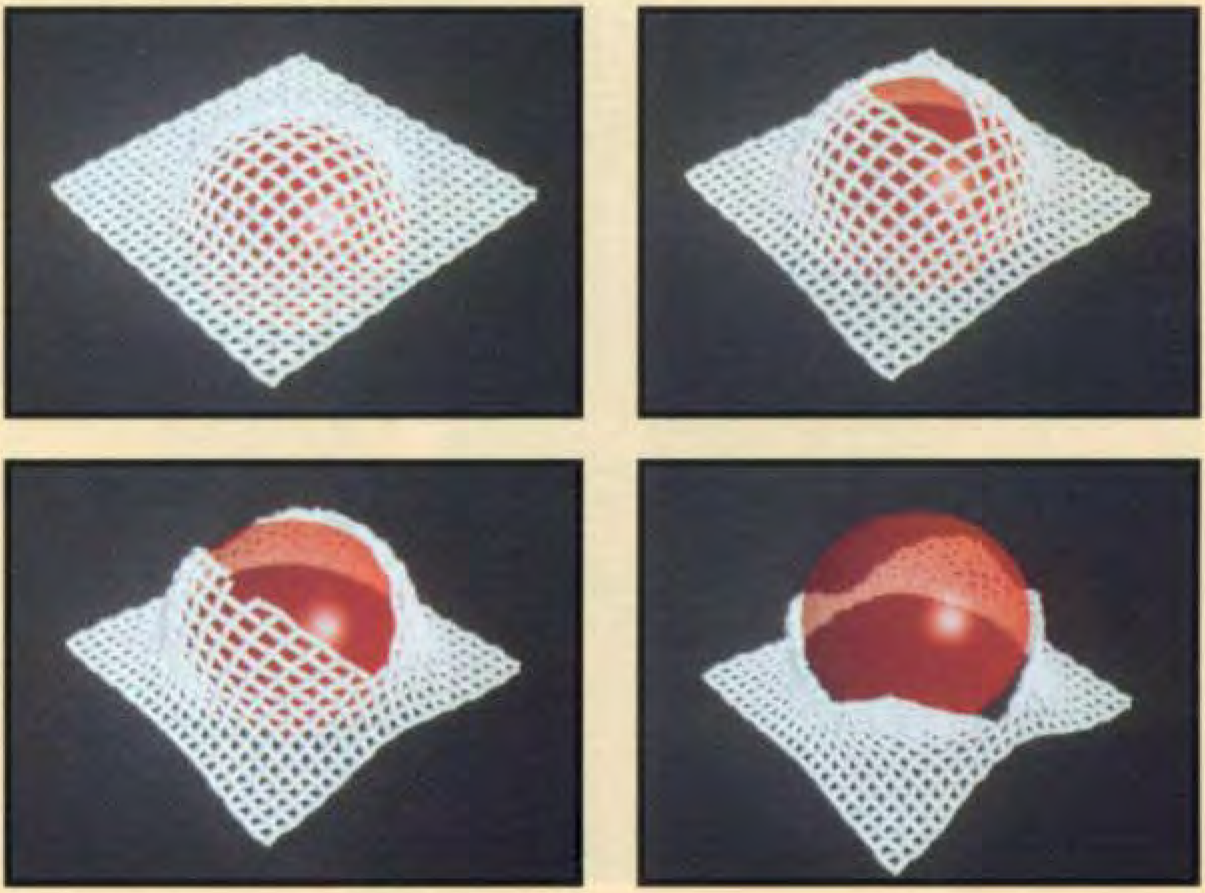


Plate IV.15 (a) A piece of cloth suspended at five points. (b) Multiple sheets of cloth. (Courtesy of Jerry Weil / AT&T Bell Laboratories.)

Plate IV.16 A net falling over a spherical obstacle, with fractures developing. (Courtesy of Demetri Terzopoulos and Kurt Fleischer, Schlumberger.)



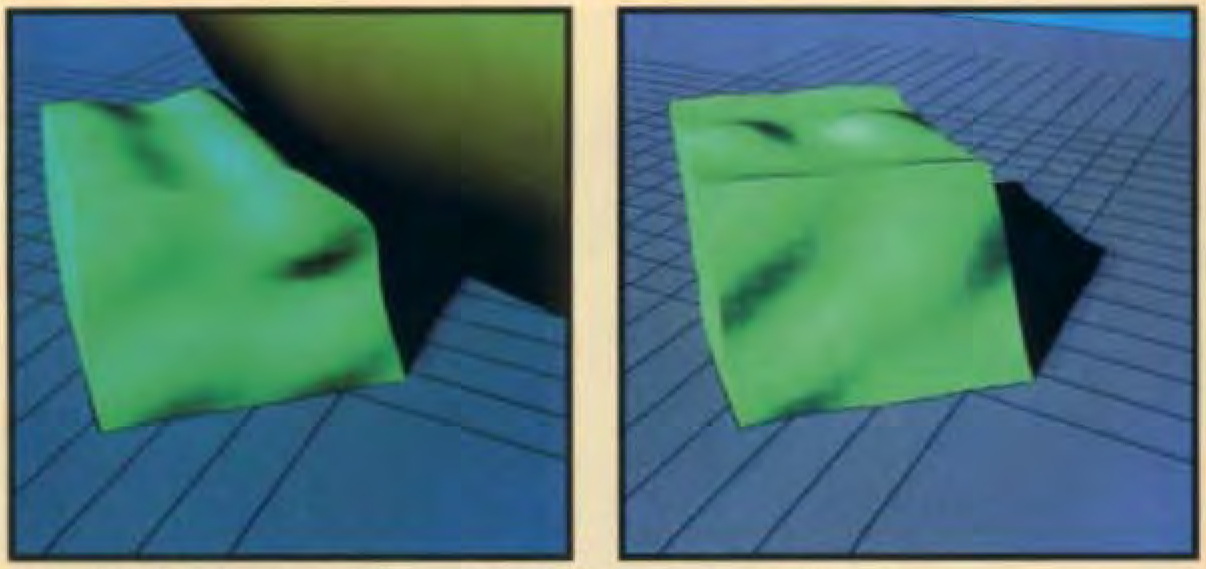


Plate IV.17 An elastic model is squashed by a large sphere and then returns to its rest shape. (Courtesy of Caltech Computer Science Graphics Group, John Platt and Alan Barr.)

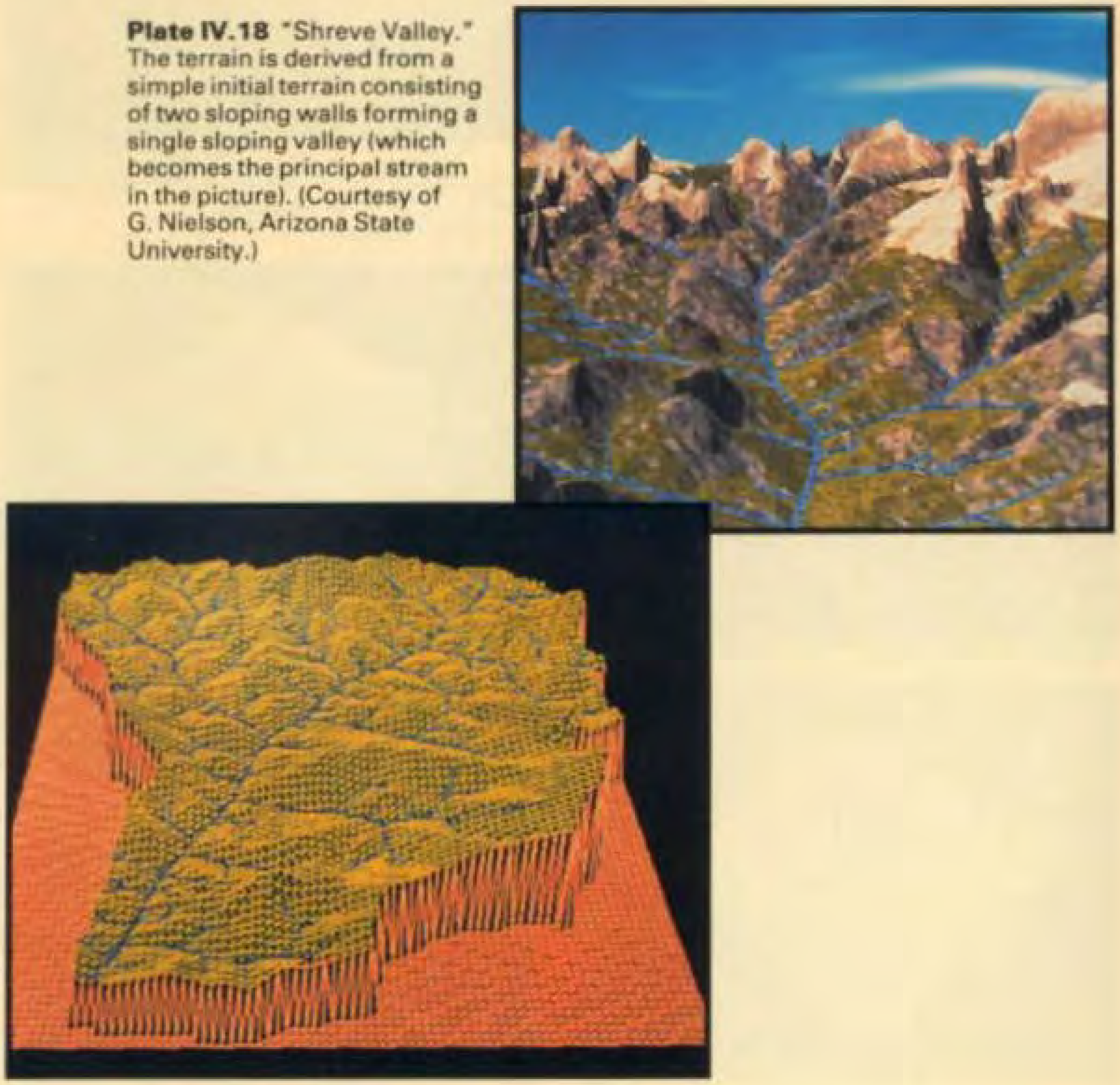


Plate IV.18 "Shreve Valley." The terrain is derived from a simple initial terrain consisting of two sloping walls forming a single sloping valley (which becomes the principal stream in the picture). (Courtesy of G. Nielson, Arizona State University.)

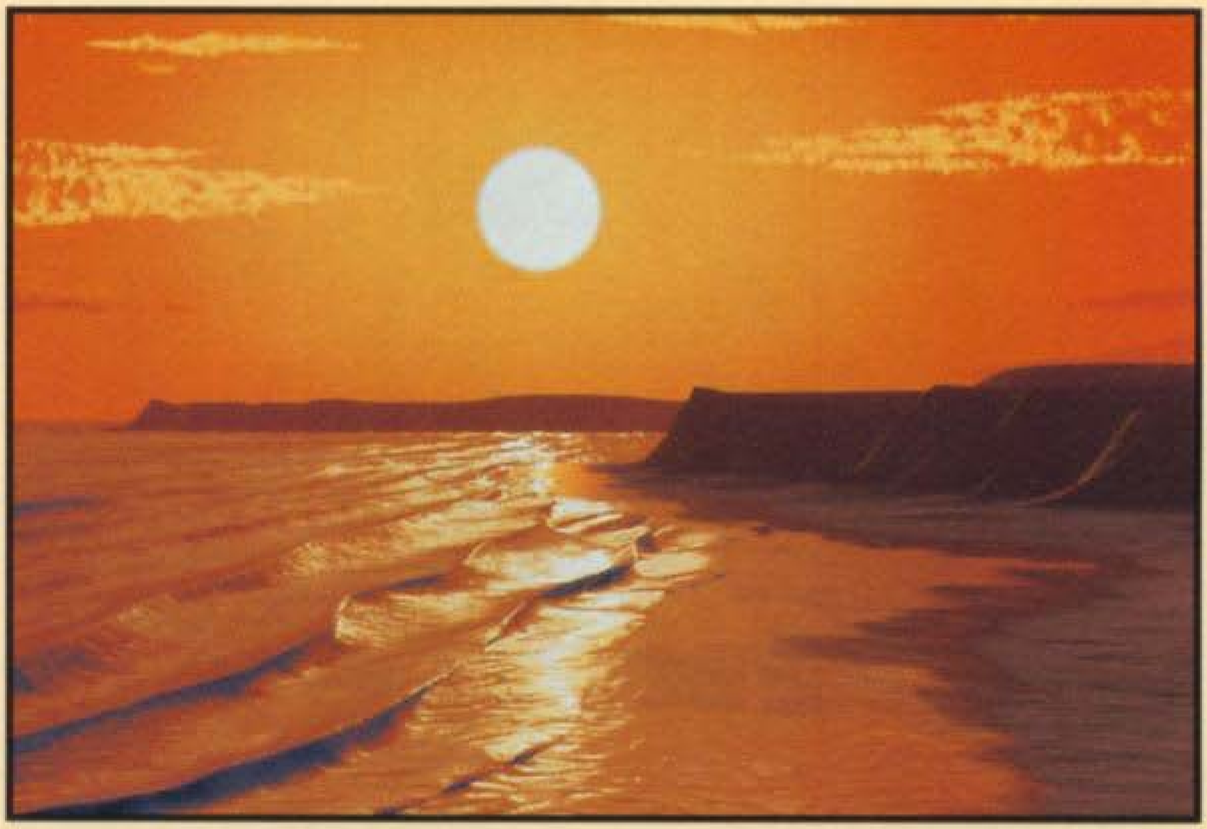
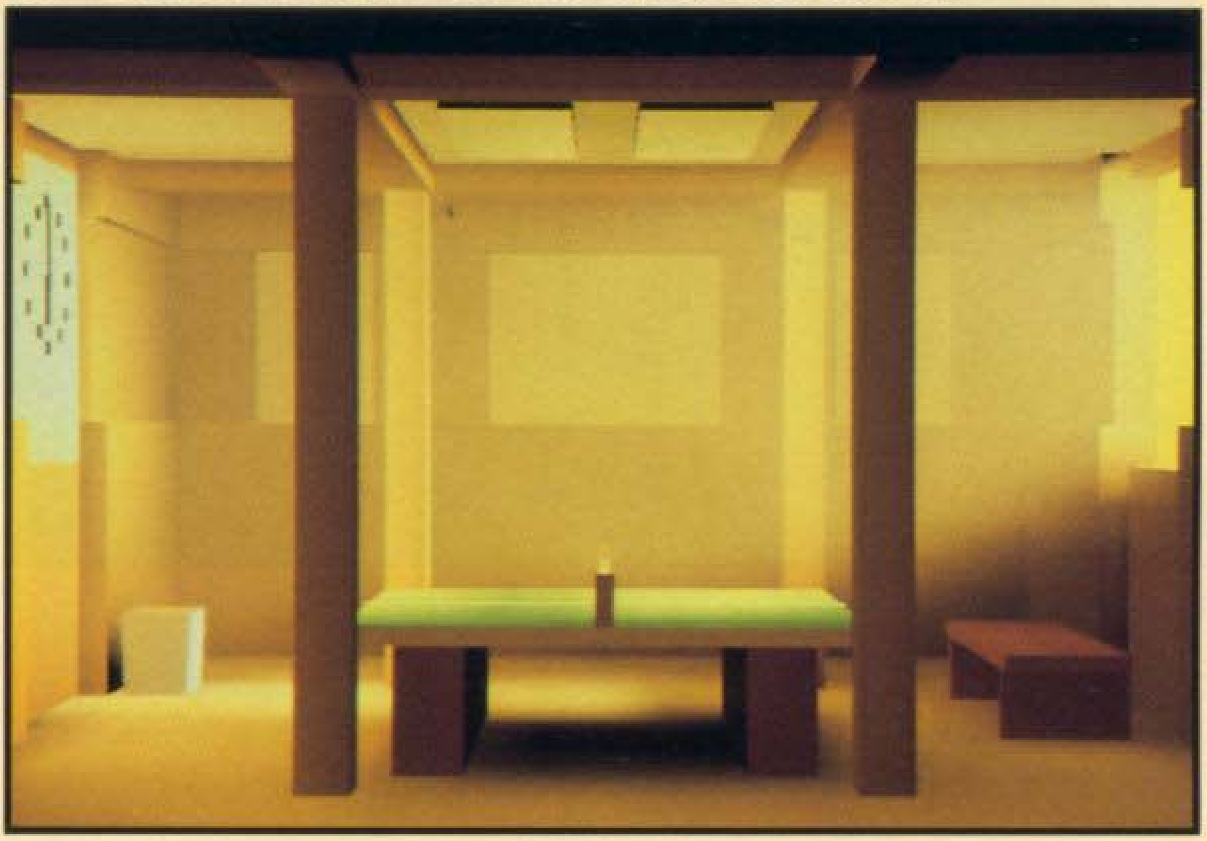


Plate IV.19 A beach at sunset. (Courtesy of Bill Reeves, Pixar, and Alain Fournier, University of Toronto.)

Plate IV.20 A late afternoon scene with a scattering medium in a room. (Holly Rushmeier, Courtesy of Program of Computer Graphics, Cornell University.)



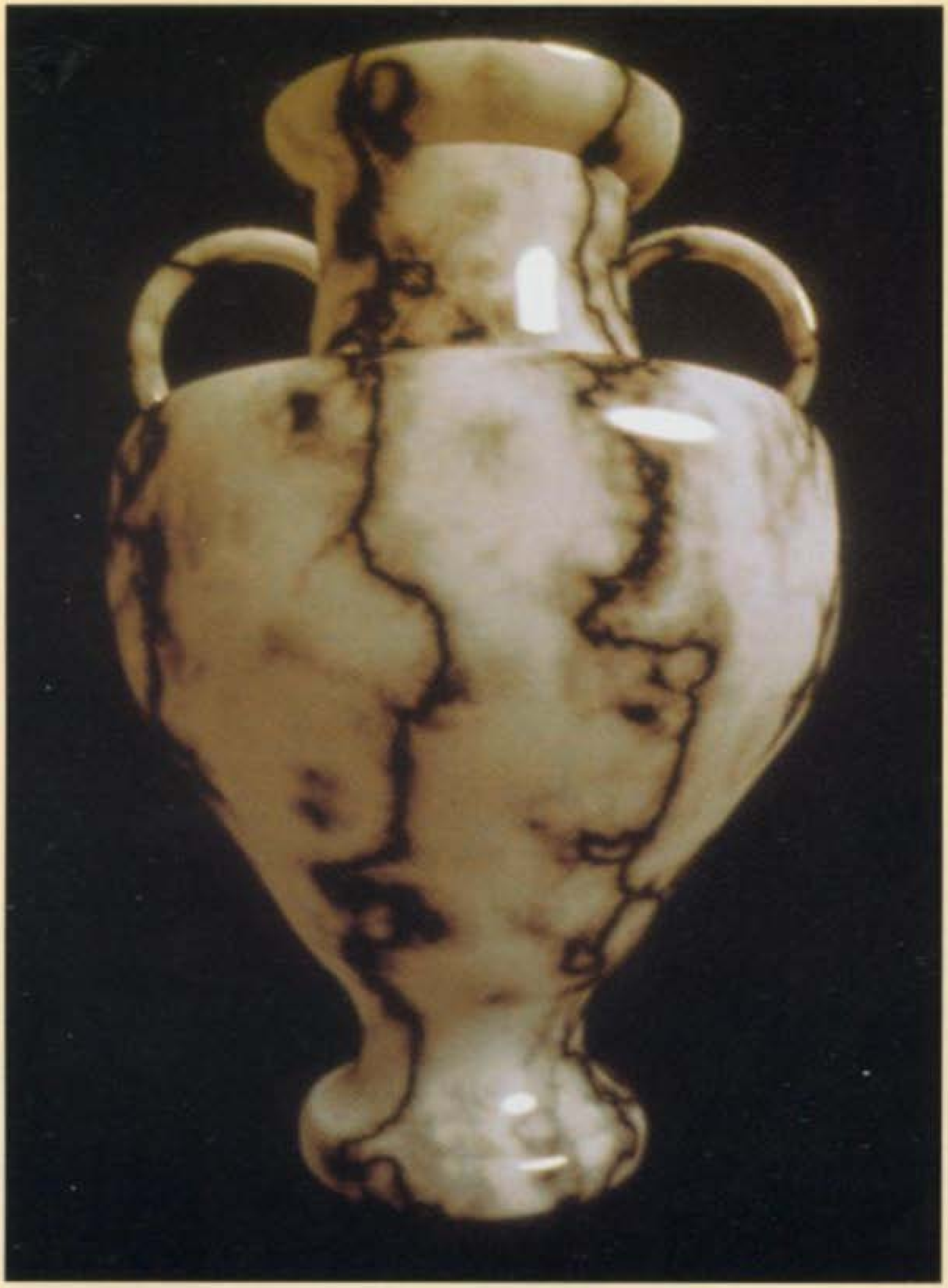
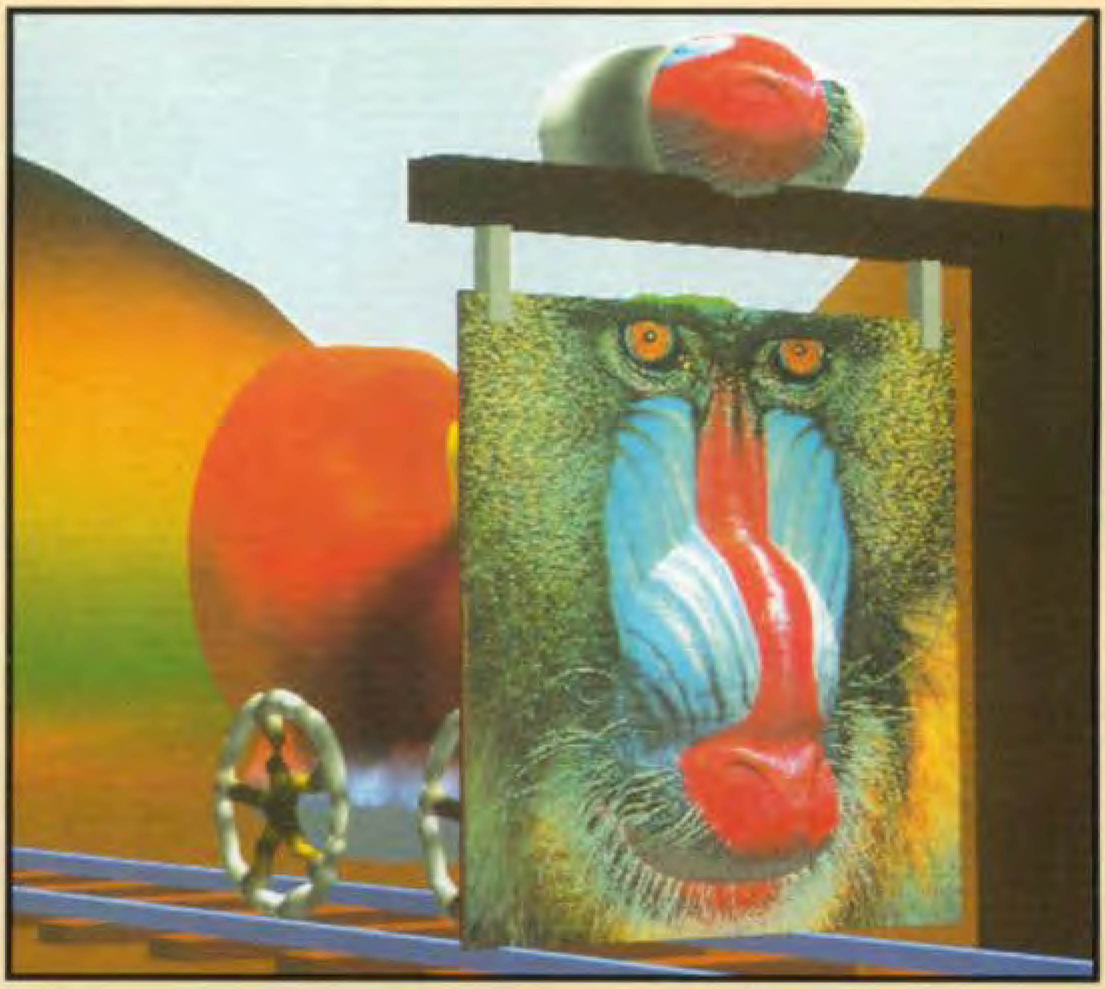


Plate IV.21 A marble vase modeled with solid textures. (Courtesy of Ken Perlin.)



▲

Plate IV.22 A train modeled with soft objects. ("Entering Mandrill Space" from *The Great Train Rubbery* by Brian Wyvill and Angus Davis, University of Calgary.)

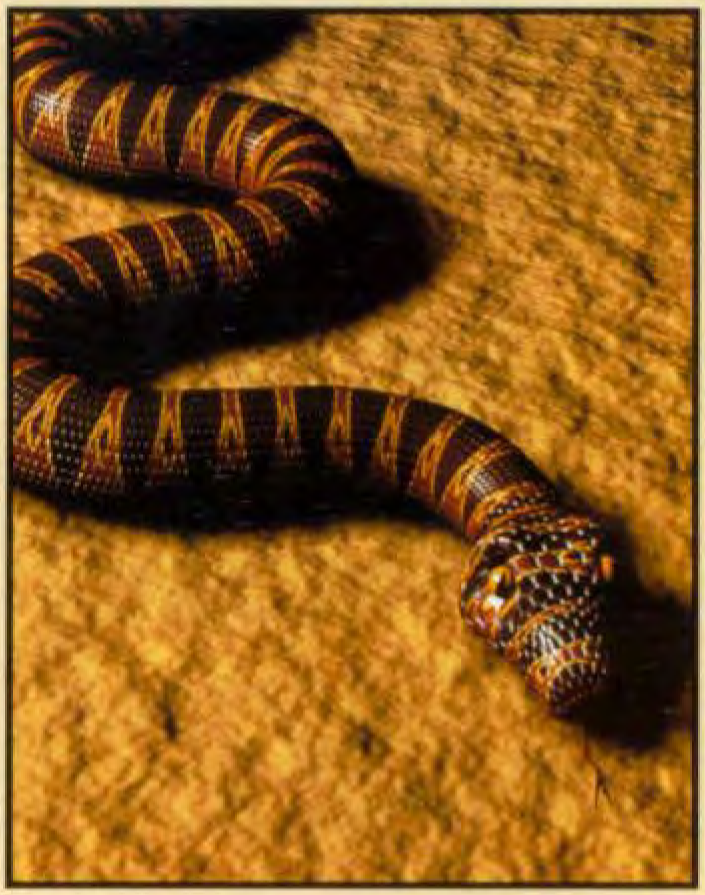
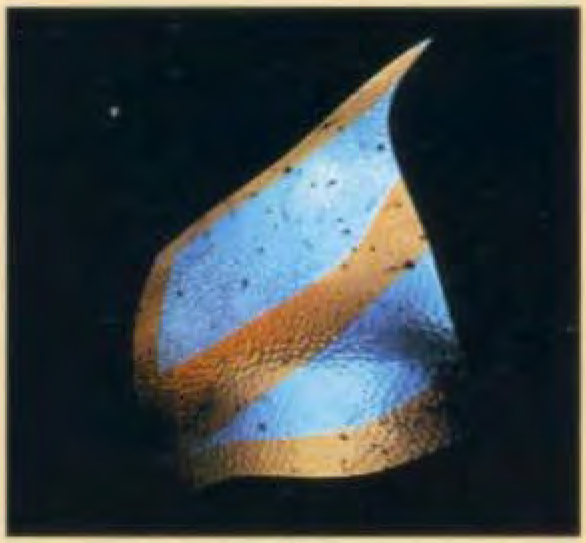


Plate IV.23 The snake was produced by Alias Research for the September 1988 cover of *IEEE Computer Graphics and Applications*. The snake was modeled with the ALIAS system and rendered with a color texture map for the skin markings and a bump map for the scales. (Produced by Gavin Miller and Robert LeBlanc of Alias.)

(a)

(b)

(c)


(d)

(e)

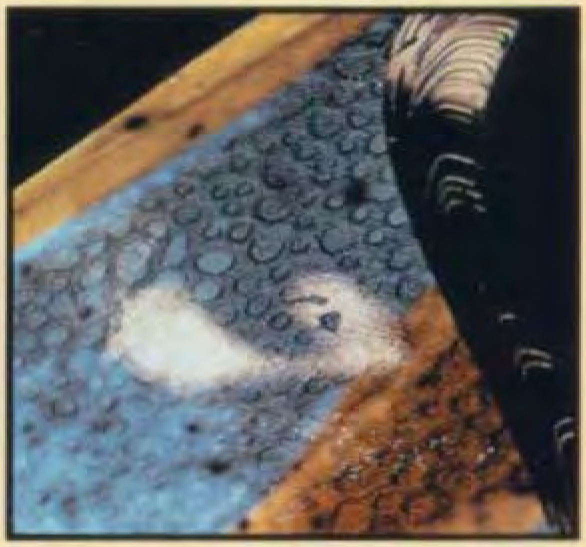

(f)

Plate IV.24 The construction of the glass knight for the movie, *Young Sherlock Holmes.* (a) The basic shape of the shoulder guard, with a color map, (b) the piece with an environment map, (c) the environment map modified by a bump map and illumination function, (d) spots of dirt and small bubbles are added, (e) an additional color map is added to provide the stains for the seams and rivets, (f) a detail of the piece, (g) the complete figure; the shoulder piece is in the upper right. (Copyright © 1989 Paramount Pictures. All rights reserved. Courtesy of Industrial Light & Magic.)

(g)

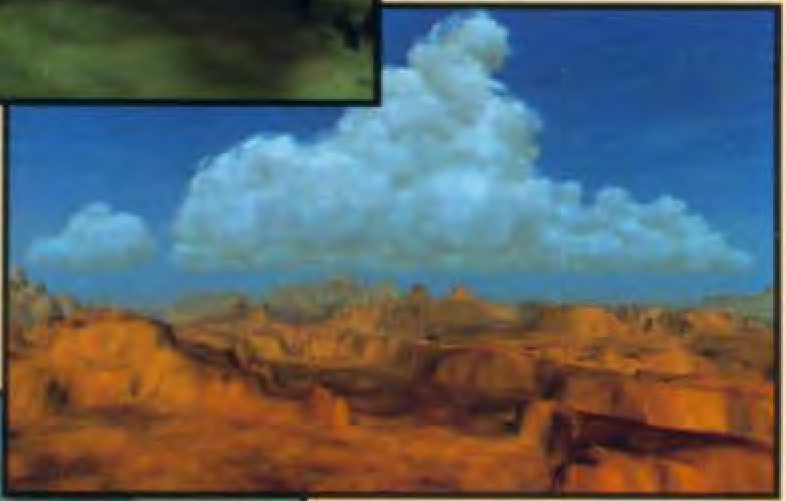


Plate IV.25 The trees in these scenes were positioned with an automatic placement mechanism. The objects are generated using textured quadrics and fractal models. (Courtesy of G.Y. Gardner, Grumman Data Systems.)

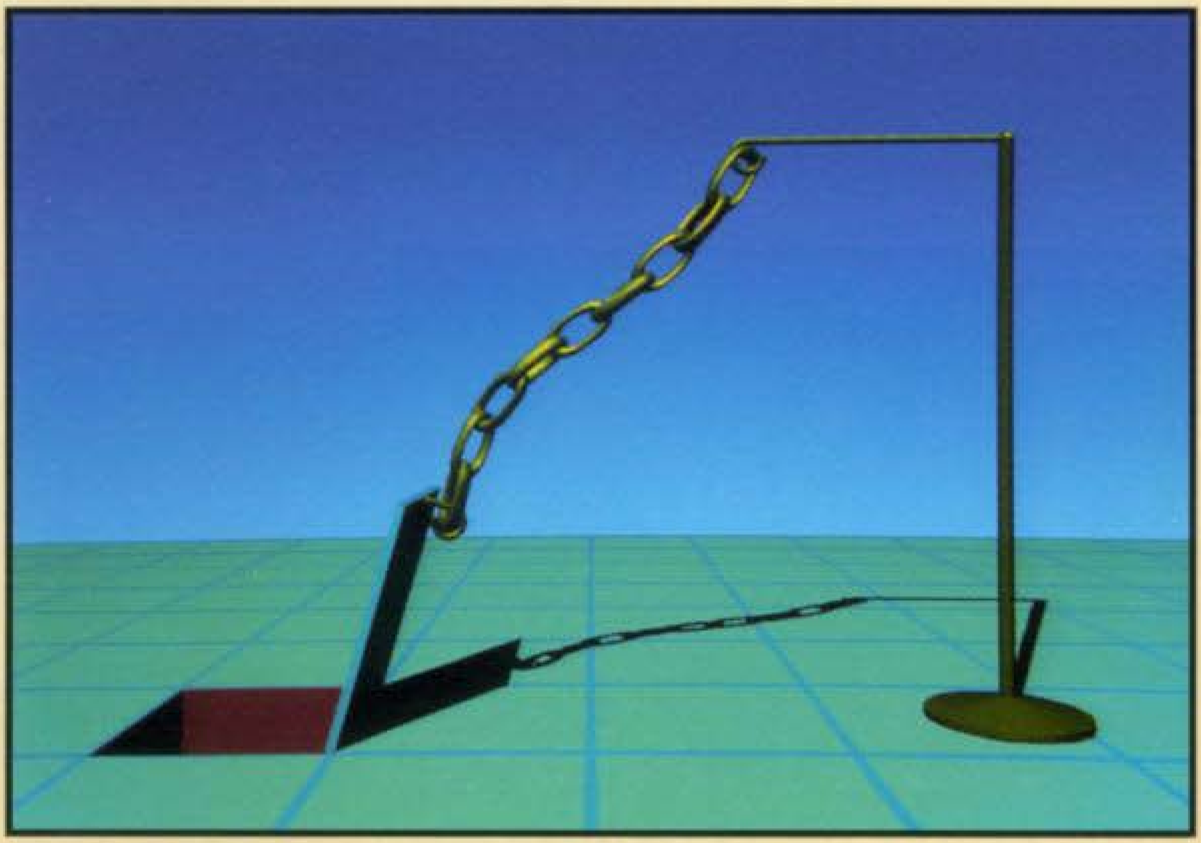


Plate IV.26 A self-assembling system modeled with dynamic constraints. (Courtesy of Caltech Computer Graphics Group, Ronen Barzel and Alan Barr.)

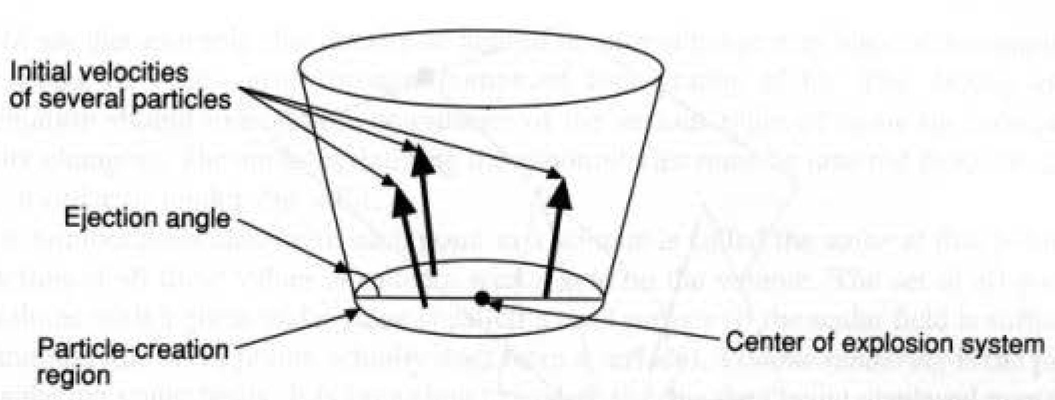

Initial velocities of several particles
Ejection angle
Particle-creation region
Center of explosion system

Fig. 20.17 The initial stage of a particle system for modeling an explosion.

the planet. Reeves achieved the nearby lighting by placing a conical light source of high intensity near the surface of the planet, instead of computing the direct illumination from the particles.

In the forest scene for the movie "André and Wally B." [REEV85; LUCA84], a different rendering scheme was required, since the particle systems were no longer light emitters, but instead were trees and grass, which acted as light reflectors. Special-purpose techniques were developed to render the particle systems; some trees obscured others, various portions of the trees were in shadow, the grass was sometimes in shadow from the trees, and so on. The solutions were two-fold: developing probabilistic models for shadowing and using modified *z*-buffer techniques to compute obscuring. The particles in the tree (leaves and stems) were shaded by computing the depth of the particle into the tree along a ray from the light source to the particle (see Fig. 20.18). This depth was used to compute an exponential drop off in the diffuse component of the light: $D = e^{-kd}$, where D is the diffuse component, k is a constant, and d is the depth of the particle. Particles with small values of d had stochastically computed specular highlights; if d was small and the direction of the light and the direction of the branch were nearly perpendicular, a specular highlight might be added. Finally, the ambient light, which is small inside the tree and larger near its edge, was computed by setting $A = \max(e^{-js}, A_{\min})$, where j is a constant, s is the distance from the particle to the edge of the tree (in any direction), and $A_{\min}$ is a lower bound for the ambient light (even the deepest parts of the tree are slightly illuminated). If a

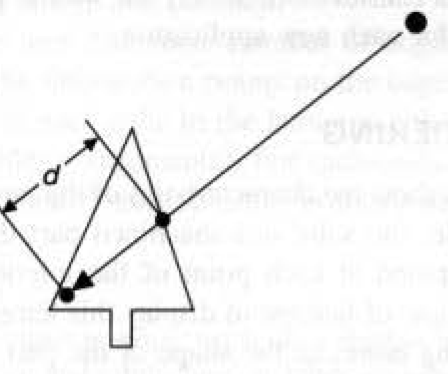


Fig. 20.18 Each point in a tree lies at some depth along the line from the light source to the particle. This distance determines the likelihood of the particle being illuminated.

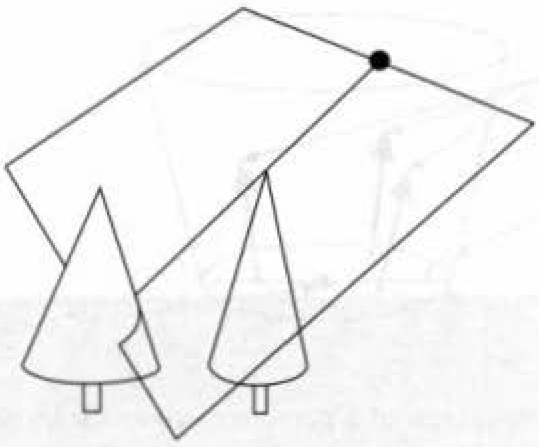


Fig. 20.19 The plane from an adjacent tree, which determines the shadowing of a tree.

tree is in shadow from another tree, the specular and diffuse components should not be added. This was implemented by determining planes from nearby trees to the tree under consideration; the plane contained the top of the nearby tree and the light source, and had the largest possible y component in its normal, as shown in Fig. 20.19. Particles above this plane were lit with all three components, whereas those below were given (probabilistically) less and less diffuse and specular light as the distance from the plane increased.

Even with these simplified lighting computations, visible surfaces still had to be determined. The trees in the scene were sorted back to front, and were rendered in that order. Trees were rendered with a bucket-sort type of z-buffer. Each tree's depth extent was divided into a great many buckets; every particle that was generated was inserted into the bucket for its depth in the tree. When all particles had been generated, they were rendered in back-to-front bucket order. Each particle was drawn as a small circle or short line segment (antialiased). After each tree was rendered, the information about the tree was discarded. The result of this ordering is that a branch of a nearby tree may obscure a branch of one slightly farther away, even though the second branch lies in front of the first, since the first branch is part of a tree that is rendered (entirely) after the second. In scenes with sufficient complexity, this sorting error seems not to be a problem.

Still, this difficulty in rendering the scene does highlight a drawback of particle systems in general: The modeler gets considerable power, but special-purpose rendering techniques may need to be developed for each new application.

20.6 VOLUME RENDERING

Volume rendering is used to show the characteristics of the interior of a solid region in a 2D image. In a typical example, the solid is a machined part that has been heated, and the temperature has been computed at each point of the interior through some physical or mathematical means. It is now of interest to display this temperature visually. This is not, strictly speaking, a modeling issue, as the shape of the part and the characteristics to be displayed are both available a priori. But the conversion of these data to information in a pixel map is a form of modeling; namely, the modeling of the transformation from 3D to

2D. In another example, the density of human or animal tissue may have been computed at each point of a 3D grid through computed tomography (CT). The display of this information should indicate the boundaries of the various types of tissue (as indicated by density changes). The surfaces defining these boundaries must be inferred from the sample data in order to render the solid.

A number associated with each point in a volume is called the *value* at that point. The collection of all these values is called a *scalar field* on the volume. The set of all points in the volume with a given scalar value is called a *level surface* (if the scalar field is sufficiently continuous, this set of points actually does form a surface). *Volume rendering* is the process of displaying scalar fields. It is important to realize that the data being displayed may not be ideal. If the data have been sampled at the points of a regular grid, the scalar field they represent may contain frequencies higher than the Nyquist frequency for the sampling (see Chapter 14). In tomography, for example, the transition from flesh to bone is very abrupt, and hence contains very high frequencies, but the sampling rate is likely to be too low to represent this change accurately. Also, the data that describe the interior of a solid may be clustered in some irregular pattern, as might arise in geographic data taken from core samples, where it may be impossible to sample uniformly.

Several approaches to volume rendering have been developed. They can be divided into two categories: those that compute level surfaces and those that display integrals of density along rays. The two can be combined by assigning density only to certain level surfaces and then ray tracing the result (which amounts to creating a different volume to be displayed). If animation is available, a third category of display is possible: a series of 2D slices of the data is computed and displayed sequentially, using color or brightness to indicate the scalar value at each point of the slices. If interactive control of the slice direction and level is provided, this approach can give an excellent sense of the interior structure of the scalar field.

Nonetheless, it is sometimes useful to view data in the aggregate, rather than by slices. One approach (though by no means the first) is the *marching-cubes* algorithm. In this algorithm, scalar values are assumed to be given at each point of a lattice in 3-space. A particular level surface can be approximated by determining all intersections of the level surface with edges of a lattice.[6] We look for pairs of adjacent lattice points whose field values surround the desired value (i.e., the value of one vertex is greater than the chosen level, the value of the other is less). The location of an intersection of the level surface with the edge is then estimated by linear interpolation.

Each cube in the lattice now has some number of edges marked with intersection points. The arrangement of the intersection points on the edges can be classified into 256 cases (each of eight vertices of each cube in the lattice is either above or below the target value, giving $2^8 = 256$ possible arrangements). For each case, a choice is made of how to fill in the surface within the cube. Figure 20.20 shows two such cases.

[6]A *lattice* is an array of points and lines in space, much like a children's jungle gym. The points of the lattice are evenly spaced in the x, y, and z directions, and they are joined by line segments parallel to the coordinate axes. The set of all points with integer coordinates and of all axis-parallel line segments joining them constitutes an example, called the *integer lattice*.

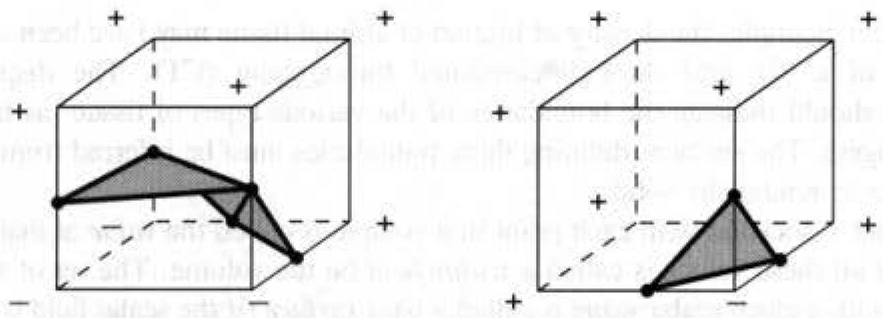


Fig. 20.20 Two possible arrangements of intersections of a level surface with a cube in the integer lattice, with choices of how to fill in a surface for each.

The collection of all the surface pieces just defined constitutes a surface. This surface can be assigned (at each subpolygon) a normal vector to be used for shading in the following manner. At each vertex of the cube, a numerical estimate of the gradient of the scalar field is made. These values are interpolated to estimate the gradient vector at some point of the subsurface. Since the gradient of a scalar field always lies in the direction of the normal vector to the level surface, this interpolated value provides a good estimate for the normal vector. (The special case of zero must be handled separately.)

The resulting level surface can be rendered with conventional techniques. This strategy can be of use in medical imaging to show the shape of the boundary between different types of tissue. Unfortunately, it computes only one shape at a time, and the relative positions of different layers are difficult to see.

Upson and Keeler [UPSO88] also assume that the scalar field varies linearly between sample points, and they present two methods for its display. In both, the user first creates four functions, R, G, B, and O, where O is *opacity*. The arguments of these functions are values of the scalar field; we therefore assume that the scalar field has been normalized to have values between 0 and 1. The choices of the R, G, B, and O functions drastically affect the resulting image. If the functions are chosen to have tight peaks at particular values of the scalar field, the level surfaces for those values are highlighted. If the functions are chosen to vary smoothly over the field values, then color can be used to indicate field value (see Fig. 20.21). Thus, in effect, we obtain sophisticated color-map pseudocoloring.

The interpolation of the scalar field over each cube in the lattice of sample points is a linear equation in each variable, and hence is trilinear in 3-space (i.e., of the form $S(x, y, z) = A + Bx + Cy + Dz + Exy + Fxz + Gyz + Hxyz$). If we parameterize a ray in the form $(x, y, z) = (a, b, c) + t(u, v, w)$ as in ray tracing, then the value of S at points of the ray is a cubic function of t.

The ability to compute this cubic rapidly forms the basis for Upson and Keeler's first rendering method, based on a ray-tracing mechanism for volume data developed in [KAJI84]. For each ray from the eyepoint through an image pixel, the R, G, B, and O values are accumulated for the ray as it passes through the volume data. This accumulation stops when the opacity reaches a value of 1 or the ray exits the volume, whichever happens first. Actually, far more is accumulated: the scalar field, shading function, opacity, and depth cueing are all computed at each of several steps within each pixel volume so as to integrate the cubic interpolant accurately.

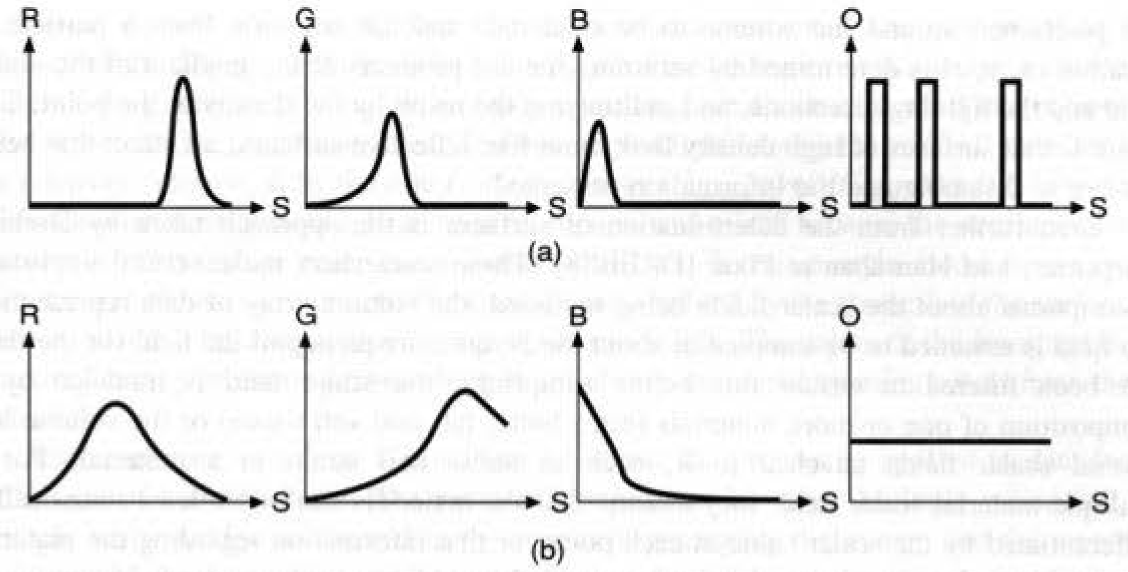


Fig. 20.21 Two different choices for the shapes of the R,G,B, and O functions. In (a), certain level surfaces of the scalar field are highlighted in red, green, and blue. In (b), the color will change gradually as a function of the scalar field.

Upson and Keeler's second rendering method uses the same basic notion of integration along rays, but accumulates values in pixels by processing the cubes in the lattice of values in front-to-back order (which can be easily determined for any particular view orientation). The authors take great pains to ensure the computational efficiency of the process by using adaptive quadrature methods for the integrations and never solving a system of equations more than once at each point (when performing interpolations). It is important to observe, as they do, that this method "is designed as an analytic tool, not as a technique to synthesize realistic images" [UPSO88, p. 64].

Sabella takes a similar approach [SABE88]. He assigns a *density emitter* to each point in the volume to be rendered, to simulate light coming from translucent objects. The simulation models only part of the effects of light in such media; namely, the occlusion of parts deeper in the medium by those nearer the front. Sabella deliberately ignores shadowing and the variation in color due to differences in scattering at different wavelengths, asserting that they may actually detract from the perception of density variation. The density emitters are imagined to be tiny particles that both emit and scatter light. The density of such particles within each small region of space is given by the value of the scalar field there. The light reaching the eye along any ray is computed by summing up the emission from all the emitters along the ray, and then attenuating the light from each emitter by the probability that it is scattered during its travel to the eye. Sabella computes four numbers: M, the peak value of the scalar field along the ray; D, the distance at which that peak is encountered; I, the attenuated intensity just described; and C, the "center of gravity" of the density emitters along the ray. By mapping combinations of these numbers into various color scales (e.g., using hue–saturation–value, he maps M to hue, D to saturation, and I to value), he can highlight various characteristics of the scalar field. He further allows for "lighting" effects by giving a directionality to the particle emissions. Each particle's emissions are attenuated by a Lambert lighting model: Several light sources are positioned around the volume to be rendered, and the emission from a particle at location (x, y, z) is determined by summing the dot products of the gradient of the scalar field and the lighting directions, and multiplying the result by the density at the point. The result is that surfaces of high density look more like reflective surfaces, an effect that helps the eye to disambiguate the information presented.

Even further from the determination of surfaces is the approach taken by Drebin, Carpenter, and Hanrahan at Pixar [DREB88]. These researchers make several important assumptions about the scalar fields being rendered: the volume array of data representing the field is assumed to be sampled at about the Nyquist frequency of the field (or the field has been filtered to ensure this before sampling); the scalar field is modeled by a composition of one or more materials (e.g., bone, fat, and soft tissue) or the volume has several scalar fields attached to it, such as stress and strain in a material. For a multiple-material scalar field, they assume that the materials can be (at least statistically) differentiated by the scalar value at each point, or that information regarding the material composition of each volume element is provided in addition to the scalar field.

Given such information, they create several new scalar fields on the array of sample points: the *material percentage volumes* (they use the term *volume* to mean a scalar field on a volume). The value at a grid point in a material percentage volume is the percentage of one material present in the volume element (or *voxel*) surrounding that point. If multiple fields are specified in the original data, computing these material percentages may be simple. If only a single field is given, the material percentages may have to be estimated by Bayesian analysis.

After computing the material percentage volumes, the authors associate a color and opacity with each material; they then form composite colors and opacities by taking a linear combination of all the colors and opacities for each of the material percentage volumes. (Opacity here is used in the sense of the α channel described in Section 17.6, and the linear combinations are the same as the combinations described there. In particular, the colors are premultiplied by the opacity values before combination.) They further allow compositing with *matte volumes*, which are scalar fields on the volume with values between 0 and 1. By multiplying these matte volumes with the color/opacity volumes, they can obtain slices or portions of the original volumetric data. Making a smooth transition between 0 and 1 preserves the continuity of the data at the matte boundaries.

The lighting model used here is similar to that in the other two algorithms. A certain amount of light enters each voxel (the light from voxels behind the given voxel), and a different amount exits from it. The change in light can be affected by the translucence of the material in the voxel, or by "surfaces" or "particle scatterers" contained in the voxel that may both attenuate the incoming light and reflect light from external light sources. These effects are modeled by (1) requiring that light passing through a colored translucent voxel have the color of that voxel plus the incoming light multiplied by $(1 - \alpha)$ for that voxel (this is the **over** operation of the Feibush–Levoy–Cook compositing model in Section 17.6), and (2) determining surfaces and their reflectance and transmission properties.

The surface determination is not as precise as the ones described previously; each voxel is assigned a density that is a weighted sum of the densities of the component materials for the voxels (weighted by the material percentages). "Surfaces" are simply places where this composite density changes rapidly. The *strength* of a surface is the magnitude of the gradient of the density, and the surface normal used in shading calculations is the direction vector of the gradient. To compute the surface shading, we divide each voxel into regions in front of, on, and behind the surface. The intensity of light leaving the voxel, I', is related to the intensity entering, I, by the rule $I' = (C_{\text{front}}$ **over** $(C_{\text{surface}}$ **over** $(C_{\text{back}}$ **over** $I)))$. The three terms associated with the voxel can be precomputed and mixed because the **over** operator is associative. The surface color is computed by a Cook–Torrance–style model to give both specular and diffuse components; these values are weighted by the strength of the surface so that no reflective lighting appears in homogeneous solids. The colors of the front and back are computed by estimating from which material they came and by using colors from those materials.

The results are excellent. Color Plate I.1 shows the process as applied to data from a CT scan of a child's head. The process is expensive, however. Multiple volumes (i.e. multiple scalar fields) are created in the course of generating the image, and the memory requirements are vast. Also, the assumption that the fields are sampled at or above the Nyquist frequency may not be practical in all cases: sometimes, the data are given, and we wish to see the results even with some aliasing. Finally, the assumption that the data are from a heterogeneous mixture of materials is not always valid, so the applications of the method are limited.

20.7 PHYSICALLY BASED MODELING

The behavior and form of many objects are determined by the objects' gross physical properties (as contrasted with biological systems, whose behavior may be determined by the systems' chemical and microphysical properties). For example, how a cloth drapes over objects is determined by the surface friction, the weave, and the internal stresses and strains generated by forces from the objects. A chain suspended between two poles hangs in an arc determined by the force of gravity and the forces between adjacent links that keep the links from separating. *Physically based modeling* uses such properties to determine the shape of objects (and even their motions in some cases). Current work on this subject is collected in [BARR89].

Most of this modeling uses mathematics well beyond the scope of this book, but we can give the general notions of the techniques. It is in this sort of modeling that the distinction between graphics and other sciences is most blurred. The computations that produce a tear in a model of a thin cloth when it is dropped over an obstruction are purely in the domain of solid mechanics. But such computations would not be done unless the results could be displayed in some fashion, so the motivation for physical research is now being provided by the ability (or desire) to visualize results. At the same time, the wish to generate more realistic graphics models drives research in the physical modeling process. In this section, we discuss a few of the more impressive examples. The next section describes models of natural phenomena that are less directly based on scientific principles and may contain some (or many) compromises in order to produce attractive results. There is a continuous variation between scientific foundations and ad hoc approaches, and the dividing line is not at all clear.

20.7.1 Constraint-Based Modeling

When constructing an object out of primitive objects using Boolean operations, we find it convenient to be able to say ''I want to put this sphere on top of this cube so that they touch only at one point.'' Even with an interactive program that lets the user position objects by eye, it may be difficult to make the two objects touch at a single point.[7] Rules such as this one are called *constraints*. Constraint-based modeling systems allow the user to specify a collection of constraints that the parts of the model are supposed to satisfy. A model may be *underconstrained*, in which case there are additional degrees of freedom that the modeler can adjust (e.g., the location of the point of contact of the sphere and the cube), or *overconstrained*, in which case some of the constraints may not be satisfied (which could happen if both the top and bottom of the sphere were constrained to lie on the top face of the cube). In constraint-based modeling, the constraints must be given a priority, so that the most important constraints can be satisfied first.

The specification of constraints is complex. Certain constraints can be given by sets of mathematical equalities (e.g., two objects that are constrained to touch at specific points), or by sets of inequalities (e.g., when one object is constrained to lie inside another). Other constraints are much more difficult to specify. For example, constraining the motion of an object to be governed by the laws of physics requires the specification of a collection of differential equations. Such constraint systems, however, lie at the heart of physically based modeling.

The earliest constraint-based modeling was done by Sutherland in the Sketchpad system, described in Chapter 21. Many constraint-based modeling systems have been developed since, including constraint-based models for human skeletons [ZELT82; KORE82; BADL87], in which connectivity of bones and limits of angular motion on joints are specified, the *dynamic constraint* system of [BARR88], and the *energy constraints* of [WITK87; WILH87]. These fall into two classes: those in which general constraints can be specified, and those that are tailored for particular classes of constraints. In modeling skeletons, for example, point-to-point constraints, in which corresponding points on two bones are required to touch, are common, as are angular limit constraints, in which the angle between bones at a joint is restricted to lie in a certain range. But constraints that specify that the distance between the centers of mass of two objects be minimized are not so likely to occur. Special-purpose constraint systems may admit analytic solutions of a particular class of constraints, whereas the general-purpose systems are more likely to use numerical methods.

In the energy-constraint system we mentioned, for example, constraints are represented by functions that are everywhere nonnegative, and are zero exactly when the constraints are satisfied (these are functions on the set of all possible states of the objects being modeled). These are summed to give a single function, E. A solution to the constraint problem occurs at a state for which E is zero. Since zero is a minimum for E (its component terms are all nonnegative), we can locate such states by starting at any configuration and altering it so as to reduce the value of E. Finding this minimum is done using numerical methods. In the

[7]Typing in numbers is not an adequate compromise, since it may require that the modeler solve a system of equations before typing the numbers.

course of such a process, we may get "stuck" at a local minimum for E, but if we do not, we will eventually reach a global minimum. Such a global minimum is either zero, in which case all constraints are satisfied, or nonzero, in which case some constraints may not be satisfied. By changing the coefficients of the individual constraints in the funtion E, we can stress the importance of some constraints over others. In the case where the system reaches a local minimum, the modeler may start with a different initial configuration, or, in an ideal system, may give a "push" to the configuration to make it move away from the local minimum and toward the global minimum. The sequence of configurations that occurs as the assembly is moving toward a minimum of the function E can be an interesting animation, even though the initial intent was just to model an assembly that satsifies the constraints. In fact, an animation of this sequence of events can be useful in determining characteristics of the function-minimizing algorithm being used.

Further examples of constraint-based modeling are described in Section 20.9 and in Chapter 21.

20.7.2 Modeling Cloth and Flexible Surfaces

Several approaches to modeling cloth and other surfaces have been developed in recent years [WEIL86; WEIL87; TERZ88]. Weil assumes that the cloth is a rectangular weave of threads, each of which is inelastic. The warp and woof positions of a point on the surface provide a coordinate system in which to describe events internal to the cloth, whereas each such point has some 3D location as well. The first assumption in Weil's model is that the cloth is suspended by holding certain points on the cloth at certain positions in 3-space; thus, the "position" of the cloth is initially determined at some finite number of points. The line between any two such points (in the intrinsic coordinate system) is assumed to map onto a catenary curve (which is the shape in which a chain hangs). This determines the positions of several lines in the cloth. Notice that, at a point where two lines cross, the position of the intersection point is overdetermined; Weil simply ignores the lower catenary in any such case. The lines between suspension points on the surface determine regions in the cloth, each of which is filled in with more catenaries. The shape of the cloth has now been determined (at least initially). So far, the structure of the cloth has been ignored: The threads making up the cloth may be stretched, whereas they were supposed to be inelastic. Weil proceeds to a relaxation process that iteratively moves the points in a manner to relieve the "tension" in the threads, by computing the direction vectors between each point and its neighbors. These vectors are multiplied by their own lengths, then are averaged to compute a displacement for the point itself (the multiplication ensures that larger errors have greater effects). This process is iterated until the surface is sufficiently close to satisfying the constraints. A similar method is used to model stiffness of the cloth. Color Plate IV.15 shows the results of this model and the modified model described in [WEIL87].

Terzopoulos and Fleischer [TERZ88] take a more sophisticated approach, and model media more general than cloth as well. They assume that a material is arranged as a grid (possibly 3D, but 2D for cloth), and that adjacent points in the grid are connected by *units* consisting of springs, dashpots (which are like shock absorbers), and plastic slip units. A spring responds to a force by deforming elastically in an amount proportional to the force; when the force goes away, so does the deformation. A dashpot responds to a force by

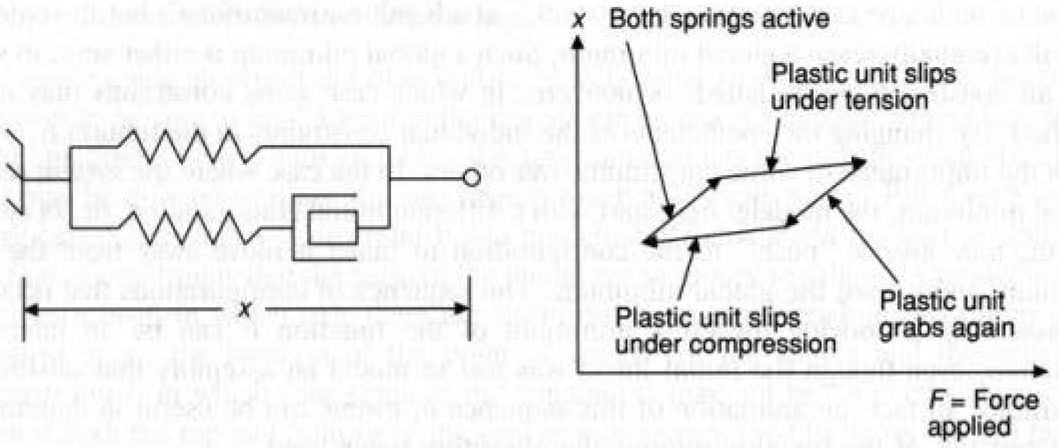

x Both springs active
Plastic unit slips under tension
x
Plastic unit slips under compression
Plastic unit grabs again
F = Force applied

Fig. 20.22 A plastic slip unit connected in series with one spring and in parallel with another creates a unit that responds to a deforming force in a springlike fashion until the force reaches a threshold value. Then the slip unit slips, and retains the deformation until a sufficiently great force restores the unit to its original state.

deforming at a *rate* proportional to the force. Thus, a constant force causes a dashpot to stretch until the force is removed, at which point the dashpot stops deforming. A plastic slip unit responds to a force by doing nothing until the force reaches a certain level, and then slipping freely; these units are best used in combination with other units, such as spring units. Placing a plastic and two spring units in the arrangement shown in Fig. 20.22 creates a unit that stretches gradually (both springs stretching, plastic unit static) until the force on the plastic unit reaches its threshold. At this point, the plastic unit slips and the spring attached to it contracts for a moment, until the other spring takes up some of the load. Thus, at that point, it is as though the system consisted of only the solo spring. Once the applied force is reduced, the lower spring takes up some (compression) load, until the force becomes sufficiently negative to cause the plastic slip unit to slip again.

A grid of such units, subject to laws of physics (modeled globally by rigid-body dynamics but locally by stress and strain rules related to the structure of the units in the material in an internal coordinate system), deforms and stretches. In particular, if threads in the cloth are modeled by plastic slip units, then at some level of tension the units will slip (i.e., the thread will break), and a tear will result. Color Plate IV.16 shows an example of the results.

20.7.3 Modeling Solids

The Terzopoulos and Fleischer model discussed in the previous section can also be used to describe either linear or solid assemblies as collections of points linked by units of varying elasticity or viscosity. Platt and Barr [PLAT88] have done similar work in modeling deformable solids (e.g., putty or gelatin) by combining the solid mechanics underlying such structures with the tools of dynamic constraints. The essence of their work is to set up large collections of differential equations that determine the state of the particle assemblies (or finite-element mesh) at each time, subject to the goals that certain functions (such as energy) be minimized while certain constraints (such as noninterpenetration of objects) are met. Their actual model considers the constraints as objectives to be achieved, along with the minimization of the functions, and thus the constraints are met only approximately. The stronger the effect of each constraint, the more difficult the solution of the differential equations becomes. Despite this numerical difficulty, the results are certainly impressive enough to warrant further research (see Color Plate IV.17).

20.7.4 Modeling Terrain

In another instance of physically based modeling, Kelley and associates [KELL88] extend stream-erosion models from geomorphology. They begin with statistical observations about the distribution of tributaries, the relationships between link length (a *link* is a segment of a system of tributaries between two junctions or between a source and its first junction) and stream gradient, and the mean valley-sidewall slopes. They use these to model the 2D layout of the pattern of streams for a given gross initial terrain, and then to alter the terrain in the vertical dimension so as to compute the finer detail that fits the stream system so constructed. Color Plate IV.18 shows the results for a simple initial drainage system.

20.8 SPECIAL MODELS FOR NATURAL AND SYNTHETIC OBJECTS

A great deal of work has been done on the modeling of natural phenomena by techniques that are not directly related to the underlying causes of the phenomena; modeling of clouds as patterned ellipsoids is a good example. Much work has also gone into the modeling of phenomena that have no specific visual appearance, such as molecules. The examples in this section lie at all points of the range between scientific accuracy and clever manipulations for generating attractive pictures. These models are meant as tools for graphics, rather than as strict scientific visualizations. It is essential that people creating these models understand the underlying phenomena while recognizing the benefits of a good fake.

20.8.1 Waves

Ocean waves were among the earliest natural phenomena modeled in graphics. Ripples resemble sine waves emanating from either a point or a line, and are simple to model as such. If the distance from the eye to the waves is large enough, it may be unnecessary actually to perturb the surface at all, and the entire effect of the ripples can be generated through bump mapping (although one of the first widely shown examples actually raytraced a complete height field [MAX81]). More complex patterns of ripples or waves can be assembled by summing up band-limited noise to make texture maps describing wave trains [PERL85], and then using these to texture map a planar surface. These patterns look best viewed from above, of course, since realistic side views of waves should show the variations in the height of the surface.

Fournier and Reeves [FOUR86], taking a much more sophisticated approach, model the surface of a body of water as a parametric surface rather than as a height field, allowing the possibility of waves curling over. They take into account much of the theory of deep-water waves as well as the effects of underwater topography on surface waves and the refraction and reflection of waves about obstacles (e.g., the way that waves bend around the end of a breakwater). Conversely, in simulating breaking waves, where theoretical knowledge is limited, they provide some clever approximations that generate good results. Their waves are unfortunately somewhat too smooth near the break, and lack a sharp edge between the leading surface and the trailing edge as the wave is about to break. Nonetheless, the results are extremely good (see Color Plate IV.19). Similar work by Peachey [PEAC85] uses a somewhat less complex model; the overall appearance of the waves is not as realistic, but breaking waves are modeled better.

20.8.2 Clouds and Atmosphere

Fog and haze can both be modeled stochastically and then composited onto images. To enhance the realism, we can weight the effect of the fog by the z values of the image onto which it is composited, so that points farther away are more obscured than are those close by. Similarly, fog and haze can be fitted into ray-tracing schemes by attenuating the image by some power of the distance from the eye to the first intersection point (or, even better, for nonuniform fog, by integrating the fog density along the ray to compute an attenuation function). These techniques have some basis in physics, since light is scattered more as it travels farther through fog. Several distinct models of clouds and of atmospheric haze have been developed. Voss [VOSS85] has generated clouds based on fractals, whereas Gardner [GARD84; GARD85] has modeled clouds as textured ellipsoids. Voss's technique is to generate a fractal in 4-space whose fourth coordinate represents a water-vapor density. By allowing local light scattering to vary with the water-vapor density, he generates some realistic clouds (he uses fractal dimensions 3.2 to 3.5).

By contrast, Gardner's method is based completely on the observed shape of clouds—the clouds look like sheets or blobs, and so are modeled as textured planes and ellipsoids. This model consists of a sky plane, in which thin cloud layers reside, ellipsoids (used to model thick clouds, such as cumuli), and a texturing function for each, that handles the varying shading and translucence of the clouds and sky plane.

Gardner creates a particularly simple texture function, akin the ones used by Perlin for solid textures. He defines

$$T(x, y, z) = k\sum_{i=1}^{n}[c_i \sin(f_i x + p_i) + T_0]\sum_{i=1}^{n}[c_i \sin(g_i y + q_i) + T_0],$$

where the c_i are the amplitudes of the texture at various frequencies, the f_i and g_i are frequencies in the x and y directions, respectively, and the p_i and q_i are corresponding phase shifts. This function has different characteristics depending on the values of the various constants. Assigning values with $f_{i+1} = 2f_i$, $g_{i+1} = 2g_i$ and $c_{i+1} = \sqrt{2}/2\ c_i$ produces variations at several different frequencies, with the amplitudes of variations decreasing as the frequencies increase. Notice how similar this is to the fractal models of terrain height: The mountains have large height variations, the boulders on them are smaller, the sharp corners on the boulders are even smaller.

The phase shifts p_i and q_i are used to prevent all the sine waves from being synchronized with one another—that is, to generate randomness (if these are omitted, the texture function has a visible periodicity). For planar texturing, Gardner suggests $p_i = (\pi/2)\sin(g_i\, y/2)$, and similarly for q_i. For ellipsoidal textures, he defines $p_i = (\pi/2)\sin(g_i\, y/2) + \pi \sin(f_i\, z/2)$, which generates phase shifts in all three dimensions, and he finds that using values $0 \le i \le 6$ provides rich textures.

This set of values defines the texture function. The texture function and the sky plane or cloud ellipsoid must now be combined to generate an image. Gardner uses a lighting model of the form

$$I_1 = (1 - s)\, I_d + sI_s,\ I_2 = (1 - t)\, I_1 + tI_t,\ I = (1 - a)\, I_2 + a,$$

where I_d and I_s are the specular and Lambert components of the intensity, computed as in Chapter 16; I_t is $T(x, y, z)$; and a, t, and s determine the fractions of ambient, texture, and specular reflection, respectively. In addition, to get the effect of a cloud rather than of an ellipse with a cloud painted on it, the edges of the cloud must be made translucent. For clouds in the sky plane, regions of the plane must be made translucent. This is done by defining the translucence, V, by the rule

$$V = \begin{cases} 0 & \text{if } I_t \geq V_1 + D, \\ 1 - (I_t - V_1)/D & \text{if } V_1 + D > I_t \geq V_1, \\ 1 & \text{otherwise,} \end{cases}$$

where V_1 and D together determine the range over which the translucence varies from 0 to 1: at $I_t = V_1$ the translucence is 1; at $I_t = V_1 + D$, the translucence has decreased to 0. This is adequate for sky-plane clouds, but for an ellipsoidal cloud, we expect the translucence to be higher at the edges than it is at the center. Gardner determines a function $g()$, which is 1 at the center of the projection of the ellipsoid onto the film plane and 0 on the edge of the projection. With this function, a different translucence function V for ellipsoids can be created, with two different values, V_1 and V_2, determining the translucence threshold at the edge and at the center:

$$V = 1 - (I_t - V_1 - (V_2 - V_1)(1 - g()))/D.$$

This value must be clamped between 0 and 1. Combining the lighting and translucence models gives extremely realistic clouds (especially if they are clustered nicely).

Atmospheric effects with less "substance" than clouds—such as haze, dust, and fog—have been generated using scattering models, typically with the assumption that light is scattered only infrequently within any small volume. Blinn's model of the rings of Saturn [BLIN82a] handled the special case of nearly planar scattering layers made of tiny spherical particles by considering four aspects of scattering:

1. *Phase function*—a tiny spherical particle reflects incident light to a viewer in much the same way as the moon reflects the sun's light to us, which depends on the relative positions of the earth, sun, and moon.
2. *Low albedo*—if the reflectivity of each particle is low, then multiple scattering effects (i.e., the light from reflections bouncing off two or more particles) are insignificant.
3. *Shadowing and masking*—particles more distant from a light source are shadowed by particles in front of them, and light emitted from a particle is attenuated by particles between it and the viewer, and both attenuations are exponential functions of depth into the particle layer.
4. *Transparency*—the transparency of a cloud layer can be described as the probability that a ray passing through it hits no particles, and is an inverse exponential function of the length of the ray contained in the layer.

Max [MAX86] extends this model, by incorporating the shadow volumes developed by Crow [CROW77a]. He computes the light reaching the eye from a surface by taking the light reflected from the surface and adding to it all the light reflected by the intervening atmosphere, just as in Blinn's model; however, some portions of the intervening atmosphere (those in the shadow volumes) reflect no additional light. This generates the appearance of columns of shade (or light) in a reflective atmosphere, like the beams one sees coming through a window in a dusty room. Nishita, Miyawaki, and Nakamae [NISH87] developed a similar technique that handles multiple light sources, light sources with varying intensities (discussed later), and scattering media of varying densities. Their technique is based on determining, for each ray in a ray-tracing renderer, through exactly which volumes of illuminated atmosphere the ray passes, and what illumination comes from each such patch. They incorporate a phase function, different from Blinn's, which is based on an approximation to a more complex scattering theory for relatively small particles, such as dust or fog.

Even further along the same direction is Rushmeier and Torrance's extension of the radiosity model to handle scattering [RUSH87], based on similar theories for modeling heat transfer. In their model, each volume in space (which is divided into small cubes) is dealt with as a separate radiosity element, and not only surface-to-surface interactions, but also surface-to-volume and volume-to-volume interactions, are considered. This can generate extremely complicated systems of equations, but the results are extremely realistic—they constitute some of the most impressive images generated by computer graphics so far (see Color Plate IV.20).

Nishita and Nakamae have also studied the effects of scattering on illumination: A light source that might have been purely directional (such as the light of the sun on the moon's surface) can be diffused by an atmosphere (such as the earth's) and become a scattered source of illumination. An object set on the ground outdoors is illuminated not only by direct sunlight, but also by the light from other regions of the sky (by atmospheric scattering). They model the entire sky as a hemispherical light source with varying intensity, then compute the lighting for an object by integrating over this hemisphere. Color Plate III.22(c) shows an interior scene illuminated by this hemisphere.

20.8.3 Turbulence

The accurate mathematical modeling of turbulence has been of interest for many years, and good fluid-mechanics simulators are now available. These can be used to model turbulence directly, as done by Yeager and Upson [YEAG86], or more empirical models can be used to generate good approximations of the effects of turbulence, as done by Perlin [PERL85]. Perlin's model is particularly simple to replicate in the form of a solid texture (see Section 20.1.2). The turbulence at a point $p = (x, y, z)$ is generated by summing up a collection of Noise() functions of various frequencies; pseudocode for this is given in Fig. 20.23.

The resulting Turbulence() function can be used to generate marble textures by defining Marble(x, y, z) = MarbleColor(sin(x + Turbulence(x, y, z))), where MarbleColor maps values between -1 and 1 into color values for the marble. The x within the sin() is used to generate a smoothly varying function, which is then perturbed by the turbulence function. If MarbleColor has sufficiently high derivatives (i.e., sufficiently great intensity changes) at

```
double Turbulence (double x, double y, double z)
{
    double turb = 0.0;      /* Turbulence is a sum of Noise () terms */
    double s = 1.0;         /* s = scale of the noise; 1 = whole image */

    while (s is greater than pixel size) {
        turb += fabs (s * Noise ( x/s, y/s, z/s) );
        s /= 2.0;
    }
    return turb;
} /* Turbulence */
```

Fig. 20.23 Pseudocode for the turbulence function.

a few points, there will be sharp boundaries between the basic marble and the veins that run through it (see Color Plate IV.21).

20.8.4 Blobby Objects

Molecules are typically portrayed by ball-and-stick models. But the actual physics of molecules reveals that the electron clouds around each atom are not spherical, but rather are distorted by one another's presence (and by other effects as well). To get a better image of surfaces of constant electron density, we must consider the effects of neighboring atoms. In the same way, any collection of items, each of which creates a spherically symmetric scalar field and whose fields combine additively, has *isosurfaces* (surfaces along which the field is constant) modeled not by a collection of overlapping spheres, but rather by some more complex shape. Computing the exact isosurfaces may be impractical, but several good approximations have been made. This was first done independently by Blinn [BLIN82b], in whose system the fields created by each item decayed exponentially with distance and by Nishimura et al. for use in the LINKS project [NISH83a]. Wyvill, McPheeters, and Wyvill [WYVI86] modify Blinn's technique nicely. They model "soft objects" by placing a collection of field sources in space and then computing a field value at each point of space. The field value is the sum of the field values contributed by each source, and the value from each source is a function of distance only. They use a function of distance that decays completely in a finite distance, R, unlike Blinn's exponential decay. Their function,

$$C(r) = \begin{cases} -(\frac{4}{9})r^6/R^6 + (\frac{17}{9})r^4/R^4 - (\frac{22}{9})r^2/R^2 + 1 & \text{if } 0 \le r \le R, \\ 0 & \text{if } R < r, \end{cases}$$

has the properties that $C(0) = 1$, $C(R) = 0$, $C'(0) = 0$, $C'(R) = 0$, and $C(R/2) = \frac{1}{2}$. Figure 20.24 shows a graph of $C(r)$. These properties ensure that blending together surfaces gives smooth joints, and that the field has a finite extent. They compute a number, m, with the property that the volume of the set where $C(r) \ge m$ is exactly one-half the volume of the set where $2C(r) \ge m$. If two sources are placed at the same location and the level-m isosurface is constructed, it therefore has twice the volume of the isosurface for a single source. Thus,

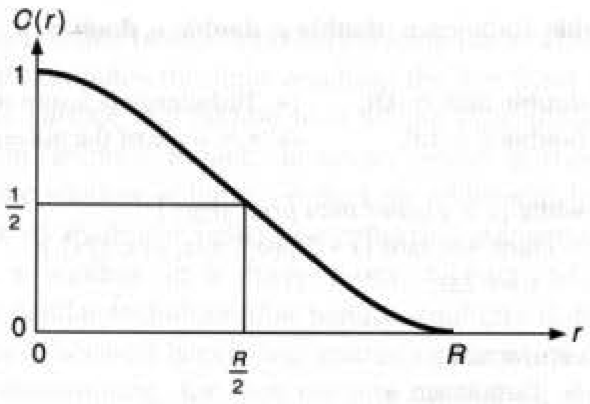


Fig. 20.24 The function $C(r)$.

when soft objects merge, their volumes add. (Notice that, if two sources are far apart, the isosurface may have two separate pieces.)

An isosurface of the field can be computed by an algorithm that resembles the marching-cubes algorithm discussed in Section 20.6, but that is far faster. By evaluating the field at a sequence of grid points along one axis extending from each source, we find a cube with an edge intersected by the isosurface (the edge lies between the last grid point whose value was greater than m and the first whose value is less than m). Because the field value for each source decreases with distance, this collection of cubes (called *seed cubes*) has the property that each piece of the isosurface intersects at least one of them. Thus, by working outward from these seed cubes, we can locate the entire level surface. Additional work can be avoided by flagging each cube that has been processed (these flags, together with various function values, can be stored in a hash table to prevent excessively large data structures). (Another method for computing implicit surfaces is given in [BLOO88].)

The objects modeled with this technique resemble plasticine models [WYVI88] and can be used for molecular modeling, or for modeling droplets of fluid that flow into one another. See Color Plate IV.22 for an example.

20.8.5 Living Things

The plants described in Section 20.4, with their basis in L-grammars, are comparatively simple living things, and the regularity of their form makes them relatively easy to model. Models for shells and coral [KAWA82] and for imaginary living things [KAWA88] have been developed as well. Some biologically simple animals have been modeled recently by physically based techniques, but these models have been adapted to produce good pictures at the cost of some biological realism [MILL88a]. As Miller notes, "One of the great advantages of modeling something like worms is that no one wants to look at them too closely" [MILL88b]. Although the computational costs of biologically and physically realistic models are still prohibitive, the implication of Miller's observation is important—a good eye for appearance may be entirely sufficient to model a peripheral aspect of a scene.

Miller's model of worms and snakes is based on interactions between masses and springs, with muscle contractions modeled as changes in spring tension. The forward motion of the worms and snakes is modeled by adding in directional friction as a force—each segment is allowed to move only forward, and trying to move backward instead draws forward the more tailward segments of the creature. Miller uses bump mapping and pattern mapping to model the appearance of the snakes and worms. He also generates hair for caterpillars by stochastically distributing the bases of the hairs over the body, and distributing the hair ends in a local coordinate system for each hair based on the surface normal and tangent directions to the surface at the base of the hair. He thus can model directional hair. One of his snakes is shown in Color Plate IV.23.

The flight and flocking of birds and schooling of fish have been modeled by Reynolds [REYN87]. The simulation of behavior is so good in this model that the rough appearance of the creatures is only somewhat distracting. As modeling proceeds to the higher genera, the necessity for accuracy increases, since our familiarity with such creatures makes it impossible for us to ignore modeling flaws.

20.8.6 Humans

The modeling of humans is the final frontier. Our ability to recognize and distinguish faces is remarkable; computer graphics images of people must be extremely convincing to satisfy our demands for realism. It is far easier to model a roomful of realistic objects than it is to create one realistic face.

The need for such models has been recognized for some time. Many of the scenes with which we are familiar have people in them, and it would be useful to model these people in a nondistracting way. The eventual goal is to move from this use of people as "extras" in computer-generated movies to their use in "bit parts" and eventually in leading roles. Some progress has been made in this area. Catmull has modeled hands [CATM72] as polygonal objects. The pieces of the hand (fingers, individual joints, etc.) are structured hierarchically, so moving a finger moves all of its joints. Furthermore, each vertex within a joint may be specified either as being a part of the joint itself or in the description of its parent in the hierarchy (the next joint closer to the palm). Thus, when the parent is moved, the shape of the joint may change.

Parke [PARK82], Platt and Badler [PLAT81], and Waters [WATE87] have all developed facial models. Waters models the face as a connected network of polygons whose positions are determined by the actions of several muscles; these muscles are modeled as sheets that can contract. Some of these muscle sheets are anchored to a fixed point in the head, and some are embedded in the skin tissue. The former act by a contraction toward the anchor point, and the latter by contraction within themselves. The facial polygons are modeled by giving their vertices as points on the muscle sheets. Activating a muscle therefore distorts the face into an expression. This arrangement is an improvement on a similar model by Platt and Badler, in which the muscles were modeled as contractable networks of lines, rather than as sheets. Parke also extends this work by allowing control of both the expression and the *conformation* (the characteristics that make one person's face different from another's). In all these models, an essential feature is that the control of the expression is reduced to a few parameters, so that the modeler does not need to place each vertex of each polygon explicitly.

Zeltzer [ZELT82] has done extensive work in modeling the motions of skeletal creatures. The actual structure of the skeletons is comparatively simple (a hierarchical jointed collection of rigid bodies); it is the modeling of the motion that is more complicated. Recent work by Girard [GIRA87] on the motion of legged animals is extremely promising. The modeling of motion is discussed further in Chapter 21.

20.8.7 An Example from the Entertainment Industry

One final example of special-purpose modeling comes from the entertainment industry, in which computer graphics has been widely applied. In the movie *Young Sherlock Holmes*, there is a scene in which a priest hallucinates that he is being attacked by a glass knight that has jumped out from a stained-glass window. The effect would have been quite difficult to produce by conventional animation techniques, as any armatures used to control the knight's motion would have been readily visible through the semi-transparent glass. Computer graphics therefore were used instead.

The series of images in Color Plate IV.24 shows the various techniques involved in modeling the glass. Virtually all of these were implemented by modifying the reflectance function using pattern- and bump-mapping techniques. Part (a) shows a single piece of glass, the shoulder guard, with a color map applied to it, defining its gold stripes. In part (b), an environment map has been applied, showing the church scene behind the piece of glass. In part (c), a bump map has been added, together with an illumination function, and together these modify the environment map of part (b), so that the environment appears refracted through the glass. The shape of the arches is still just barely visible. In part (d), spots of dirt and small bubbles have been added to all the previous effects. In part (e), additional bump maps describe the uneven surfaces on the front of the glass and along the glass's right edge. Part (f) shows a detail of the object. Altogether, three color maps, three bump maps, one transparency map, and one environment map were required to give the glass its realistic appearance. Part (g) shows the complete figure; the shoulder piece is in the upper right.

The pieces of glass were assembled into a hierarchical model and animated using a 3D keyframe animation program. "Spotlights" were strategically placed in the scene so that glints would appear on the knight's sword just as he thrusts it toward the priest. In one shot, the movie camera that photographed the live action was moving, and so the synthetic camera recording the computer-generated action has to move as well, matching the motion of the movie camera exactly. The final effect is most impressive, and in one instance quite startling: When the camera swivels around to show the back side of the knight, we see the same motion, but instead of seeing the back of the knight's head, we see his face again. This gives the motion an uncanny effect, since the limbs seem to bend the wrong way.

20.9 AUTOMATING OBJECT PLACEMENT

Most of this chapter has discussed the creation of objects; some of these objects, such as the terrain molded by erosion, constitute the environment for a scene, but most of them must be *placed* in a scene. Often, a human modeler chooses a location and puts a tree, a flag, or a handkerchief there. When many objects need to be placed in a scene, however, some automation of the process may be necessary. Considering another dimension, we see that the position of a single object at two times may be known, but its position at all intermediate times may need to be determined. This is really the subject of animation, which involves modeling the changes of position and attributes of objects over time, as discussed further in Chapter 21. In situations in which realistic motion of energy-minimizing assemblies is being modeled, we can do the intermediate animation automatically (human motion may be of this form, since humans often try to get from one place to another in the most efficient manner possible). We shall discuss this special case of object placement as well.

Automatic object placement in scenes has not been studied widely. Reeves and Blau [REEV85] discuss a special case in which the trees in a forest are placed automatically by applying a general stochastic rule. The modeler provides a grid size determining spacing between trees and a parameter determining the minimum distance between any two trees, the regions of the horizontal plane to be forested, and the surface contour over these regions, which determines the elevation of the base of the tree. The program then generates at most one tree per grid point, randomly displacing the trees in the x and y directions to avoid giving a gridlike appearance in the final result. If after displacement the new tree would be too close to others, it is eliminated and the algorithm proceeds to the next grid point. This model has some small realism to it: The placement of trees is somewhat random, and forest densities tend to be nearly constant, so that one rarely sees lots of trees all in the same area. Reeves and Blau also let the placement of their trees affect the modeling of the individual trees. The elevation of the tree determines (probabilistically) whether a tree is deciduous (low elevations) or evergreen (higher elevations). This interaction between the terrain and the trees is similar in form to the interacting procedural models of Amburn, Grant, and Whitted [AMBU86], described in Section 20.2, in which characteristics of the terrain influenced the placement of the trees.

Gardner [GARD84] uses a mechanism that encompasses both random displacements and interaction with the terrain, while also forming clusters of objects rather than a regular grid. To determine placements of features in a scene (where to put a tree, for example), he uses a function much like the texture function used in his models for clouds. When this "texture" function is above some critical value, a feature is generated. Using this technique, Gardner generates some exceptionally realistic distributions of features in scenes (Color Plate IV.25).

In all these cases, it is important to avoid both regularity and complete randomness. Much work remains to be done, but it appears that, for such applications, a stochastic control mechanism that can interact with the environment will provide good results.

Another type of automatic object placement is determining the intermediate stages in animations of constrained objects. In some cases, an object's positions in the course of an animation are completely determined by physics; actually computing these position may be very difficult. Witkin and Kass describe a method for determining these intermediate positions [WITK88]. The basic idea is simple: Assuming that an object has been modeled as a physical assembly with various muscles (parts of the assembly that can produce energy) to move other parts, we can describe the states (positions and velocities) of all the parts of the assembly as a function of time (these states include the amount of energy being expended by each muscle at each time, which is related to the muscle tension). This function can be thought of as taking a time value, t, between an initial and a final time, and associating with it a collection of numbers describing the state of the assembly. Thus, the function can be thought of as a path through some high-dimensional space. (The dimension

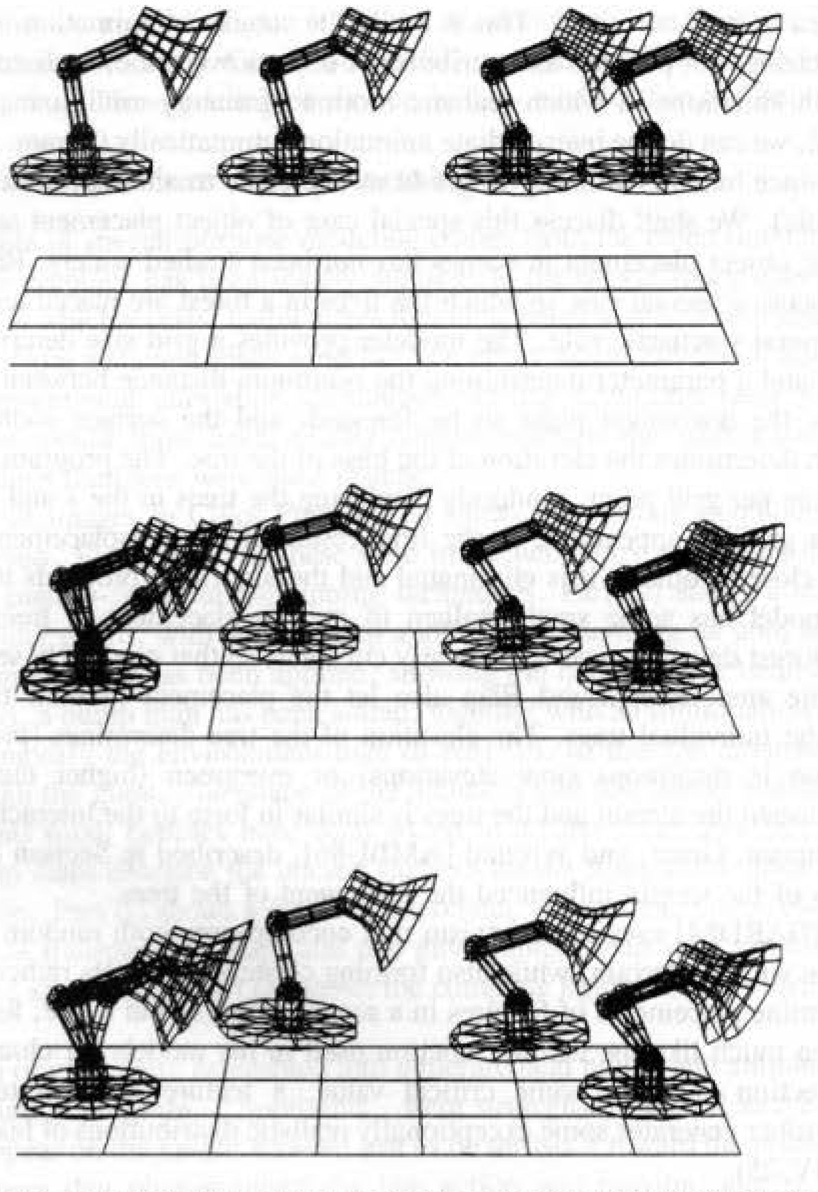
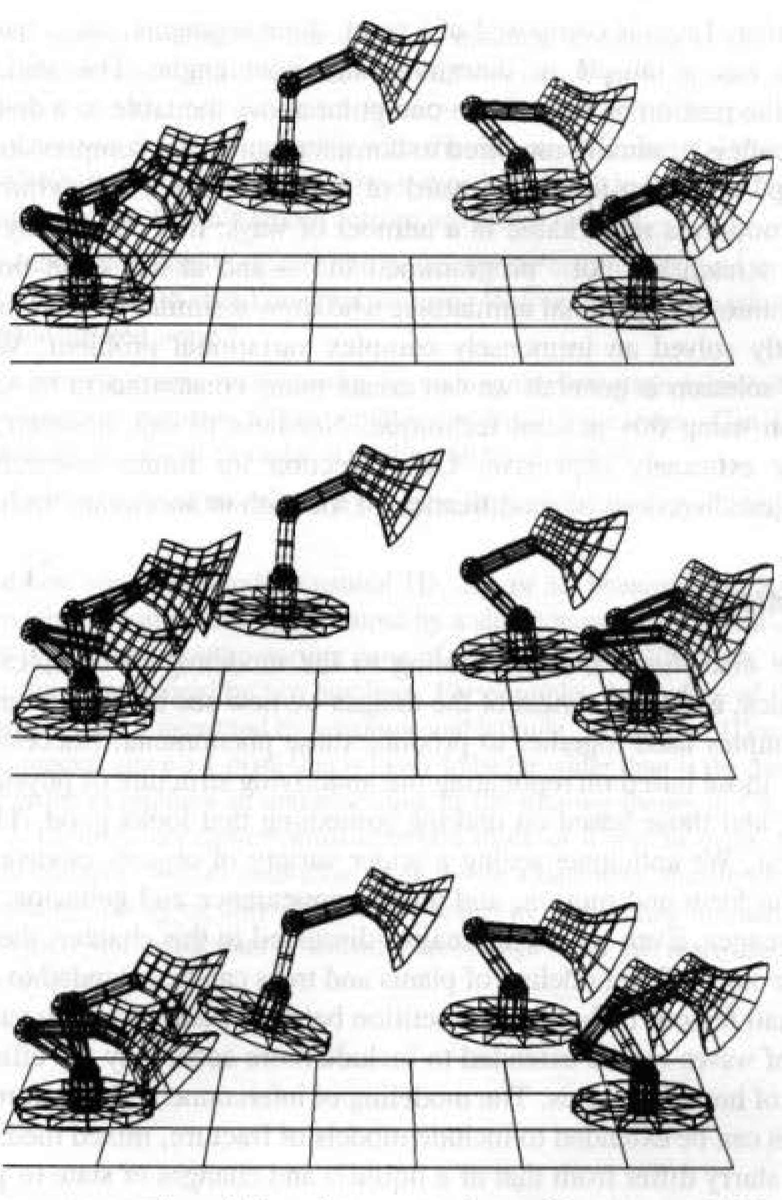


Fig. 20.25 Luxo Jr. is asked to jump from one position on the table to another. An initial path is specified in which Luxo moves above the table. Iterations of a variational technique lead Luxo to find a crouch–stretch–followthrough approach to the motion

of the space is about twice the number of degrees of freedom in the assembly.) Among the collection of all such functions, there are some whose total energy expenditure is lower than that of others. There also are some whose initial position for the parts is the desired initial position and whose ending position is the desired ending position, and we can measure how far a path is from satisfying these conditions. Some functions will represent physically possible sequences of events (e.g., in some paths, the momentum of each part will be, in the absence of external forces, proportional to the derivative of the part's position).

To compute the path of the object over time, we now take an approach called *variational calculus*, which is similar to gradient methods used for finding minima of ordinary real-valued functions. We start with any path and alter it slightly by moving certain points on the path in some direction. We now determine whether the path is closer to a good



that minimizes energy and satisfies the constraints. (Courtesy of Michael Kass and Andrew Witkin.)

path (where "good" means "low energy expenditure," "laws of physics satisfied," and "starting and ending conditions satisfied") or is farther away. If it is closer, it becomes our new path, and we repeat the operation. If it is farther away, we alter the original path by exactly the opposite perturbations, and let this be our new path. As we iterate this process, we get closer and closer to a low-energy path that satisfies the constraints. Once we reach a path satisfying the constraints, we can continue the process until we reach the lowest-energy path possible. It turns out that the best alteration to the path at any time can be determined, so we approach a minimum-energy path very quickly.

The actual mechanism by which this alteration is effected is extremely complex, but the underlying idea is simple. Figure 20.25 is an example of the method in action; a model of "Luxo Jr." from a Pixar animation [PIXA86], is supposed to jump from one position on the table to another. Luxo is composed of a head, three segments, and a base. Each joint is frictionless and has a muscle to determine the joint angle. The initial path for the computation is the motion of Luxo from one point above the table to a distant point above the table. This path is gradually modified to consist of an initial compression of the body, a stretch and leap, a pulling-up-and-forward of the base and a followthrough to prevent toppling. This motion is remarkable in a number of ways: it is completely synthetic—the crouch and the stretch are not "programmed in"—and at the same time it shows the remarkable intuition of traditional animators, who drew a similar motion for the object and thereby implicitly solved an immensely complex variational problem. Witkin and Kass remark that the solution is general; we can create many constraints to be satisfied and still find the solution using this general technique. Needless to say, however, the method is computationally extremely expensive. One direction for future research is enabling a modeler to suggest directions of modification of the path to accelerate finding the solution.

20.10 SUMMARY

More and more disciplines are contributing to the modeling of complex phenomena in computer graphics, and the richness of the images we now see being generated is due to the variety of techniques used together to produce these phenomena. Successful models still have two forms: those based on replicating the underlying structure or physics of the objects being modeled, and those based on making something that looks good. The second often precedes the first. We anticipate seeing a wider variety of objects modeled in the future. Modeling human form and motion, and animal appearance and behavior, are particularly significant challenges. Even within the realms discussed in this chapter, there is substantial room for further work. The modeling of plants and trees can be extended to modeling of the ecology of a small region, including competition between plant forms for various resources. The modeling of waves can be extended to include more accurately the effects of wind and the appearance of breaking waves. The modeling of interconnected structures such as cloth, clay, and liquids can be extended to include models of fracture, mixed media (how does the movement of a slurry differ from that of a liquid?) and changes of state (e.g., melting ice). We look forward to seeing these new models and their successors, and eagerly await the day when computer-synthesized scenes are routinely mistaken for photographs. Although this ability to fool the eye may not always be the final goal in modeling, it is a good measure of the power available to computer graphics: If we can model reality, we can model anything.

EXERCISES

20.1 Show that the Bernstein polynomials $Q_{n;i}(t)$ used in the Sederberg-Parry deformation technique satisfy $\sum_{i=0}^{n} Q_{n;i}(t) = 1$ by using the binomial theorem, which says that

$$\sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i} = (a + b)^n.$$

20.2 Implement the Perlin texturing model on your computer. Can you fine tune the model to compute textures at only the points of a specific surface (such as a sphere), so that you can generate a real-time texture editor? What are the difficulties in generating multifrequency noise in real time? Try bump mapping the normal vectors by the dNoise() function described in the text—that is, by adjusting the normals by a rule of the form *newNormal* = *oldNormal* + dNoise (*currentPoint*). Let $f(x)$ be a function that is 0 for $x < a$ and is 1 for $x > b$, where a and b are positive numbers with $a < b$. If you assume you do bump mapping with the rule *newNormal* = *oldNormal* + f (Noise (*currentPoint*)) * dNoise (*currentPoint*), what do you expect the result to look like for various values of a and b? Try to replicate the Perlin's stucco texture using this method.

20.3 Perlin uses cubic interpolation to generate values for the coefficients used in computing noise. Can you think of an easy way to speed this process using look-up tables? Can you think of a quicker way to generate band-limited noise?

20.4 Implement a particle system for fireworks, by making the first-stage particle follow a parabolic trajectory, and subsequent particles follow smaller parabolic trajectories. Can you combine this system with the soft-object model to make exploding blobs of water?

20.5 Implement Gardner's cloud model and try to tune the parameters to give good-looking cumulus clouds.

20.6 Think about how you could model a natural 1D, 2D, or 3D phenomenon. By a 1D object, we mean an object on which position can be measured by a single number, such as a curve in the plane (the single number is distance from the starting point of the curve). By a 2D object, we mean an object on which position can be measured by two numbers. For example, the surface of the sphere is a 2D object because position can be measured by longitude and latitude. Notice that 1D phenomena such as hair are difficult to render, since a 1-pixel line is likely to be far wider than is the desired image of the hair. Solving this problem requires an understanding of the filtering theory in Chapters 14 and 17. Some interesting 2D objects are flower petals (can you think of a way to make a movie of a rose unfolding?), ribbed surfaces (such as umbrellas, or skin over a skeleton), and ribbons (can you model the shape of a ribbon by specifying only where a few points lie, and letting mechanics determine the rest?). Some 3D objects you might want to consider are sponge (or is this really fractal?), translucent glass, and mother-of-pearl.

21 Animation

To *animate* is, literally, to bring to life. Although people often think of animation as synonymous with motion, it covers all changes that have a visual effect. It thus includes the time-varying position (*motion dynamics*), shape, color, transparency, structure, and texture of an object (*update dynamics*), and changes in lighting, camera position, orientation, and focus, and even changes of rendering technique.

Animation is used widely in the entertainment industry, and is also being applied in education, in industrial applications such as control systems and heads-up displays and flight simulators for aircraft, and in scientific research. The scientific applications of computer graphics, and especially of animation, have come to be grouped under the heading *scientific visualization*. Visualization is more than the mere application of graphics to science and engineering, however; it can involve other disciplines, such as signal processing, computational geometry, and database theory. Often, the animations in scientific visualization are generated from *simulations* of scientific phenomena. The results of the similations may be large datasets representing 2D or 3D data (e.g., in the case of fluid-flow simulations); these data are converted into images that then constitute the animation. At the other extreme, the simulation may generate positions and locations of physical objects, which must then be rendered in some form to generate the animation. This happens, for example, in chemical simulations, where the positions and orientations of the various atoms in a reaction may be generated by simulation, but the animation may show a ball-and-stick view of each molecule, or may show overlapping smoothly shaded spheres representing each atom. In some cases, the simulation program will contain an embedded animation language, so that the simulation and animation processes are simultaneous.

If some aspect of an animation changes too quickly relative to the number of animated frames displayed per second, *temporal aliasing* occurs. Examples of this are wagon wheels that apparently turn backward and the jerky motion of objects that move through a large field of view in a short time. Videotape is shown at 30 frames per second (fps), and photographic film speed is typically 24 fps, and both of these provide adequate results for many applications. Of course, to take advantage of these rates, we must create a new image for each videotape or film frame. If, instead, the animator records each image on two videotape frames, the result will be an effective 15 fps, and the motion will appear jerkier.[1]

Some of the animation techniques described here have been partially or completely implemented in hardware. Architectures supporting basic animation in real time are essential for building flight simulators and other real-time control systems; some of these architectures were discussed in Chapter 18.

Traditional animation (i.e., noncomputer animation) is a discipline in itself, and we do not discuss all its aspects. Here, we concentrate on the basic concepts of computer-based animation, and also describe some state-of-the-art systems. We begin by discussing conventional animation and the ways in which computers have been used to assist in its creation. We then move on to animation produced principally by computer. Since much of this is 3D animation, many of the techniques from traditional 2D character animation no longer apply directly. Also, controlling the course of an animation is more difficult when the animator is not drawing the animation directly: it is often more difficult to describe *how* to do something than it is to do that action directly. Thus, after describing various animation languages, we examine several animation control techniques. We conclude by discussing a few general rules for animation, and problems peculiar to animation.

21.1 CONVENTIONAL AND COMPUTER-ASSISTED ANIMATION

21.1.1 Conventional Animation

A conventional animation is created in a fairly fixed sequence: The story for the animation is written (or perhaps merely conceived), then a *storyboard* is laid out. A storyboard is an animation in outline form—a high-level sequence of sketches showing the structure and ideas of the animation. Next, the soundtrack (if any) is recorded, a detailed layout is produced (with a drawing for every scene in the animation), and the soundtrack is read—that is, the instants at which significant sounds occur are recorded in order. The detailed layout and the soundtrack are then correlated.[2] Next, certain *key frames* of the animation are drawn—these are the frames in which the entities being animated are at extreme or characteristic positions, from which their intermediate positions can be inferred. The intermediate frames are then filled in (this is called *inbetweening*), and a trial film is made (a *pencil test*). The pencil-test frames are then transferred to *cels* (sheets of acetate

[1]This lets the animator generate only half as many frames, however. In some applications, the time savings may be worth the tradeoff in quality.

[2]The order described here is from conventional studio cartoon animation. In fine-arts animation, the soundtrack may be recorded last; in computer-assisted animation, the process may involve many iterations.

film), either by hand copying in ink or by photocopying directly onto the cels. In multiplane animation, multiple layers of cels are used, some for background that remains constant (except perhaps for a translation), and some for foreground characters that change over time. The cels are colored in or painted, and are assembled into the correct sequence; then, they are filmed. The people producing the animation have quite distinct roles: some design the sequence, others draw key frames, others are strictly inbetweeners, and others work only on painting the final cels. Because of the use of key frames and inbetweening, this type of animation is called *key-frame animation*. The name is also applied to computer-based systems that mimic this process.

The organizational process of an animation is described [CATM78a] by its storyboard; by a *route sheet*, which describes each scene and the people responsible for the various aspects of producing the scene; and by the *exposure sheet*, which is an immensely detailed description of the animation. The exposure sheet has one line of information for each frame of the animation, describing the dialogue, the order of all the figures in the frame, the choice of background, and the camera position within the frame. This level of organization detail is essential in producing a coherent animation. For further information on conventional animation, see [LAYB79; HALA68; HALA73].

The entire process of producing an animation is supposed to be sequential, but is often (especially when done with computers) iterative: the available sound effects may cause the storyboard to be modified slightly, the eventual look of the animation may require that some sequences be expanded, in turn requiring new sound-track segments, and so on.

21.1.2 Computer Assistance

Many stages of conventional animation seem ideally suited to computer assistance, especially inbetweening and coloring, which can be done using the seed-fill techniques described in Section 19.5.2. Before the computer can be used, however, the drawings must be digitized. This can be done by using optical scanning, by tracing the drawings with a data tablet, or by producing the original drawings with a drawing program in the first place. The drawings may need to be postprocessed (e.g., filtered) to clean up any glitches arising from the input process (especially optical scanning), and to smooth the contours somewhat. The composition stage, in which foreground and background figures are combined to generate the individual frames for the final animation, can be done with the image-composition techniques described in Section 17.6.

By placing several small low-resolution frames of an animation in a rectangular array, the equivalent of a pencil test can be generated using the pan-zoom feature available in some frame buffers. The frame buffer can take a particular portion of such an image (the portion consiting of one low-resolution frame), move it to the center of the screen (*panning*), and then enlarge it to fill the entire screen (*zooming*).[3] This process can be repeated on the

[3]The panning and zooming are actually effected by changing the values in frame-buffer registers. One set of registers determines which pixel in the frame-buffer memory corresponds to the upper-left corner of the screen, and another set of registers determines the pixel-replication factors—how many times each pixel is replicated in the horizontal and vertical direction. By adjusting the values in these registers, the user can display each of the frames in sequence, pixel-replicated to fill the entire screen.

several frames of the animation stored in the single image; if done fast enough, it gives the effect of continuity. Since each frame of the animation is reduced to a very small part of the total image (typically one twenty-fifth or one thirty-sixth), and is then expanded to fill the screen, this process effectively lowers the display device's resolution. Nonetheless, these low-resolution sequences can be helpful in giving a sense of an animation, thus acting as a kind of pencil test.

21.1.3 Interpolation

The process of inbetweening is amenable to computer-based methods as well, but many problems arise. Although a human inbetweener can perceive the circumstances of the object being interpolated (is it a falling ball or a rolling ball?), a computer-based system is typically given only the starting and ending positions. The easiest interpolation in such a situation is *linear interpolation*: Given the values, v_s and v_e, of some attribute (position, color, size) in the starting and ending frames, the value v_t at intermediate frames is $v_t = (1 - t)v_s + t\, v_e$; as the value t ranges from 0 to 1, the value of v_t varies smoothly from v_s to v_e. Linear interpolation (sometimes called *lerping*—Linear intERPolation), although adequate in some circumstances, has many limitations. For instance, if lerping is used to compute intermediate positions of a ball that is thrown in the air using the sequence of three key frames shown in Fig. 21.1 (a), the resulting track of the ball shown in Fig. 21.1(b) is entirely unrealistic. Particularly problematic is the sharp corner at the zenith of the trajectory: Although lerping generates continuous motion, it does not generate continuous derivatives, so there may be abrupt changes in velocity when lerping is used to interpolate positions. Even if the positions of the ball in the three key frames all lie in a line, if the distance between the second and third is greater than that between the first and second, then lerping causes a discontinuity in speed at the second key frame. Thus, lerping generates derivative discontinuities in time as well as in space (the time discontinuities are measured by the parametric continuity described in Chapter 11).

Because of these drawbacks of lerping, splines have been used instead to smooth out interpolation between key frames. Splines can be used to vary any parameter smoothly as a function of time. The splines need not be polynomials.[4] For example, to get smooth initiation and termination of changes (called *slow-in* and *slow-out*) and fairly constant rates of change in between, we could use a function such as $f(t)$ in Fig. 21.2. A value can be interpolated by setting $v_t = (1 - f(t))v_s + f(t)v_e$. Since the slope of f is zero at both $t = 0$ and $t = 1$, the change in v begins and ends smoothly. Since the slope of f is constant in the middle of its range, the rate of change of v is constant in the middle time period.

Splines can make individual points (or individual objects) move smoothly in space and time, but this by no means solves the inbetweening problem. Inbetweening also involves interpolating the shapes of objects in the intermediate frames. Of course, we could describe a spline path for the motion of each point of the animation in each frame, but splines give the smoothest motion when they have few control points, in both space and time. Thus, it is preferable to specify the positions of only a few points at only a few times, and somehow to

[4]This is an extension of the notion of spline introduced in Chapter 11, where a spline was defined to be a piecewise cubic curve. Here we use the term in the more general sense of any curve used to approximate a set of control points.

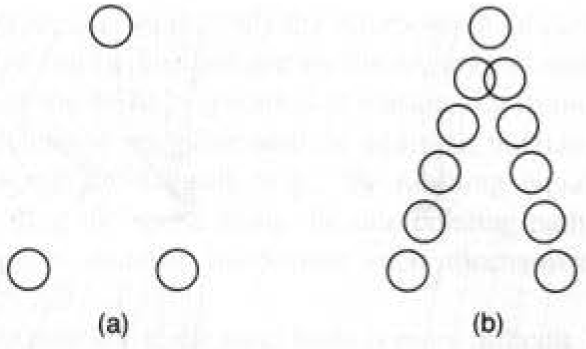


Fig. 21.1 Linear interpolation of the motion of a ball generates unrealistic results. (a) Three key-frame positions for the ball. (b) The resulting interpolated positions.

extend the spline interpolation over intermediate points and times. At least one special case deserves mention: A figure drawn as a polyline can be interpolated between key frames by interpolating each vertex of the polyline from its starting to ending position. As long as the key frames do not differ too much, this is adequate (for examples where this fails, see Exercise 21.1).

Several approaches to this have been developed. Burtnyk and Wein [BURT76] made a *skeleton* for a motion by choosing a polygonal arc describing the basic shape of a 2D figure or portion of a figure, and a neighborhood of this arc (see Fig. 21.3). The figure is represented in a coordinate system based on this skeleton. They then specify the thickness of the arc and positions of the vertices at subsequent key frames and redraw the figure in a new coordinate system based on the deformed arc. Inbetweening is done by interpolating the characteristics of the skeleton between the key frames. (A similar technique can be developed for 3D, using the trivariate Bernstein polynomial deformations or the heirarchical B-splines described in Chapter 20.)

Reeves [REEV81] designed a method in which the intermediate trajectories of particular points on the figures in successive key frames are determined by hand-drawn paths (marked by the animator to indicate constant time intervals). A region bounded by two such *moving-points paths* and an arc of the figure in each of the two key frames

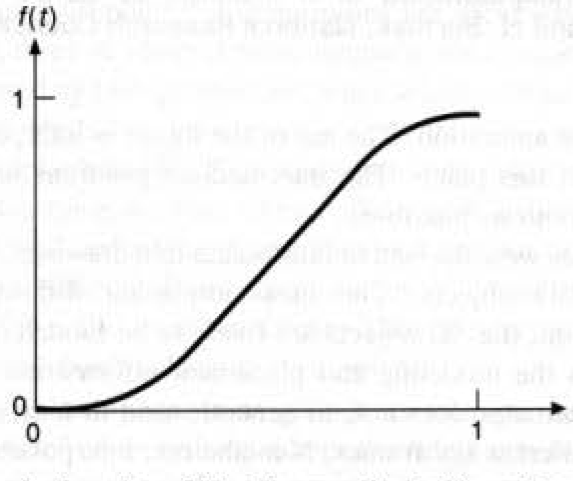


Fig. 21.2 The graph of a function $f(t)$ with zero derivative at its endpoints and constant derivative in its middle section.

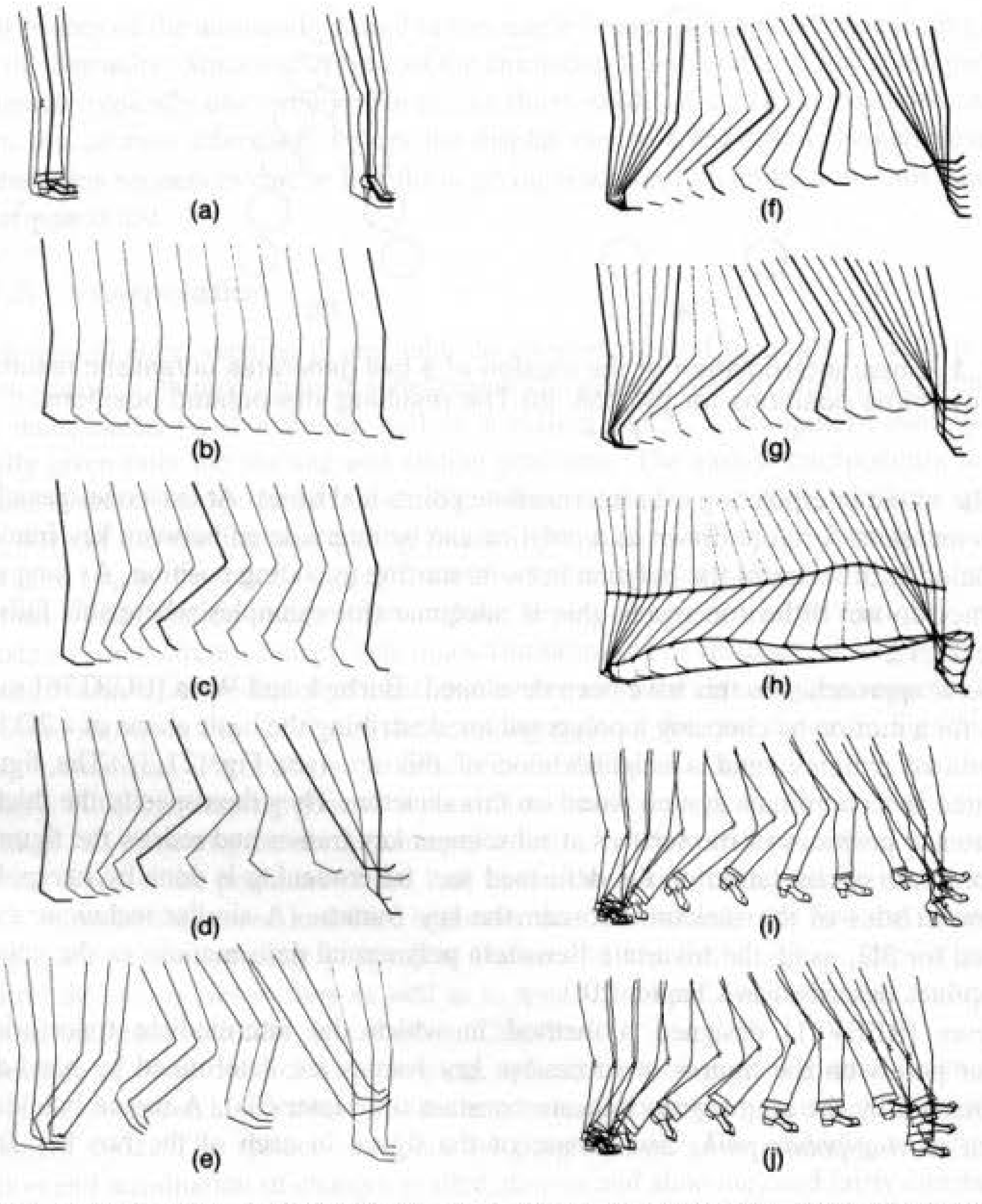


Fig. 21.3 Use of a neighborhood of a skeleton to define interpolated shapes. (Courtesy of M. Wein and N. Burtnyk, National Research Council of Canada.)

determines a *patch* of the animation. The arc of the figure is interpolated by computing its intermediate positions in this patch. The intermediate positions are determined so as to make the motion as smooth as possible.

Both these techniques were devised to interpolate line drawings, but the same problems arise in interpolating 3D objects. The most important difference is that, in most computer-based animation, the 3D objects are likely to be modeled explicitly, rather than drawn in outlines. Thus the modeling and placement information is available for use in interpolation, and the animator does not, in general, need to indicate which points on the objects correspond in different key frames. Nonetheless, interpolation between key frames is a difficult problem.

For the time being, let us consider only the interpolation of the position and orientation of a rigid body. Position can be interpolated by the techniques used in 2D animation: The position of the center of the body is specified at certain key frames, and the intermediate positions are interpolated by some spline path. In addition, the rate at which the spline path is traversed may be specified as well (e.g., by marking equal-time intervals on the trajectory, or by specifying the speed along the interpolating path as a function of time). Many different animation systems implement such mechanisms; some of these are discussed in Section 21.2.3.

Interpolating the orientation of the rigid body is more difficult. In fact, even specifying the orientation is not easy. If we specify orientations by amounts of rotation about the three principal axes (called *Euler angles*), then the order of specification is important. For example, if a book with its spine facing left is rotated 90° about the x axis and then $-90°$ about the y axis, its spine will face you, whereas if the rotations are done in the opposite order, its spine will face down. A subtle consequence of this is that interpolating Euler angles leads to unnatural interpolations of rotations: A rotation of 90° about the z axis and then 90° about the y axis has the effect of a 120° rotation about the axis (1, 1, 1). But rotating 30° about the z axis and 30° about the y axis does not give a rotation of 40° about the axis (1, 1, 1)—it gives approximately a 42° rotation about the axis (1, 0.3, 1)!

The set of all possible rotations fits naturally into a coherent algebraic structure, the *quaternions* [HAMI53]. The rotations are exactly the *unit quaternions*, which are symbols of the form $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, where a, b, c, and d are real numbers satisfying $a^2 + b^2 + c^2 + d^2 = 1$; quaternions are multiplied using the distributive law and the rules $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$, $\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$, and $\mathbf{ki} = \mathbf{j} = -\mathbf{ik}$. Rotation by angle ϕ about the unit vector $[b \quad c \quad d]^t$ corresponds to the quaternion $\cos \phi/2 + b \sin \phi/2\, \mathbf{i} + c \sin \phi/2\, \mathbf{j} + d \sin \phi/2\, \mathbf{k}$. Under this correspondence, performing successive rotations corresponds to multiplying quaternions. The inverse correspondence is described in Exercise 21.7.

Since unit quaternions satisfy the condition $a^2 + b^2 + c^2 + d^2 = 1$, they can be thought of as points on the unit sphere in 4D. To interpolate between two quaternions, we simply follow the shortest path between them on this sphere (a *great arc*). This spherical linear interpolation (called *slerp*) is a natural generalization of linear interpolation. Shoemake [SHOE85] proposed the use of quaternions for interpolation in graphics, and developed generalizations of spline interpolants for quaternions.

The compactness and simplicity of quaternions are great advantages, but difficulties arise with them as well, three of which deserve mention. First, each orientation of an object can actually be represented by two quaternions, since rotation about the axis $\mathbf{v}$ by an angle ϕ is the same as rotation about $-\mathbf{v}$ by the angle $-\phi$; the corresponding quaternions are antipodal points on the sphere in 4D. Thus to go from one orientation to another, we may interpolate from one quaternion to either of two others; ordinarily we choose the shorter of the two great arcs. Second, orientations and rotations are not exactly the same thing: a rotation by 360° is very different from a rotation by 0° in an animation, but the same quaternion $(1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k})$ represents both. Thus specifying multiple rotations with quaternions requires many intermediate control points.

The third difficulty is that quaternions provide an *isotropic* method for rotation—the interpolation is independent of everything except the relation between the initial and final rotations. This is ideal for interpolating positions of tumbling bodies, but not for interpolating the orientation of a camera in a scene: Humans strongly prefer cameras to be held upright (i.e., the horizontal axis of the film plane should lie in the (x, z) plane), and are profoundly disturbed by tilted cameras. Quaternions have no such preferences, and therefore should not be used for camera interpolation. The lack of an adequate method for interpolating complex camera motion has led to many computer animations having static cameras or very limited camera motion.

21.1.4 Simple Animation Effects

In this section, we describe a few simple computer-animation tricks that can all be done in real time. These were some of the first techniques developed, and they are therefore hardware-oriented.

In Section 4.4.1, we discussed the use of color look-up tables (luts) in a frame buffer and the process of double-buffering; and in Section 17.6, we described image compositing by color-table manipulations. Recall that lut animation is generated by manipulating the lut. The simplest method is to cycle the colors in the lut (to replace color i with color $i - 1$ mod n, where n is the number of colors in the table), thus changing the colors of the various pieces of the image. Figure 21.4 shows a source, a sink, and a pipe going between them. Each piece of the figure is labeled with its lut index. The lut is shown at the right. By cycling colors 1 through 5, we can generate an animation of material flowing through the pipe.

Using this lut animation is a great deal faster than sending an entire new pixmap to the frame buffer for each frame. Assuming 8 color bits per pixel in a 640 by 512 frame buffer, a single image contains 320 KB of information. Transferring a new image to the frame buffer every thirtieth of a second requires a bandwidth of over 9 MB per second, which is well beyond the capacity of most small computers. On the other hand, new values for the lut can be sent very rapidly, since luts are typically on the order of a few hundred to a few thousand bytes.

Lut animation tends to look jerky, since the colors change suddenly. This effect can be softened somewhat by taking a color to be made visible and changing its lut entry gradually over several frames from the background color to its new color, and then similarly fading it out as the next lut entry is being faded in. Details of this and other tricks are given by Shoup [SHOU79].

Lut animation can be combined with the pan-zoom movie technique described

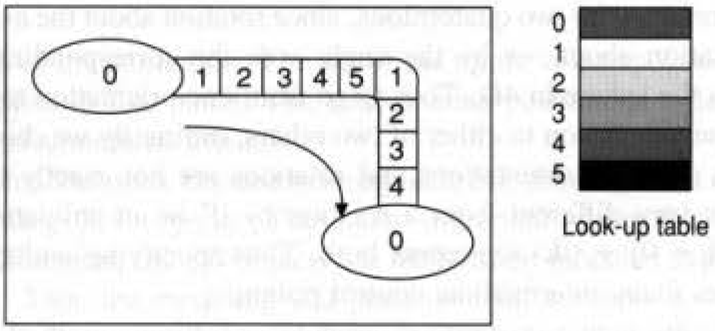


Fig. 21.4 The look-up–table entries can be cycled to give the impression of flow through the pipe.

previously to make longer pan-zoom movies with less color resolution. To make a very long two-color pan-zoom movie on a frame buffer with eight planes of memory, for example, we can generate 200 frames of an animation, each at one-twenty-fifth of full screen resolution. Frames 1 through 25 are arranged in a single image to be used for a pan-zoom movie. The same is done with frames 26 through 50, and so on up to frames 176 through 200, giving a total of eight bitmaps. These are combined, on a pixel-by-pixel basis, into a single 8-bit-deep image, which is then downloaded to the frame buffer. We make all the lut entries black except entry 00000001, which we make white. We then run a 25-frame pan-zoom movie and see the first 25 images of the animation. Then, we set entry 00000001 to black and entry 00000010 to white. Running another 25-frame pan-zoom movie shows us the next 25 images. Continuing in this fashion, we see the full 200-frame animation. By allocating several planes to each image, we can generate shorter pan-zoom movies with additional bits of color.

Finally, let's look at the hardware-based animation technique called *sprites*. A sprite is a small rectangular region of memory that is mixed with the rest of the frame-buffer memory at the video level. The location of the sprite at any time is specified in registers in the frame buffer, so altering the values in these registers causes the sprite to move. The sprites may hide the frame-buffer values at each pixel, or may be blended with them. We can use sprites to implement cursors in frame buffers, and also to generate animations by moving the sprite (or sprites) around on top of a background image. Some frame buffers have been designed to allow several sprites with different priorities, so that some sprites can be "on top of" others.

One of the most popular uses of sprites is in video games, where the animation in the game may consist almost entirely of sprites moving over a fixed background. Since the location and size of each sprite are stored in registers, it is easy to check for collisions between sprites, which further enhances the use of sprites in this application.

21.2 ANIMATION LANGUAGES

There are many different languages for describing animation, and new ones are constantly being developed. They fall into three categories: linear-list notations, general-purpose languages with embedded animation directives, and graphical languages. Here, we briefly describe each type of language and give examples. Many animation languages are mingled with modeling languages, so the descriptions of the objects in an animation and of the animations of these objects are done at the same time.

21.2.1 Linear-List Notations

In linear-list notations for animation such as the one presented in [CATM72], each event in the animation is described by a starting and ending frame number and an action that is to take place (the *event*). The actions typically take parameters, so a statement such as

```
42, 53,B ROTATE "PALM", 1, 30
```

means "between frames 42 and 53, rotate the object called PALM about axis 1 by 30 degrees, determining the amount of rotation at each frame from table B." Thus, the actions are given interpolation methods to use (in this case a table of values) and objects to act on as well. Since the statements describe individual actions and have frame values associated with them, their order is, for the most part, irrelevant. If two actions are applied to the same object at the same time, however, the order may matter: rotating 90° in x and then 90° in y is different from rotating 90° in y and then 90° in x.

Many other linear-list notations have been developed, and many notations are supersets of the basic linear-list idea. Scefo (SCEne FOrmat) [STRA88], for example, has some aspects of linear-list notation, but also includes a notion of groups and object hierarchy and supports abstractions of changes (called *actions*) and some higher-level programming-language constructs (variables, flow of control, and expression evaluation) distinguishing it from a simple linear list. Scefo also supports a model of animation that differs from many animation languages in that it is renderer-independent. A Scefo script describes only an animation; the individual objects in the script can be rendered with any renderer at all, and new renderers can easily be added to the animation system of which Scefo is the core.

21.2.2 General-Purpose Languages

Another way to describe animations is to embed animation capability within a general-purpose programming language [REYN82; SYMB85; MAGN85]. The values of variables in the language can be used as parameters to whatever routines actually generate animations, so the high-level language can actually be used to generate simulations that then generate animations as a side effect. Such languages have great potential (e.g., they can certainly do everything that linear-list notations do), but most of them require considerable programming expertise on the part of the user.

Such systems can use the constructs of the surrounding language to create concise routines that have complex effects. Of course, these can sometimes be cryptic. ASAS [REYN82] is an example of such a language. It is built on top of LISP, and its primitive entities include vectors, colors, polygons, solids (collections of polygons), groups (collections of objects), points of view, subworlds, and lights. A point of view consists of a location and an orientation for an object or a camera (hence, it corresponds to the cumulative transformation matrix of an object in PHIGS). Subworlds are entities associated with a point of view; the point of view can be used to manipulate the entities in the subworld in relation to the rest of the objects in the animation.

ASAS also includes a wide range of geometric transformations that operate on objects; they take an object as an argument and return a value that is a transformed copy of the object. These transformations include *up*, *down*, *left*, *right*, *zoom-in*, *zoom-out*, *forward*, and *backward*. Here is an ASAS program fragment, describing an animated sequence in which an object called my-cube is spun while the camera pans. Anything following a semicolon is a comment. This fragment is evaluated at each frame in order to generate the entire sequence.

```
(grasp my-cube)           ; The cube becomes the current object
(cw 0.05)                 ; Spin it clockwise by a small amount
(grasp camera)            ; Make the camera the current object
(right panning-speed)     ; Move it to the right
```

The advantage of ASAS over linear-list notations is the ability to generate procedural objects and animations within the language. This ability comes at the cost of increased skill required of the animator, who must be an able programmer. Scefo lies in the middle ground, providing some flow-of-control constructs, and the ability to bind dynamically with routines written in a high-level language, while being simple enough for nonprogrammers to learn and use readily.

21.2.3 Graphical Languages

One problem with the textual languages we have described is that it is difficult for an animator to see what will take place in an animation just by looking at the script. Of course, this should not be surprising, since the script is a program, and to the extent that the program's language allows high-level constructs, it encodes complex events in compact form. If a real-time previewer for the animation language is available, this is not a problem; unfortunately the production of real-time animations is still beyond the power of most hardware.

Graphical animation languages describe animation in a more visual way. These languages are used for expressing, editing, and comprehending the simultaneous changes taking place in an animation. The principal notion in such languages is substitution of a visual paradigm for a textual one: rather than explicitly writing out descriptions of actions, the animator provides a picture of the action. Some of the earliest work in this area was done by Baecker [BAEC69], who introduced the notion of P-curves in the GENESYS animation system. A P-curve is a parametric representation of the motion (or any other attribute) of an object or assembly of objects within a scene. The animator describes an object path of motion by graphically specifying its coordinates as a function of time (just as splines do, where functions $X(t)$, $Y(t)$, and $Z(t)$ specify the 3D location of a point on a curve as a function of an independent variable). Figure 21.5(a) shows a motion path in the plane; Fig. 21.5(b) of that figure shows the path's x and y components as functions of time. Notice that the curves in part (b)) uniquely determine the curve in part (a), but the opposite is not true: One can traverse the path in part (a) at different speeds. By marking the path in part (a) to indicate constant time steps, we can convey the time dependence of the path, as shown in part (c), which is what Baecker calls a P-curve. Note that part (c) can be constructed as shown in part (d) by graphing the x and y components as functions of t, on coordinate systems that are rotated 90° from each other, and then drawing lines to connect corresponding time points. Thus, editing the components of a parametric curve induces changes in the P-curve, and editing the placement of the hash marks on the P-curve induces changes in the components.

The diagrammatic animation language DIAL [FEIN82b] retains some of the features of linear-list notations, but displays the sequence of events in an animation as a series of parallel rows of marks: A vertical bar indicates the initiation of an action, and dashes indicate the time during which the action is to take place. The actions are defined in a DIAL script (by statements of the form "% t1 translate "block" 1.0 7.0 15.3," which defines action t1 as the translation of an object called "block" by the vector (1.0, 7.0, 15.3)), and then the applications of the actions are defined subsequently. The particular instructions that DIAL executes are performed by a user-specified back end given at run time. DIAL

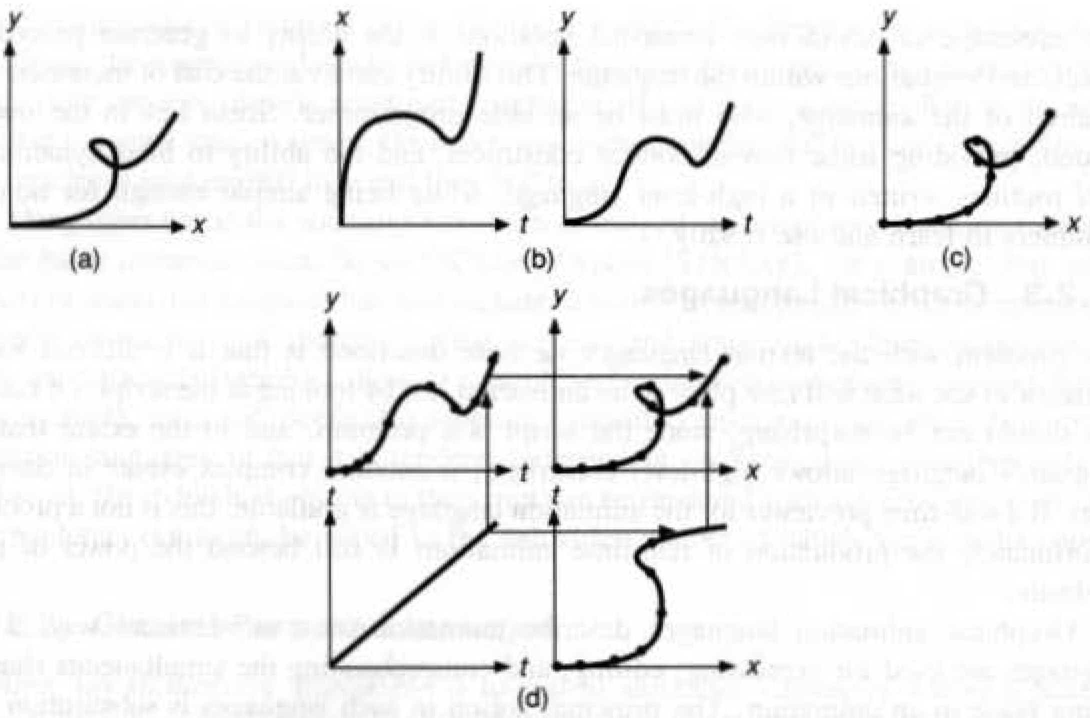


Fig. 21.5 (a) A parametric path in the plane. (b) Its x and y components as functions of time. (c) The original curve marked to indicate equal time steps. (d) The construction of a P-curve from component functions.

itself knows nothing about animation; it merely provides a description of the sequence in which instructions are to be performed. The following is a typical DIAL script (lines beginning with a blank are comments):

```
    Read in an object from a file, and assign it the name "block"
! getsurf "block.d" 5 5 "block"
    Define a window on the xy plane
! window -20 20 -20 20
    Define two actions, (1) a translation,
% t1 translate "block" 10 0 0
    and (2) a rotation in the xy plane by 360 degrees
% r1 rotate "block" 0 1 360
    Now describe a translation, spin, and a further translation:
t1  |---------      |--------

r1            |--------

r1  |------------------------
```

The line labeled "t1" indicates that action t1 is to take place from frames 1 to 10 (and hence is to translate by one unit per frame, since linear interpolation is the default), and then again from frames 17 to 25. At frame 11, the block stops translating and rotates 40° per frame for six frames (the first line labeled "r1" indicates this), and then rotates and translates for the next three, and then just translates.

For longer animations, each tick mark can indicate multiple frames, so that animations of several seconds' duration can be specified easily without indicating every frame. Long animations can be described by starting a new sequence of tick marks on a new line, following a completely blank line. (The second line labeled "r1" above is an example: it indicates that a 360° turn should take place between frames 26 and 50.) The format is much like that of a conductor's score of a symphony: Each action corresponds to an instrument in the orchestra, and each group of lines corresponds to a staff in the score. DIAL and many linear-list notations have the advantage of being specified entirely in ASCII text, making them portable to different machines (although a back end for the language must be written for each new machine).

The S-Dynamics system [SYMB85] takes this visual paradigm one step further and combines it with parametric descriptions of actions similar to P-curves. To do this, S-Dynamics uses the full power of a bitmapped workstation. Figure 21.6 shows an S-Dynamics window. Just as in DIAL, time runs horizontally across the window. The period during which actions are to take effect is indicated by the width of the region representing the action. Each action (or *sequence* in S-Dynamics terminology) can be shown as a box that indicates the time extent of the action, or the box can be "opened"—that is, made to show more internal detail. A sequence may be a composite of

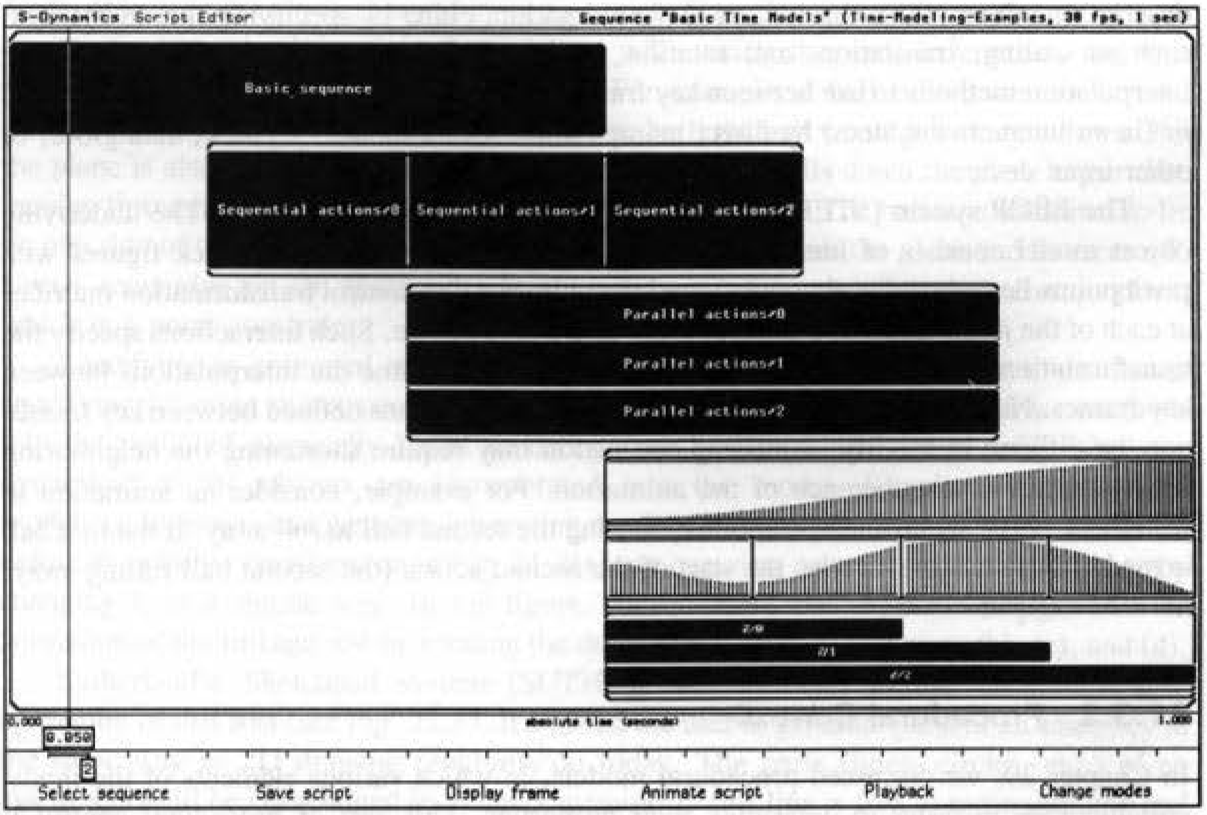


Fig. 21.6 An S-Dynamics window. (Courtesy of Symbolics Graphics Division. The software and the SIGGRAPH paper in which this image first appeared were both written by Craig Reynolds.)

several serial or parallel actions, each of which can be opened to show even more detail, including a graph indicating the time dependence of a parameter of the action.

21.3 METHODS OF CONTROLLING ANIMATION

Controlling an animation is somewhat independent of the language used for describing it—most control mechanisms can be adapted for use with various types of languages. Animation-control mechanisms range from full explicit control, in which the animator explicitly describes the position and attributes of every object in a scene by means of translations, rotations, and other position- and attribute-changing operators, to the highly automated control provided by knowledge-based systems, which take high-level descriptions of an animation ("make the character walk out of the room") and generate the explicit controls that effect the changes necessary to produce the animation. In this section, we examine some of these techniques, giving examples and evaluating the advantages and disadvantages of each.

21.3.1 Full Explicit Control

Explicit control is the simplest sort of animation control. Here, the animator provides a description of everything that occurs in the animation, either by specifying simple changes, such as scaling, translation, and rotation, or by providing key-frame information and interpolation methods to use between key frames. This interpolation may be given explicitly or (in an interactive system) by direct manipulation with a mouse, joystick, data glove, or other input device.

The BBOP system [STER83] provides this interactive sort of control. The underlying object model consists of hierarchical jointed polyhedral objects (i.e., stick figures with pivot points between adjacent sticks), and the animator can control transformation matrices at each of the joints using a joystick or other interactive device. Such interactions specify the transformations at key frames, and interactive programs define the interpolations between key frames. Notice that, in such a system, a sequence of actions defined between key frames may be difficult to modify; extending one action may require shortening the neighboring actions to preserve coherence of the animation. For example, consider an animation in which one ball rolls up and hits another, causing the second ball to roll away. If the first ball is made to move more slowly, the start of the second action (the second ball rolling away) must be delayed.

21.3.2 Procedural Control

In Chapter 20, we discussed procedural models, in which various elements of the model communicate in order to determine their properties. This sort of procedural control is ideally suited to the control of animation. Reeves and Blau [REEV85] modeled both grass and wind in this way, using a particle system modeling technique (see Section 20.5). The wind particles evolved over time in the production of the animation, and the positions of the

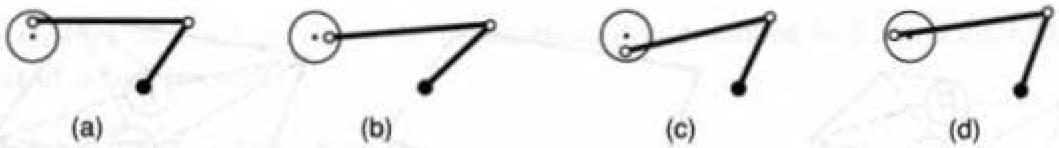


Fig. 21.7 The linkage in (a) is moved by rotating the drive wheel. The constraints generate the motions shown in (b), (c), and (d).

grass blades were then determined by the proximity of wind particles. Thus, the particle system describing the grass was affected by aspects of other objects in the scene. This sort of procedural interaction among objects can be used to generate motions that would be difficult to specify through explicit control. Unfortunately, it also requires that the animator be a programmer.

Procedural control is a significant aspect of several other control mechanisms we discuss. In particular, in physically based systems, the position of one object may influence the motion of another (e.g., balls cannot pass through walls); in actor-based systems, the individual actors may pass their positions to other actors in order to affect the other actors' behaviors.

21.3.3 Constraint-Based Systems

Some objects in the physical world move in straight lines, but a great many objects move in a manner determined by the other objects with which they are in contact, and this compound motion may not be linear at all. For example, a ball rolls down an inclined plane. If gravity were the only force acting on the ball, the ball would fall straight down. But the plane is also pushing up and sideways, and so the ball rolls down the plane rather than passing through it. We can model such motion by constraints. The ball is constrained to lie on one side of the plane. If it is dropped from a height, it strikes the plane and bounces off, always remaining on the same side. In a similar way, a pendulum swings from a pivot, which is a point constraint.

Specifying an animated sequence using constraints is often much easier to do than is specifying by using direct control. When physical forces define the constraints, we move into the realm of physically based modeling (see Section 21.3.7), especially when the dynamics[5] of the objects are incorporated into the model. Simple constraint-based modeling, however, can generate interesting results. If constraints on a linkage are used to define its possible positions, as in Fig. 21.7(a), we can view an animation of the linkage by changing it in a simple way. In the figure, for example, the animator can generate an animation of the linkage just by rotating the drive wheel, as shown in parts (b), (c), and (d).

Sutherland's Sketchpad system [SUTH63] was the first to use constraint-based animation of this sort (see Fig. 21.8). It allowed the user to generate parts of an assembly in the same way as 2D drawing programs do today. The parts (lines, circles, etc.) of an assembly could be constrained by point constraints ("this line is free to move, but one end

[5]Here we use *dynamics* in the sense of physics, to mean the change in position and motion over time, not merely to mean "change," as in earlier chapters.

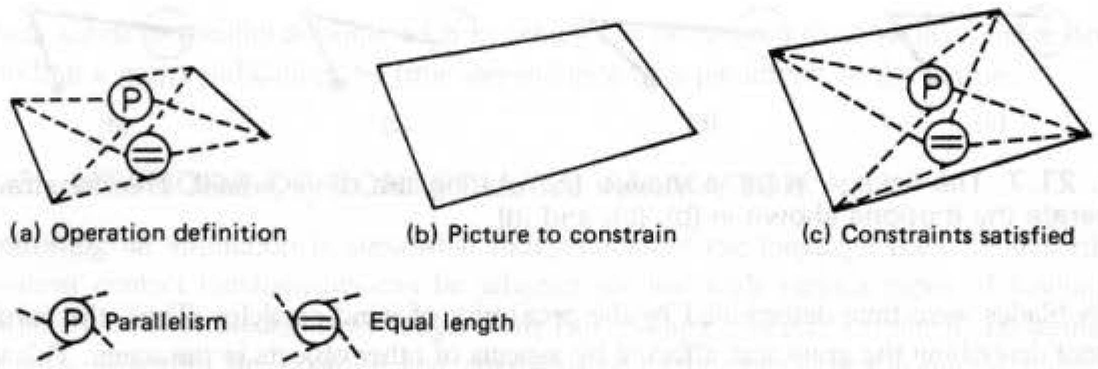


Fig. 21.8 Constraint definition and satisfaction in Sketchpad. (Adapted from [SUTH63].)

is held fixed at this point''), linkage constraints (''these lines must always remain joined end to end''), or angular constraints (''these lines must always be parallel'' or ''these lines must meet at a 60° angle''). This allowed the user to draw four lines in a quadrilateral, put linkage constraints on the corners, to put a point constraint at one corner, and to put angular constraints on opposite sides to make them parallel. This generated a parallelogram with one corner held fixed. Constraints were satisfied by a *relaxation technique* in which the assembly was moved so that the constraints came closer to being satisfied. Thus, the user could watch an assembly move so as to satisfy constraints gradually.[6] Of course, it is possible to overconstrain a system, by requiring, for example, that a line have a length of one unit, but that its ends be joined to two points that are three units apart. The constraints in Sketchpad are described by giving an error function—a function whose value is 0 when a constraint is satisfied, and is positive otherwise. Relaxation attempts to make the sum of these functions 0; when it fails, many constraints may be unsatisfied. Similarly, a system may be underconstrained, and have many solutions that satisfy all the constraints. In this case, the relaxation technique finds one solution that is close to the initial configuration.

Borning's similar ThingLab [BORN79] was really a metasystem: It provided a mechanism for defining systems like Sketchpad, but a user could define a system for modeling electrical circuits in the same framework. This system design was later improved to include a graphical interface [BORN86b]. A system, once designed, provided a world in which a user could build experiments. In a world meant to model geometry, for instance, the user could instantiate lines, point constraints, midpoint constraints, and so on, and then could move the assembly under those constraints. Figure 21.9 shows an example; the user has instantiated four MidpointSegments (segments with midpoints), has constrained their ends to be joined, and has also drawn four lines between adjacent midpoints. The user can vary the outer quadrilateral and observe that the inner quadrilateral always remains a parallelogram. For related work, see [BIER86a].

The extension of constraint-based animation systems to constraint systems supporting hierarchy, and to constraints modeled by the dynamics of physical bodies and the structural

[6]The animations therefore served two purposes: they generated assemblies satsifying the constraints, and they gave a *visualization* of the relaxation technique.

characteristics of materials (as in the plasticity models described in Section 20.7.3), is a subject of active research.

21.3.4 Tracking Live Action

Trajectories of objects in the course of an animation can also be generated by tracking of live action. There are a number of methods for doing tracking. Traditional animation has used *rotoscoping*: A film is made in which people (or animals) act out the parts of the characters in the animation, then animators draw over the film, enhancing the backgrounds and replacing the human actors with their animation equivalents. This technique provides exceptionally realistic motion. Alternatively, key points on an object may be digitized from a series of filmed frames, and then intermediate points may be interpolated to generate similar motion.

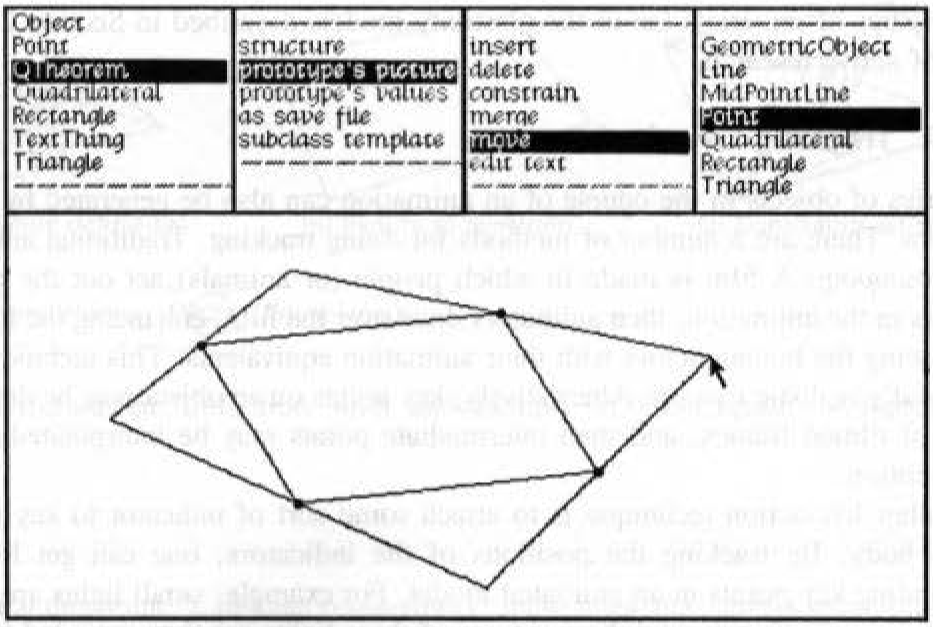
Another live-action technique is to attach some sort of indicator to key points on a person's body. By tracking the positions of the indicators, one can get locations for corresponding key points in an animated model. For example, small lights are attached at key locations on a person, and the positions of these lights are then recorded from several different directions to give a 3D position for each key point at each time. This technique has been used by Ginsberg and Maxwell [GINS83] to form a graphical marionette; the position of a human actor moving about a room is recorded and processed into a real-time video image of the motion. The actor can view this motion to get feedback on the motion that he or she is creating. If the feedback is given through a head-mounted display that can also display prerecorded segments of animation, the actor can interact with other graphical entities as well.

Another sort of interaction mechanism is the data glove described in Chapter 8, which measures the position and orientation of the wearer's hand, as well as the flexion and hyperextension of each finger joint. This device can be used to describe motion sequences in an animation as well, much like a 3D data tablet. Just as 2D motion can be described by drawing P-curves, 3D motion (including orientation) can be described by moving the data glove.

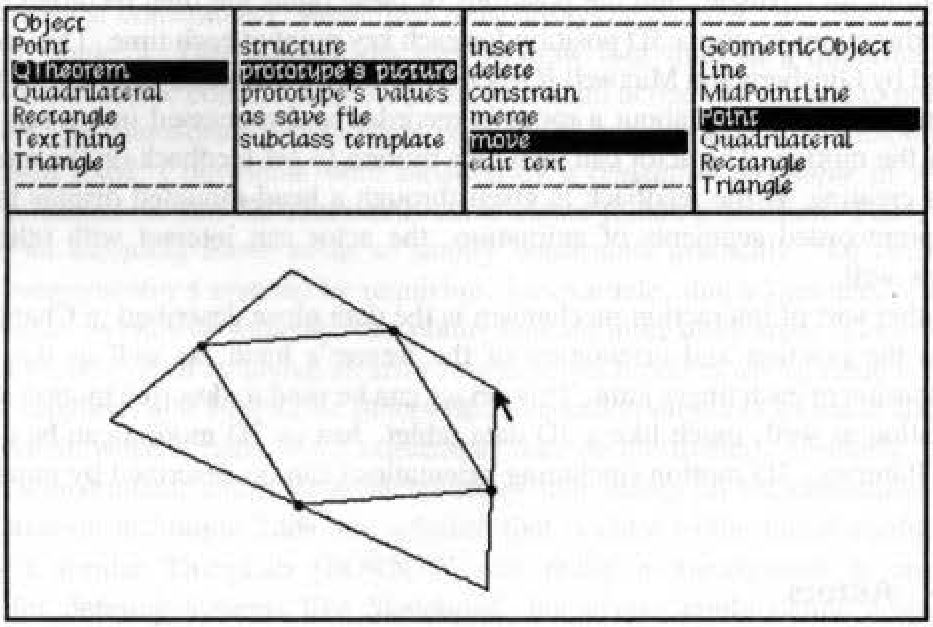
21.3.5 Actors

The use of *actors* is a high-level form of procedural control. An actor in an animation is a small program invoked once per frame to determine the characteristics of some object in the animation. (Thus, an actor corresponds to an ''object'' in the sense of object-oriented programming, as well as in the sense of animation.) An actor, in the course of its once-per-frame execution, may send messages to other actors to control their behaviors. Thus we could construct a train by letting the engine actor respond to some predetermined set of rules (move along the track at a fixed speed), while also sending the second car in the train the message ''place yourself on the track, with your forward end at the back end of the engine.'' Each car would pass a similar message to the next car, and the cars would all follow the engine.

Such actors were originally derived from a similar notion in Smalltalk [GOLD76] and other languages, and were the center of the ASAS animation system described in Section 21.2.2. The concept has been developed further to include actors with wide ranges of ''behaviors'' that they can execute depending on their circumstances.



(a)



(b)

21.3.6 Kinematics and Dynamics

Kinematics refers to the positions and velocities of points. A kinematic description of a scene, for example, might say, "The cube is at the origin at time $t = 0$. It moves with a constant acceleration in the direction (1, 1, 5) thereafter." By contrast, *dynamics* takes into account the physical laws that govern kinematics (Newton's laws of motion for large bodies, the Euler–Lagrange equations for fluids, etc.). A particle moves with an acceleration proportional to the forces acting on it, and the proportionality constant is the mass of the particle. Thus, a dynamic description of a scene might be, "At time $t = 0$ seconds the cube is at position (0 meters, 100 meters, 0 meters). The cube has a mass of 100 grams. The

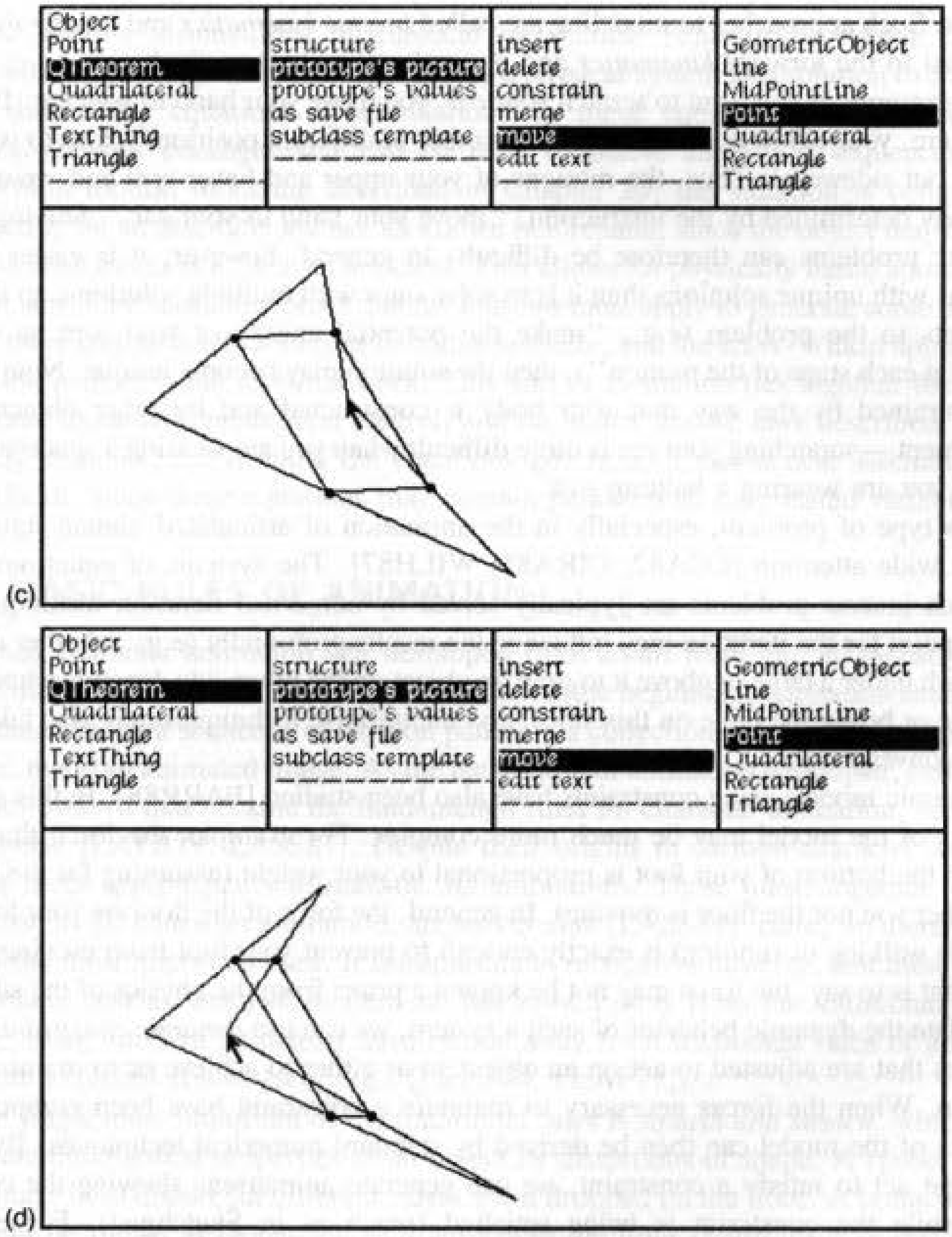


Fig. 21.9 A ThingLab display. (Courtesy of Alan Borning, Xerox PARC and University of Washington.)

force of gravity acts on the cube.'' Naturally, the result of a dynamic simulation[7] of such a model is that the cube falls.

Both kinematics and dynamics can be inverted; that is, we can ask the question, ''What must the (constant) velocity of the cube be for it to reach position (12, 12, 42) in 5 seconds?'' or, ''What force must we apply to the cube to make it get to (12, 12, 42) in 5 seconds?'' For simple systems, these sorts of questions may have unique answers; for more complicated ones, however, especially hierarchical models, there may be large families of

[7]This simulation could be based on either an explicit analytical solution of the equations of motion or a numerical solution provided by a package for solving differential equations.

solutions. Such approaches to modeling are called *inverse kinematics* and *inverse dynamics*, in contrast to the *forward kinematics* and *dynamics* already described.

For example, if you want to scratch your ear, you move your hand to your ear. But when it gets there, your elbow can be in any of a number of different positions (close to your body or stuck out sideways). Thus, the motions of your upper and lower arm and wrist are not completely determined by the instruction, "move your hand to your ear." Solving inverse kinematic problems can therefore be difficult. In general, however, it is easier to solve equations with unique solutions than it is to solve ones with multiple solutions, so if we add constraints to the problem (e.g., "make the potential energy of your arm as small as possible at each stage of the motion"), then the solution may become unique. Note that you are constrained by the way that your body is constructed and by other objects in the environment—scratching your ear is more difficult when you are wearing a spacesuit than it is when you are wearing a bathing suit.

This type of problem, especially in the animation of articulated human figures, has received wide attention [CGA82; GIRA85; WILH87]. The systems of equations arising from such inverse problems are typically solved by numerical iteration techniques. The starting point for the iteration may influence the results profoundly (e.g., whether a robot's arms reach under a table or above it to grab an object on the other side depends whether they are above or below the table on this side), and the iterative techniques may also take a long time to converge.

Dynamic models using constraints have also been studied [BARR88]. In this case, the dynamics of the model may be much more complex. For example, the force that a floor exerts on the bottom of your foot is proportional to your weight (assuming for the moment that neither you nor the floor is moving). In general, the force of the floor on your foot (even if you are walking or running) is exactly enough to prevent your foot from moving into the floor. That is to say, the force may not be known a priori from the physics of the situation.[8] To simulate the dynamic behavior of such a system, we can use *dynamic constraints*, which are forces that are adjusted to act on an object so as either to achieve or to maintain some condition. When the forces necessary to maintain a constraint have been computed, the dynamics of the model can then be derived by standard numerical techniques. By adding forces that act to satisfy a constraint, we can generate animations showing the course of events while the constraint is being satisfied (much as in Sketchpad). For example, constraining the end of a chain to connect to a post makes it move from where it is toward the post. This example was the subject of an animation by Barr and Barzel, one frame of which is shown in Color Plate IV.26.

21.3.7 Physically Based Animation

The dynamics we have described are examples of physically based animations. So, in animated form, are the physically based models of cloth, plasticity, and rigid-body motion described in Chapter 20. These models are based on simulations of the evolution of physical

[8]Of course, the floor actually does move when you step on it, but only a very small amount. We usually want to avoid modeling the floor as a massive object, and instead just model it as a fixed object.

systems. Various formulations of classical mechanical behavior have been developed [GOLD80]; they all represent the evolution of a physical system as a solution to a system of partial differential equations. The solutions to these equations can be found with numerical-analysis packages and can be used to derive animation sequences. In the Kass–Witkin motion modeling described in Chapter 20, the situation is complex. The forces acting on an assembly are not all known beforehand, since the object may be able to supply its own forces (i.e., use its muscles). This allows for physically based animation of a different sort: One seeks the forces that the muscles must apply to generate some action. Of course, there may be many solutions to such a problem, and the Kass–Witkin approach is to choose the path with the minimal work. This sort of animation ties together the work on constraints, dynamics, procedural control, and the actors that we have described. It is also extremely complex; determining the equations governing a mechanical assembly can be very difficult, since these equations may contain hundreds of interrelated variables.

21.4 BASIC RULES OF ANIMATION

Traditional character animation was developed from an art form into an industry at Walt Disney Studio between 1925 and the late 1930s. At the beginning, animation entailed little more than drawing a sequence of cartoon panels—a collection of static images that, taken together, made an animated image. As the techniques of animation developed, certain basic principles evolved that became the fundamental rules for character animation, and are still in use today [LAYB79; LASS87]. Despite their origins in cartoon-character animation, many of them apply equally to realistic 3D animations. These rules, together with their application to 3D character animation, are surveyed in [LASS87]. Here, we merely discuss a few of the most important ones. It is important to recognize, however, that these rules are not absolute. Just as much of modern art has moved away from the traditional rules for drawing, many modern animators have moved away from traditional rules of animation, often with excellent results (see, e.g., [LEAF74; LEAF77]).

The single most important of the traditional rules is *squash and stretch*, which is used to indicate the physical properties of an object by distortions of shape. A rubber ball or a ball of putty both distort (in different ways) when dropped on the floor. A bouncing rubber ball might be shown as elongating as it approachs the floor (a precursor to motion blur), flattening out when it hits, and then elongating again as it rises. By contrast, a metal sphere hitting the floor might distort very little but might wobble after the impact, exhibiting very small, high-frequency distortions. The jump made by Luxo Jr., described in Chapter 20 and simulated by the physically based modeling described in this chapter, is made with a squash and stretch motion: Luxo crouches down, storing potential energy in his muscles; then springs up, stretching out completely and throwing his base forward; and then lands, again crouching to absorb the kinetic energy of the forward motion without toppling over. It is a tribute to the potential of the Kass–Witkin simulation that it generated this motion automatically; it is also a tribute to traditional animators that they are able, in effect, to estimate a solution of a complex partial differential equation.

A second important rule is to use slow-in and slow-out to help smooth interpolations. Sudden, jerky motions are extremely distracting. This is particularly evident in interpolating the *camera position* (the point of view from which the animation is drawn or computed).

An audience viewing an animation identifies with the camera view, so sudden changes in camera position may make the audience feel motion sickness. Thus, camera changes should be as smooth as possible.

A third rule that carries over naturally from the 2D character-animation world to 3D animations, whether they are for the entertainment industry or for scientific visualization, is to *stage* the action properly. This includes choosing a view that conveys the most information about the events taking place in the animation, and (when possible) isolating events so that only one thing at a time occupies the viewer's attention. In the case of animations for scientific visualization, this isolation may not be possible—the events being simulated may be simultaneous—but it may be possible to view the scene from a position in which the different events occupy different portions of the image, and each can be watched individually without visual clutter from the others.

There are many other aspects of the design of animations that are critical. Many of these are matters of "eye" rather than strict rules, although rules of thumb are gradually evolving. The appropriate use of color is too often ignored, and garish animations in which objects are obscured by their colors are the result. The timing of animations is often driven by computing time instead of by final appearance; no time is given to introducing actions, to spacing them adequately, or to terminating them smoothly, and the resulting action seems to fly by. The details of an animation are given too much attention at the cost of the overall feeling, and the result has no aesthetic appeal. When you are planning an animation, consider these difficulties, and allot as much time as possible to aesthetic considerations in the production of the animation.

21.5 PROBLEMS PECULIAR TO ANIMATION

Just as moving from 2D to 3D graphics introduced many new problems and challenges, the change from 3D to 4D (the addition of the time dimension) poses special problems as well. One of these problems is *temporal aliasing*. Just as the aliasing problems in 2D and 3D graphics are partially solved by increasing the screen resolution, the temporal aliasing problems in animation can be partially solved by increasing temporal resolution. Of course, another aspect of the 2D solution is antialiasing; the corresponding solution in 3D is temporal antialiasing.

Another problem in 4D rendering is the requirement that we render many very similar images (the images in an ideal animation do not change much from one frame to the next—if they did, we would get jerky changes from frame to frame). This problem is a lot like that of rendering multiple scan lines in a 2D image: each scan line, on the average, looks a lot like the one above it. Just as scan-line renderers take advantage of this inter–scan-line coherence, it is possible to take advantage of interframe coherence as well. For ray tracing, we do this by thinking of the entire animation as occupying a box in 4D space–time—three spatial directions and one time direction. Each object, as it moves through time, describes a region of 4D space–time. For example, a sphere that does not move at all describes a spherical tube in 4D. The corresponding situation in 3D is shown in Fig. 21.10: If we make the 2D animation of a circle shown in part (a), the corresponding box in 3D space–time is that shown in part (b). The circle sweeps out a circular cylinder in space–time. For the 4D case, each image rendered corresponds to taking a 2D picture of a

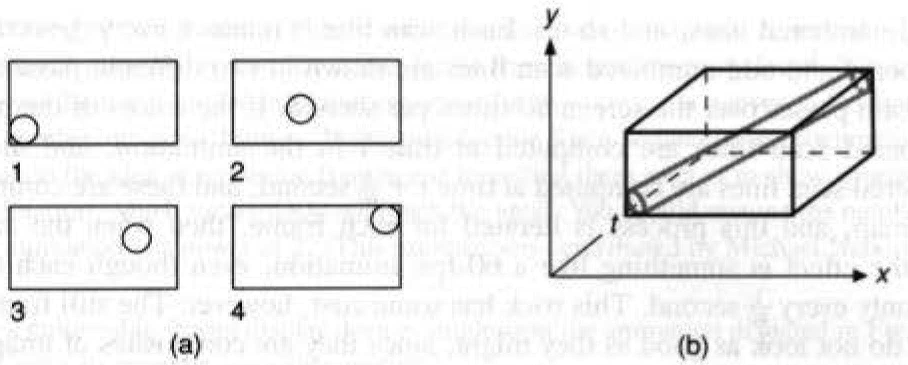


Fig. 21.10 The circle in (a) moves from lower left to upper right. By stacking these pictures along a third axis, we get the space–time animation shown in (b); the set of circles has become a tube in space–time.

3D slice of the 4D space–time. That is to say, we cast rays from a particular space–time point $(x, y, z; t)$ whose direction vectors have a time component of zero, so that all rays hit points whose time coordinate is also t. By applying the usual space-subdivision tricks for ray tracing to this 4D space–time, we can save a lot of time. A single hyperspace subdivision can be used for the entire course of the animation, so the time spent in creating the space subdivision does not need to be repeated once per frame. This idea and other uses of interframe coherence in ray tracing are described in [GLAS88].

High temporal resolution (many frames per second) may seem unnecessary. After all, video motion[9] seems smooth, and it is achieved at only 30 fps. Movies, however, at 24 fps, often have a jerkiness about them, especially when large objects are moving fast close to the viewer, as sometimes happens in a panning action. Also, as noted before, wagon wheels in movies sometimes appear to roll backward because of strobing. Higher temporal resolution helps to solve these problems. Doubling the number of frames per second lets the wagon wheel turn twice as fast before it seems to turn backward, and it certainly helps to smooth out the motion of fast-moving objects on the screen. The new Showscan technology [SHOW89] involves making and showing movies at 60 fps, on 70-millimeter film; this produces a bigger picture, which therefore occupies a larger portion of the visual field, and produces much smoother motion.

Temporal antialiasing can be done by taking multiple samples of a signal and computing their weighted average. In this case, however, the multiple samples must be in the time direction rather than in the spatial direction, so we compute the intensity at a point in the image for several sequential times and weight these to get a value at a particular frame. Many approaches to temporal-aliasing problems have been developed; super-sampling, box filtering in the time domain, and all the other tricks (including postfiltering!) from spatial antialiasing have been applied. One of the most successful is the distributed ray tracing described in Chapter 16 [COOK86].

Another trick for reducing temporal aliasing deserves mention: animation on fields. A conventional video image is traced twice; all the even-numbered scan lines are drawn, then

[9]We mean video motion filmed by a camera, not synthetically generated.

all the odd-numbered ones, and so on. Each scan line is redrawn every $\frac{1}{30}$ second, but the even-numbered and odd-numbered scan lines are drawn in two different passes. Thus, the electron beam passes over the screen 60 times per second. If the colors of the pixels of the even-numbered scan lines are computed at time t in the animation, and those for the odd-numbered scan lines are computed at time $t + \frac{1}{60}$ second, and these are composed into a single pixmap, and this process is iterated for each frame, then when the animation is displayed the effect is something like a 60-fps animation, even though each scan line is refreshed only every $\frac{1}{30}$ second. This trick has some cost, however: The still frames from an animation do not look as good as they might, since they are composites of images taken at two different times, and they thus seem to flicker if shown on an interlaced display. Also, twice as many frames must be rendered, so twice as many interpolated positions of the objects must be computed, and so on. Despite these drawbacks, the technique is widely employed in the computer-animation industry.

At the other extreme in animation is the process of *animating on twos*, or threes, and so on, in which the animation is produced at a temporal resolution lower than the display's refresh rate. Typically, each frame of the animation is displayed for two frames of video ("on twos"), so the effective refresh rate for video becomes 12 fps rather than 24 fps. This approach necessarily produces jerkier images (if no temporal antialiasing is done) or blurrier images (if it is). Animating on multiple frames and then filling in the intermediate ones can be useful in developing an animation, however, since it allows the animator to get a sense of the animation long before the individual frames have all been created (see Exercise 21.2.)

21.6 SUMMARY

Computer animation is a young field, and high-level animation is a recent development. As the computational power available to animators increases and as animation systems become more sophisticated, generating a high-quality computer animation will become simpler. At present, however, many compromises must be accepted. Simulation software is likely to advance rapidly, and the automated generation of graphical simulations is just a step away. On the other hand, until animation software contains knowledge about the tricks of conventional animation, computer character animation will remain as much an art as a science, and the "eye" of the animator will continue to have an enormous effect on the quality of the animation.

EXERCISES

21.1 Consider a unit square with corners at (0, 0) and (1, 1). Suppose we have a polygonal path defined by the vertices (0, 1), (1, 1), and (1, 0), in that order, and we wish to transform it to the polygonal path defined by the vertices (1, 0), (0, 0), and (0, 1) (i.e., we want to rotate it by 180°). Draw the intermediate stages that result if we linearly interpolate the positions of the vertices. This shows that strict interpolation of vertices is not adequate for key-frame interpolation unless the key frames are not too far apart.

21.2 Suppose that you are creating an animation, and can generate the frames in any order. If the animation is 128 frames long, a first "pencil sketch" can be created by rendering the first frame, and displaying it for a full 128 frames. (This is very low temporal resolution!). A second approximation can be generated by displaying the first frame for 64 frames, and the sixty-fourth frame for the next 64. Suppose you have a video recorder that can record a given image at a given video-frame number, for a given number of video frames. Write pseudocode for a sequential-approximation recording scheme based on the idea of rendering frames and recording them such as to show approximations of the entire animation, which successively approach the ideal. You should assume the number of frames in the entire animation is a power of 2. (This exercise was contributed by Michael Natkin and Rashid Ahmad.)

21.3 Using a color-table–based display device, implement the animation depicted in Fig. 21.4 . Can you think of ways to generate smoother motion?

21.4 Using a frame-buffer that supports *pan* and *zoom* operations, implement the pan-zoom movie technique described in Section 21.1.2.

21.5 Make an animation of fireworks, using the particle systems of Section 20.5. If you do not have a frame buffer capable of displaying the images, you may instead be able to program the particle systems in POSTSCRIPT, and to display them on a printer. Hold the resulting pictures as a book and riffle through them, making a *flip-book* animation.

21.6 Suppose you were trying to make a 2D animation system that started with scanned-in hand drawings. Suggest techniques for cleaning up the hand drawings automatically, including the closing of nearly closed loops, the smoothing of curved lines, but not of sharp corners, etc. The automation of this process is extremely difficult, and trying to imagine how to automate the process suggests the value of interactive drawing programs as a source for 2D animation material.

21.7 a. Suppose that q and r are quaternions corresponding to rotations of ϕ and θ about the axis $\mathbf{v}$. Explicitly compute the product qr and use trigonometric identities to show that it corresponds to the rotation about $\mathbf{v}$ by angle $\phi + \theta$.

b. Show that the product of two unit quaternions is a unit quaternion.

c. If q is the unit quaternion $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, and s is the quaternion $x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$, we can form a new quaternion $s' = qsq^{-1}$, where $q^{-1} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$. If we write $s' = x'\mathbf{i} + y'\mathbf{j} + z'\mathbf{k}$, then the numbers x', y', and z' depend on the numbers x, y, and z. Find a matrix Q such that $[x' \quad y' \quad z']^t = Q[x \quad y \quad z]^t$. When we generate rotations from quaternions, it is this matrix form that we should use, not an explicit computation of the quaternion product.

d. Show that the vector $[b \quad c \quad d]^t$ is left fixed under multiplication by Q, so that Q represents a rotation about the vector $[b \quad c \quad d]$. It actually represents a rotation by angle $2\cos^{-1}(a)$, so that this describes the correspondence between quaternions and rotations.

Appendix: Mathematics for Computer Graphics

This appendix reviews much of the mathematics used in the book. It is by no means intended as a text on linear algebra or geometry or calculus. The approach we take is somewhat unconventional, since most modern books on linear algebra do not mention affine spaces, and we choose to emphasize them. The text is liberally laced with exercises, which you should work through before looking at the solutions provided. The solutions are generally brief, and are intended to let you know whether you did the problem correctly, rather than to tell you how to do it.

The assumption we make in this appendix is that you have had courses in plane geometry, calculus, and linear algebra, but that your familiarity with all three subjects has faded somewhat. Thus, we give definitions for many important terms and state some important results, but the proofs are, for the most part, omitted; we have found that students interested in such proofs can generally construct them, and that those who are not interested in them find them distracting. Readers interested in reviewing this material in more detail should consult [BANC83; HOFF61; MARS85].

The first part of the appendix describes the geometry of affine spaces in some detail. In later sections, in which the material should be more familiar, we give considerably less detail. The final section discusses finding roots of real-valued functions, and is unrelated to the rest of the material.

A.1 VECTOR SPACES AND AFFINE SPACES

A vector space is, loosely, a place where addition and multiplication by a constant make sense. More precisely, a vector space consists of a set, whose elements are called *vectors*

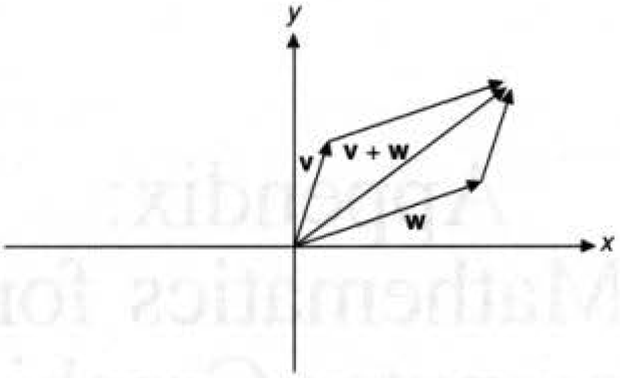


Fig. A.1 Addition of vectors in the plane.

(which we will denote by boldfaced letters, usually **u**, **v**, or **w**), together with two operations: addition of vectors, and multiplication of vectors by real numbers (called *scalar multiplication*).[1] The operations must have certain properties. Addition must be commutative, must be associative, must have an identity element (i.e., there must be a vector, traditionally called $\mathbf{0}$, with the property that, for any vector $\mathbf{v}$, $\mathbf{0} + \mathbf{v} = \mathbf{v}$), and must have inverses (i.e., for every vector $\mathbf{v}$, there is another vector $\mathbf{w}$ with the property that $\mathbf{v} + \mathbf{w} = \mathbf{0}$; $\mathbf{w}$ is written "$-\mathbf{v}$"). Scalar multiplication must satisfy the rules $(\alpha\beta)\mathbf{v} = \alpha(\beta\mathbf{v})$, $1\mathbf{v} = \mathbf{v}$, $(\alpha + \beta)\,\mathbf{v} = \alpha\mathbf{v} + \beta\mathbf{v}$, and $\alpha(\mathbf{v} + \mathbf{w}) = \alpha\mathbf{v} + \alpha\mathbf{w}$.

This definition of a vector space abstracts the fundamental geometric properties of the plane. We can make the plane into a vector space, in which the set of vectors is precisely the set of points in the plane. This identification of vectors and points is temporary, and is used for this example only. For now, we consider a point in the plane and a vector to be the same thing. To make the plane into a vector space, we must first choose a particular point in the plane, which we call the origin. We define addition of vectors by the well known *parallelogram rule*: To add the vectors $\mathbf{v}$ and $\mathbf{w}$, we take an arrow from the origin to $\mathbf{w}$, translate it so that its base is at the point $\mathbf{v}$, and define $\mathbf{v} + \mathbf{w}$ as the new endpoint of the arrow. If we also draw the arrow from the origin to $\mathbf{v}$, and do the corresponding process, we get a parallelogram, as shown in Fig. A.1. Scalar multiplication by a real number α is defined similarly: We draw an arrow from the origin to the point $\mathbf{v}$, stretch it by a factor of α, holding the end at the origin fixed, and then $\alpha\mathbf{v}$ is defined to be the endpoint of the resulting arrow. Of course, the same definitions can be made for the real number line or for Euclidean 3-space.

Exercise: Examine the construction of the vector space in the preceding paragraph. Does it depend in any way on assigning coordinates to points in the plane, or is it a purely geometrical construction? Suppose that we assign coordinates to points of the plane in the familiar fashion used in graphing. If we add the vectors whose coordinates are (a, b) and (c, d), what are the coordinates of the resulting vector? Suppose that instead we lay down coordinate lines so that one set of lines runs horizontally, but the other set, instead of

[1]Scalars (i.e., real numbers) will be denoted by Greek letters, typically by those near the start of the alphabet.

running vertically, runs at 30° away from vertical. What are the coordinates of the sum now?

Answer: No, it is purely geometrical (where geometry includes distance measure). The vector sum has coordinates $(a + c, b + d)$, in both cases.

The classic example of a vector space is $\mathbf{R}^n$, the set of all ordered n-tuples of real numbers. Addition is defined componentwise, as is scalar multiplication. Elements of $\mathbf{R}^n$ are written vertically, so that a sample element of $\mathbf{R}^3$ is

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}.$$

We can sum elements;

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}.$$

Most of graphics is done in $\mathbf{R}^2$, $\mathbf{R}^3$, or $\mathbf{R}^4$.

Given the two operations available in a vector space, there are some natural things to do with vectors. One of these is forming *linear combinations*. A linear combination of the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is any vector of the form $\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n$. Linear combinations of vectors are used for describing many objects. In the Cartesian plane example, the line through a nonzero point $\mathbf{v}$ and the origin can be described as the set of all vectors of the form $\alpha\mathbf{v}$, where α ranges over the real numbers. The ray from the origin through $\mathbf{v}$ is the same thing, except with α ranging over the nonnegative reals. These are both examples of linear "combinations" of a single vector. We will encounter more complex combinations later.

In general, the collection of all possible linear combinations of a set of vectors is called the *span* of the set. The span of a nonzero vector in the Cartesian plane example was a line through the origin. The span of two vectors that point in different directions is a plane.

Before we go further with vector spaces, we shall discuss affine spaces. An *affine space* is approximately describable as a set in which geometric operations make sense, but in which there is no distinguished point. (In a vector space, the vector $\mathbf{0}$ is special, and this is reflected in the example of the Cartesian plane, in which the origin plays a special role in the definition of addition and scalar multiplication.) A more precise definition of an affine space is that it consists of a set, called the *points* of the affine space; an associated vector space; and two operations. Given two points, P and Q, we can form the *difference* of P and Q, which lies in the vector space; given a point, P, and vector, $\mathbf{v}$, we can add the vector to the point to get a new point, $P + \mathbf{v}$. Again, there are certain properties that these operations must satisfy, such as $(P + \mathbf{v}) + \mathbf{w} = P + (\mathbf{v} + \mathbf{w})$, and $P + \mathbf{v} = P$ if and only if $\mathbf{v} = 0$.

This definition is based on a more classical model of geometry, in which there is no preferred origin. If you think of the surface of a table as an example of a (truncated) plane, there is no *natural* origin—no point of the table is preferred to any other. But if you take a point, P, on the table, and place a set of coordinate axes with their origin at P, every other point of the table can be measured by specifying its displacement from P using that coordinate system. By translating all the points of the coordinate axes by some fixed amount, we get new coordinate axes at another point. In this model, the points of the affine space are the points of the tabletop, and the vectors are arrows between them. Adding a vector $\mathbf{v}$ to a point P amounts to laying down the arrow with its base at P, and seeing where its end is (the endpoint is called $P + \mathbf{v}$). Taking the difference of two points Q and P, $Q - P$, consists of finding an arrow that goes from P to Q.

Affine planes make a natural model for computer graphics. Often, there is no preferred point in graphics. When you are modeling a room, for example, there is no natural point of the room to choose as an origin. Therefore, we shall discuss vector spaces and affine spaces side by side.

Linear combinations of points in an affine space make no sense (there is not even a definition of scalar multiplication), but we can define an *affine combination of the points P and Q by the real number t*. This affine combination is meant to correspond to a point that is a fraction t of the way from P to Q. (If t lies between 0 and 1, this is called a *convex combination*.) We can consider the difference of Q and P, $\mathbf{v} = Q - P$, which we think of as a vector pointing from P to Q. If we multiply this by t, we get a vector that is t times as long. Adding this vector back to P, we get the affine combination of P and Q by t, which is therefore

$$P + t(Q - P).$$

It is often tempting to rewrite this equation by gathering together the terms involving P, to get $(1 - t)P + tQ$; this makes no sense at all, however, since multiplication of points by scalars is undefined. Rather than outlaw this suggestive notation, however, we simply define it: If α and β are scalars that sum to 1, and P and Q are points in an affine space, we define $\alpha P + \beta Q$ to be $P + \beta(Q - P)$.

Affine combinations of more points are defined similarly: Given n points, $P_1, \ldots, P_n$, and n real numbers $t_1, \ldots, t_n$, satisfying $t_1 + \ldots + t_n = 1$, we define the affine combination of the Ps by the ts to be $P_1 + t_2(P_2 - P_1) + \ldots + t_n(P_n - P_1)$, which we also rewrite as $t_1P_1 + \ldots + t_nP_n$.

Exercise: Every vector space can be made into an affine space. The points of the affine space are the vectors in the vector space. The associated vector space is the original vector space. The difference of points is just defined to be the difference of vectors, and the sum of a point and a vector is the ordinary vector sum. Show that, in this case, the point we have defined as $\alpha P + \beta Q$ (where $\alpha + \beta = 1$) is actually equal, using the operations in the vector space, to the vector $\alpha P + \beta Q$.

Answer: $\alpha P + \beta Q$ is defined to be $P + \beta(Q - P)$. But ordinary vector operations apply, so this is just $P + \beta Q - \beta P = (1 - \beta)P + \beta Q = \alpha P + \beta Q$.

A.1.1 Equation of a Line in an Affine Space

If P and Q are two points in an affine space, the set of points of the form $(1 - t)P + tQ$ forms a line passing though P and Q; this form of a line is sometimes called the *parametric* *form*, because of the parameter t. The Cartesian plane, whose points are labeled with coordinates (x, y), is an affine space, and the parametric line between the point (a, b) and the point (c, d) is therefore given by

$$L = \{((1 - t)a + tc, (1 - t)b + td) \mid t \text{ is a real number}\}.$$

Exercise: Show that the set of all triples of real numbers of the form $(a, b, 1)$ also forms an affine space, with an associated vector space $\mathbf{R}^2$, provided we define the difference of two points $(a, b, 1)$ and $(c, d, 1)$ to be the vector $(a - c, b - d)$, and define the sum of a point and a vector similarly. Show that, using the definition of a parametric line given previously, the line between the points $(1, 5, 1)$ and $(2, 4, 1)$ consists entirely of points whose last coordinate is 1.

Answer: The definition of the line is the set of points of the form $(1 - t)(1, 5, 1) + t(2, 4, 1)$, which in turn is defined to mean $(1, 5, 1) + t(1, -1)$. These are points of the form $(1 + t, 5 - t, 1)$; hence, their last coordinate is 1.

A.1.2 Equation of a Plane in an Affine Space

If P, Q, and R are three points in an affine space, and they are not colinear (i.e., if R does not lie on the line containing P and Q), then the plane defined by P, Q, and R is the set of points of the form

$$(1 - s)((1 - t)P + tQ) + sR.$$

Exercise: Explain why the preceding expression makes geometric sense.

Answer: The expression is an affine combination of two points. The first point is $(1 - t)P + tQ$; the second is R. The first point is an affine combination of the points P and Q. Hence, all terms make sense.

Once again, this description of the plane is called *parametric*, because of the two parameters s and t.

Exercise: The set $\mathbf{E}^3$, consisting of all triples of real numbers, is an affine space, with an associated vector space $\mathbf{R}^3$, whose elements are also ordered triple of real numbers, but which have componentwise addition and scalar multiplication defined on them. The difference of two points in $\mathbf{E}^3$ is defined componentwise as well, as is the sum of a point and a vector. What points lie in the plane that contains the points $(1, 0, 4)$, $(2, 3, 6)$ and $(0, 0, 7)$?

Answer: The points of the plane are all points of the form $(1 - s)((1 - t)(1, 0, 4) + t(2, 3, 6)) + s(0, 0, 7)$. Because all operations are defined componentwise, we can express this as the set of all points of the form $((1 - s)(1 - t) + 2(1 - s)t, 3(1 - s)t, 4(1 - s)(1 - t) + 6t + 7s)$.

A.1.3 Subspaces

If we have a vector space, V, and a nonempty subset of V called S, then S is a *linear subspace of* V if, whenever $\mathbf{v}$ and $\mathbf{w}$ are in S, so are $\mathbf{v} + \mathbf{w}$ and $\alpha\mathbf{v}$, for every real number α. For example, if $\mathbf{v}$ is a vector, then the set of all vectors of the form $\alpha\mathbf{v}$ constitutes a subspace, because when any two scalar multiples of $\mathbf{v}$ are added, we get a third, and a scalar multiple of a scalar multiple of $\mathbf{v}$ is another scalar multiple of $\mathbf{v}$. In $\mathbf{R}^3$, the subspaces can be listed explicitly. They are (1) the origin, (2) any line through the origin, (3) any plane containing the origin, and (4) $\mathbf{R}^3$ itself.

Exercise: Show that any linear subspace of a vector space must contain the $\mathbf{0}$ vector.

Answer: Let $\mathbf{v}$ be any vector in S. Then $-1\mathbf{v} = -\mathbf{v}$ is also in S, and therefore $\mathbf{v} + (-\mathbf{v})$ is in S. But this is exactly the $\mathbf{0}$ vector. Merely scaling by 0 is not an adequate answer, since there is no a priori reason that $0\mathbf{v} = \mathbf{0}$ in a vector space. As it happens, since $(1 + (-1))\mathbf{v} = \mathbf{v} + (-\mathbf{v}) = 0$, it is actually true that $0\mathbf{v} = \mathbf{v}$; this statement merely happens not to be one of the axioms.

An affine subspace is a more general object. A nonempty subset S of a vector space V is called an *affine subspace* if the set $S' = \{\mathbf{u} - \mathbf{v} \mid \mathbf{u}, \mathbf{v} \text{ in } S\}$ is a linear subspace of V. For example, any line in the Cartesian plane is an affine subspace. If S is such a line, then S' is precisely the line parallel to it through the origin.

If S is an affine subspace of a vector space, then S can be thought of as an affine space in its own right. (Note that it is *not* a vector space. Consider the line $x = 1$ in $\mathbf{R}^2$. Both $\begin{bmatrix}1\\3\end{bmatrix}$ and $\begin{bmatrix}1\\5\end{bmatrix}$ are in this affine subspace, but their sum, $\begin{bmatrix}2\\8\end{bmatrix}$, is not.) The affine-space structure is given as follows: The associated vector space is just S'; the difference of two points in S lies in S' by definition, and the sum of a point in S with a vector in S' is another point in S.

Important Exercise: Show that S, the set of points of the form $(x, y, z, 1)$, forms an affine subspace of $\mathbf{R}^4$. What is the associated vector space? What is the difference of two points in this affine subspace?

Answer: The difference of any two points in S has the form $(a, b, c, 0)$, and the set of all points of this form is a vector space under the usual operations of addition an multiplication (in fact, it is essentially "the same" as $\mathbf{R}^3$). The associated vector space is the set of quadruples of the form $(a, b, c, 0)$. The difference of $(x, y, z, 1)$ and $(x', y', z', 1)$ is just $(x - x', y - y', z - z', 0)$.

The preceding example is important because it is the basis for all the material in Chapter 5. We can see in the example a clear distinction between *points* in the space, which are the things used to specify positions of objects in a graphics world, and *vectors*, which are used to specify displacements or directions from point to point. It is an unfortunate coincidence that a point of the form $(x, y, z, 1)$ can be stored in an array of three **reals** in Pascal, and a vector of the form $(a, b, c, 0)$ can too. Because of this, many people make the error of thinking that points and vectors are interchangeable. Nothing could be further from the truth. Denoting points in both the affine space and the ambient space by columns of numbers,

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix},$$

further confuses the issue, but it has become standard practice. We refer to the set of points in $\mathbf{R}^4$ whose last coordinate is 1 as the *standard affine 3-space in $\boldsymbol{R}^4$*. We correspondingly define the standard affine 2-space in $\mathbf{R}^3$ (which we call the standard affine plane), and so on.

Figure A.2 shows the standard affine plane in $\mathbf{R}^3$. This picture is far easier to draw than is the standard affine 3-space in $\mathbf{R}^4$, and we use it to provide intuition into that more complex case.The points in the standard affine plane are triples of the form

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

and the vectors have the form

$$\begin{bmatrix} a \\ b \\ 0 \end{bmatrix}.$$

(We have labeled the horizontal plane with the letters "x" and "y," and the vertical axis with the letter "h." This choice is meant to indicate the special nature of the third coordinate.) The set of points of the affine space forms a plane at height 1 above the (x, y) plane. The endpoints of the vectors (i.e., differences between points in the affine space) all lie in the (x, y) plane, if the starting point is placed at the origin (0, 0, 0), but are drawn as arrows in the affine plane to illustrate their use as differences of points. If we take two points, P and Q, in the affine space as shown, their sum (as vectors in $\mathbf{R}^3$) lies one full unit above the affine space. This shows geometrically the perils of adding points.

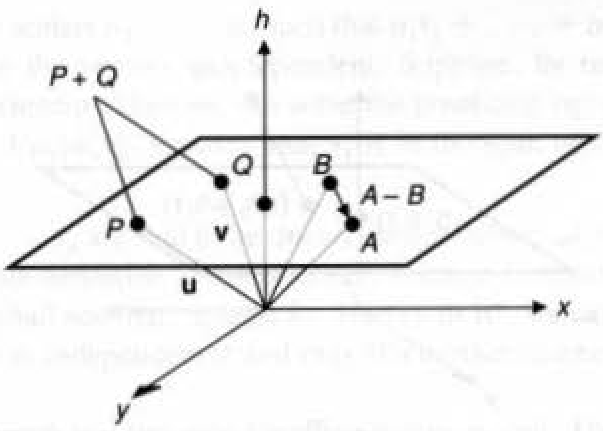


Fig. A.2 The standard affine plane in $\mathbf{R}^3$, embedded as the plane at $h = 1$. P and Q are points in the plane, but their sum lies above the plane. The difference of the points A and B is a horizontal vector.

Figure A.3 shows an important operation on this affine space: homogenization. If we take an arbitrary point

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix}$$

in 3-space, and connect it to the origin by a line, it will intersect the affine plane at a single point.

Exercise: Determine this point of intersection.

Answer: The line from

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix}$$

to the origin consists of all points of the form

$$\begin{bmatrix} \alpha x \\ \alpha y \\ \alpha h \end{bmatrix}.$$

We want to find the point whose third coordinate is 1. This point will be located precisely where $\alpha h = 1$; that is, where $\alpha = 1/h$. The coordinates of this point are therefore

$$\begin{bmatrix} x/h \\ y/h \\ h/h \end{bmatrix} = \begin{bmatrix} x/h \\ y/h \\ 1 \end{bmatrix}.$$

Naturally, this operation fails when $h = 0$, but this is no surprise, geometrically: A point in the (x, y) plane is connected to the origin by a line that never intersects the affine space.

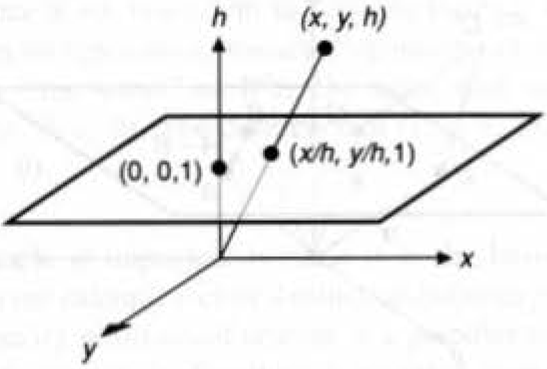


Fig. A.3 The homogenization operation in $\mathbf{R}^3$. The point (x, y, h) is homogenized to the point $(x/h, y/h, 1)$.

A.2 SOME STANDARD CONSTRUCTIONS IN VECTOR SPACES

A.2.1 Linear Dependence and Spans

We have defined the span of a set of vectors as the set of all linear combinations of those vectors. If we consider the special case of $\mathbf{R}^3$, the span of a single vector (except for $\mathbf{0}$) is the line through the origin containing the vector; the span of a pair of vectors (both nonzero, and neither lying in the span of the other) is a plane through the origin containing both; the span of three (sufficiently general) vectors is all of $\mathbf{R}^3$.

When one of the unusual cases (i.e., the cases rules out by the parenthetical conditions) in the preceding paragraph occurs, the vectors involved are said to be *linearly dependent* (or simply *dependent*). Essentially, a set of vectors is linearly dependent if one of them lies in the span of the rest.

Exercise: Show that the three vectors $\mathbf{a} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$, and $\mathbf{c} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$ are dependent by showing that $\mathbf{b}$ lies in the span of $\mathbf{a}$ and $\mathbf{c}$. Show also that $\mathbf{c}$, however, does not lie in the span of $\mathbf{a}$ and $\mathbf{b}$.

Answer: $\mathbf{b} = 2\mathbf{a} + 0\mathbf{c}$. On the other hand, the span of $\mathbf{a}$ and $\mathbf{b}$ consists of all vectors of the form

$$t\mathbf{a} + s\mathbf{b} = \begin{bmatrix} t \\ 3t \end{bmatrix} + \begin{bmatrix} 2s \\ 6s \end{bmatrix} = \begin{bmatrix} t + 2s \\ 3t + 6s \end{bmatrix} = (t + 2s) \begin{bmatrix} 1 \\ 3 \end{bmatrix};$$

hence, any vector in this span must be a scalar multiple of $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$. The vector $\mathbf{c}$ is not.

The more precise definition of linear dependence is that the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly dependent if there exist scalars $\alpha_1, \ldots, \alpha_n$ such that (1) at least one of the α_is is nonzero, and (2) $\alpha_1\mathbf{v}_1 + \ldots + \alpha_n\mathbf{v}_n = \mathbf{0}$.

Exercise: Show that, if the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are dependent, then one of them lies in the span of the others.

Answer: There are scalars $\alpha_1, \ldots, \alpha_n$ such that $\alpha_1\mathbf{v}_1 + \ldots + \alpha_n\mathbf{v}_n = \mathbf{0}$ and the scalars are not all zero, since the vectors are dependent. Suppose, by rearranging the order if necessary, that α_1 is nonzero. Then we can solve the preceding equation for $\mathbf{v}_1$ to get $\mathbf{v}_1 = (1/\alpha_1)\alpha_2\mathbf{v}_1 + \ldots + (1/\alpha_1)\alpha_n\mathbf{v}_n$, showing that $\mathbf{v}_1$ is in the span of the remaining vectors.

The vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are said to be *linearly independent* (or just *independent*) if they are not dependent. This definition is troublesome, because it requires verifying a negative statement. Later, we shall see that, at least for vectors in $\mathbf{R}^n$, we can restate it in a positive form: A set of vectors is independent if and only if a certain number (the determinant of a matrix) is nonzero.

We can define dependence and span for affine spaces as well. The span of a set of points $P_1, \ldots, P_n$ in an affine space can be defined in several ways. It is the set of all affine combinations of points in the set. We can also describe it by considering the vectors

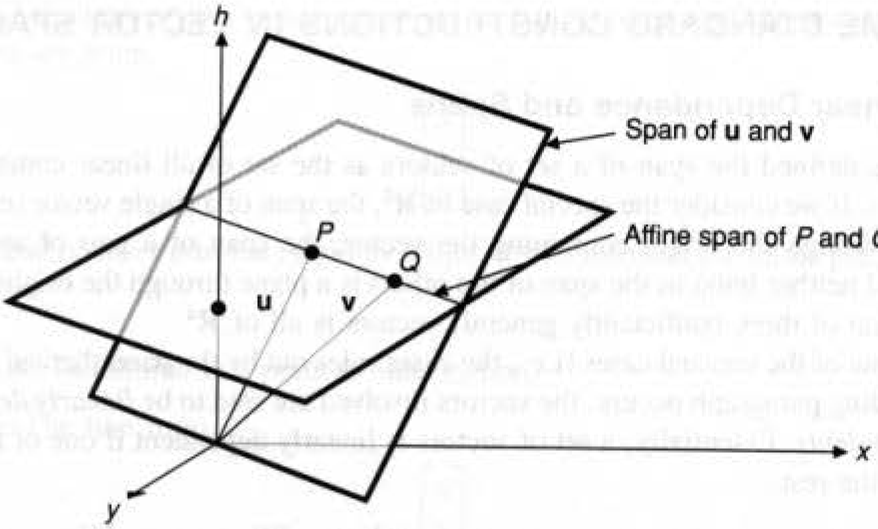


Fig. A.4 The relation between the vector space span of the vectors **u** and **v** in $\mathbf{R}^3$, whose endpoints are the points P and Q in the affine plane, and the affine span of P and Q.

$P_2 - P_1, P_3 - P_1, \ldots, P_n - P_1$ in the associated vector space, taking the span of these vectors, S, and then defining the affine span to be all points of the form $P_1 + \mathbf{v}$, where $\mathbf{v}$ is in S.

Just as for vector spaces, a collection of points in an affine space is said to be dependent if any one of them lies in the (affine) span of the others. It is independent if it is not dependent.

Consider again the special case of the standard affine plane consisting of points of the form

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

in $\mathbf{R}^3$. If we take two points in this affine space, we can form their affine span, which will be the line containing them. We can also form their span in a different way, by considering them as vectors in $\mathbf{R}^3$ and forming the vector-space span. Figure A.4 shows the relationship between these two spans—the affine span is the intersection of the vector-space span (a plane through the origin) with the affine space.

A.2.2 Coordinates, Coordinate Systems, and Bases

We can describe some large sets in a vector space using a compact notation, if we use the notion of spans. For example, we have described lines and planes as the spans of one and two vectors, respectively. We *could* describe a line by choosing two vectors that both lie in it, and saying it is the span of the two vectors, but that would be redundant—each of the two vectors would already be in the span of the other, in general.

A minimal spanning set for a vector subspace (or for an entire space) is called a *basis*. *Minimal* means the following: Any smaller set of vectors has a smaller span. Thus, in our previous example, the two vectors that spanned the line were not minimal, because one could be deleted and the remaining one would still span the line.

Exercise: Show that a basis of a subspace of a vector space is always linearly independent.

Answer: Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be a basis for the subspace S, and suppose that $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is dependent. Then (by renumbering if necessary), we can find scalars $\alpha_2, \ldots, \alpha_n$ such that $\mathbf{v}_1 = \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n$. A typical element of the span of $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is $\beta_1\mathbf{v}_1 + \ldots + \beta_n\mathbf{v}_n$. This expression can be rewritten as $\beta_1(\alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n) + \beta_2\mathbf{v}_2 + \ldots + \beta_n\mathbf{v}_n$. This can be rearranged into a linear combination of the vectors $\mathbf{v}_2, \ldots, \mathbf{v}_n$. Thus, any vector in the span of $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is also in the span of $\mathbf{v}_2, \ldots, \mathbf{v}_n$, so $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is not a minimal spanning set. Hence, the assumption that $\mathbf{v}_1, \ldots, \mathbf{v}_n$ was dependent must have been false.

Suppose we have a basis for a vector space. Then, every vector $\mathbf{v}$ in the space can be written as a linear combination $\alpha_1\mathbf{v}_1 + \ldots + \alpha_n\mathbf{v}_n$. Suppose we write $\mathbf{v}$ as a *different* linear combination, $\mathbf{v} = \beta_1\mathbf{v}_1 + \ldots + \beta_n\mathbf{v}_n$. Then, by subtraction, we get $\mathbf{0} = (\beta_1 - \alpha_1)\mathbf{v}_1 + \ldots + (\beta_n - \alpha_n)\mathbf{v}_n$. Since we assumed that the two linear combinations were different, some α_i must differ from the corresponding β_i. By renumbering, we can assume that $\alpha_1 \neq \beta_1$. But then, just as before, we can solve for $\mathbf{v}_1$ in terms of the remaining vectors, so the set $\mathbf{v}_1, \ldots, \mathbf{v}_n$ could not have been a minimal spanning set. Thus, every vector in a vector space can be written *uniquely* as a linear combination of the vectors in a basis.

If we have a basis for a vector space, $\mathbf{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, and a vector $\mathbf{v}$ in the vector space, we have just shown that there is a unique set of scalars $\alpha_1, \ldots, \alpha_n$ such that $\mathbf{v} = \alpha_1\mathbf{v}_1 + \ldots + \alpha_n\mathbf{v}_n$. This set of scalars can be thought of as an element of $\mathbf{R}^n$, and this element of $\mathbf{R}^n$,

$$\begin{bmatrix} \alpha_1 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_n \end{bmatrix},$$

is called the *coordinate vector of* $\mathbf{v}$ *with respect to the basis* $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Exercise: In $\mathbf{R}^3$, there is a *standard basis*, $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, where

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{and } \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

What are the coordinates of the vector

$$\mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}$$

with respect to this basis?

Answer: They are

$$\begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix},$$

because $\mathbf{v} = 3\mathbf{e}_1 + 4\mathbf{e}_2 + 2\mathbf{e}_3$.

The corresponding definition in an affine space is quite similar. A set of independent points in an affine space, whose affine span is the entire affine space, is called a *coordinate system*. If $P_1, \ldots, P_n$ is a coordinate system, then every point of the affine space can be written uniquely as an affine combination of $P_1, \ldots, P_n$; the coefficients are called the *affine coordinates of the point with respect to the coordinate system* $P_1, \ldots, P_n$.

Exercise: Show that, if $P_1, \ldots, P_n$ is a coordinate system for an affine space, then $P_2 - P_1, \ldots, P_n - P_1$ is a basis for the associated vector space.

Answer: Let $\mathbf{v}$ be a vector in the associated vector space, and let $Q = P_1 + \mathbf{v}$. Then Q can be written as an affine combination of $P_1, \ldots, P_n$. So there are scalars $\alpha_1, \ldots, \alpha_n$ such that $\alpha_1 + \ldots + \alpha_n = 1$ and $Q = \alpha_1 P_1 + \alpha_2 P_2 + \ldots + \alpha_n P_n = P_1 + \alpha_2(P_2 - P_1) + \ldots + \alpha_n(P_n - P_1)$. But this implies that $\mathbf{v} = \alpha_2(P_2 - P_1) + \ldots + \alpha_n(P_n - P_1)$. Hence, the set $P_2 - P_1, \ldots, P_n - P_1$ spans the associated vector space. If the set were dependent, the corresponding set of points $P_1, \ldots, P_n$ in the affine space would be dependent. Hence, it must both span and be independent, so it is a basis.

A.3 DOT PRODUCTS AND DISTANCES

The vector spaces and affine spaces we have discussed so far are purely algebraic objects. No metric notions—such as distance and angle measure—have been mentioned. But the world we inhabit, and the world in which we do graphics, both do have notions of distance and angle measure. In this section, we discuss the dot (or inner) product on $\mathbf{R}^n$, and examine how it can be used to measure distances and angles. A critical feature of distance measure and angle measure is that they make sense for vectors, not points: To measure the distance between points in an affine space, we take the difference vector and measure its length.

A.3.1 The Dot Product in $\mathbf{R}^n$

Given two vectors

$$\begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} y_1 \\ . \\ . \\ . \\ y_n \end{bmatrix}$$

in $\mathbf{R}^n$, we define their *inner product* or *dot product* to be $x_1y_1 + \ldots + x_ny_n$. The dot product of vectors $\mathbf{v}$ and $\mathbf{w}$ is generally denoted by $\mathbf{v} \cdot \mathbf{w}$.

The distance from the point (x, y) in the plane to the origin $(0, 0)$ is $\sqrt{x^2 + y^2}$. In general, the distance from the point $(x_1, \ldots, x_n)$ to the origin in n-space is $\sqrt{x_1^2 + \ldots + x_n^2}$. If we let $\mathbf{v}$ be the vector

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix},$$

we can see that this is just $\sqrt{\mathbf{v} \cdot \mathbf{v}}$. This is our definition of the *length* of a vector in $\mathbf{R}^n$. We denote this length by $\| \mathbf{v} \|$. The distance between two points in the standard affine n-space is defined similarly: The distance between P and Q is the length of the vector $Q - P$.

Exercise: The points

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

both lie in the standard affine plane, as well as in $\mathbf{R}^3$. What is the distance from each of them to the origin in $\mathbf{R}^3$? What is the distance between them? What is the distance from each to the point

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

in the standard affine plane? What is the dot product of the two vectors

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} ?$$

Answer: The distances to the origin are $\sqrt{11}$ and $\sqrt{30}$, respectively. The distance between the points is $\sqrt{5}$. The distances to

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

are $\sqrt{10}$ and $\sqrt{29}$ respectively. The dot product is 18. Note that asking for the dot products of the two *points* in the affine space makes no sense—dot products are defined only for vectors.

A.3.2 Properties of the Dot Product

The dot product has several nice properties. First, it is symmetric: $\mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{v}$. Second, it is *nondegenerate*: $\mathbf{v} \cdot \mathbf{v} = 0$ only when $\mathbf{v} = \mathbf{0}$. Third, it is *bilinear*: $\mathbf{v} \cdot (\mathbf{u} + \alpha\mathbf{w}) = \mathbf{v} \cdot \mathbf{u} + \alpha(\mathbf{v} \cdot \mathbf{w})$.

The dot product can be used to generate vectors whose length is 1 (this is called *normalizing a vector*.) To normalize a vector, **v**, we simply compute $\mathbf{v}' = \mathbf{v} / \| \mathbf{v} \|$. The resulting vector has length 1, and is called a *unit vector*.

Exercise: What is the length of the vector

$$\begin{bmatrix} 4 \\ 3 \\ 0 \end{bmatrix}?$$

What do we get if we normalize this vector? Consider the points

$$P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

in the standard affine plane. What is the unit vector pointing in the direction from P to Q?

Answers: The length of the vector is 5. The normalized vector is

$$\begin{bmatrix} 4/5 \\ 3/5 \\ 0 \end{bmatrix}.$$

The unit direction vector from P to Q is

$$\begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \\ 0 \end{bmatrix}.$$

Note that the last component is 0.

Dot products can also be used to measure angles (or, from a mathematician's point of view, to *define* angles). The *angle between the vectors* **v** *and* **w** is

$$\cos^{-1}\left(\frac{\mathbf{v} \cdot \mathbf{w}}{\| \mathbf{v} \| \| \mathbf{w} \|}\right).$$

Note that, if **v** and **w** are unit vectors, then the division is unnecessary.

If we have a unit vector **v** and another vector **w**, and we project **w** perpendicularly onto **v**, as shown in Fig. A.5, and call the result **u**, then the length of **u** should be the length of **w** multiplied by $\cos(\theta)$, where θ is the angle between **v** and **w**. That is to say,

$$\begin{aligned} \| \mathbf{u} \| &= \| \mathbf{w} \| \cos(\theta) \\ &= \| \mathbf{w} \| \left(\frac{\mathbf{v} \cdot \mathbf{w}}{\| \mathbf{v} \| \| \mathbf{w} \|}\right) \\ &= \mathbf{v} \cdot \mathbf{w}, \end{aligned}$$

since the length of **v** is 1. This gives us a new interpretation of the dot product: The dot product of **v** and **w** is the length of the projection of **w** onto **v**, provided **v** is a unit vector.

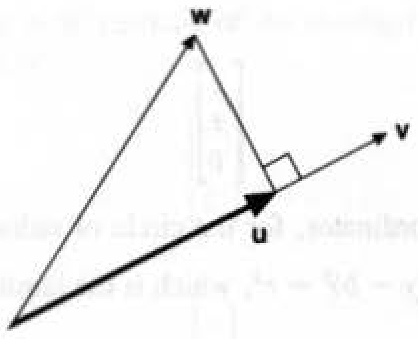


Fig. A.5 The projection of **w** onto the unit vector **v** is a vector **u**, whose length is $\| \mathbf{w} \|$ times the cosine of the angle between **v** and **w**.

Exercise: Show that, if **v** and **w** are unit vectors, then projection of **v** onto **w** and the projection of **w** onto **v** have the same length.

Answer: Both of these are represented by $\mathbf{v} \cdot \mathbf{w}$, so they are the same.

Since $\cos\theta = 0$ precisely when $\theta = 90°, 270°$, and so on, we can use the dot product of vectors to determine when they are perpendicular. Two vectors **v** and **w** are perpendicular exactly when $\mathbf{v} \cdot \mathbf{w} = 0$.

A.3.3 Applications of the Dot Product

Since dot products can be used to measure lengths, we can generate some simple equations using them. For example, if we have a point, P, in an affine plane, the equation for a circle with center P and radius r is easy to write. We simply want all points Q whose distance from P is exactly r. Thus, the equation is

$$\| Q - P \| = r.$$

We can rewrite this as

$$\sqrt{(Q - P) \cdot (Q - P)} = r,$$

or as

$$(Q - P) \cdot (Q - P) = r^2.$$

In the standard affine 3-space, the equation of the plane passing through a point P and perpendicular to a vector **v** is also easy to express. A point Q on this plane is characterized by having the difference $Q - P$ be perpendicular to the vector **v**. Hence, the equation is just

$$(Q - P) \cdot \mathbf{v} = 0.$$

Exercise: Suppose P and Q are points of the standard affine plane; P is the point

$$\begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

and Q is the indeterminate

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

What is the equation, in coordinates, for the circle of radius r about P?

Answer: It is $(x - a)^2 + (y - b)^2 = r^2$, which is the familiar formula from high-school algebra.

Exercise: If P is the point

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

in the standard affine 3-space, and $\mathbf{v}$ is the vector

$$\begin{bmatrix} 2 \\ 2 \\ 3 \\ 0 \end{bmatrix},$$

what is the equation of the plane perpendicular to $\mathbf{v}$ and passing through P?

Answer: If we let the indeterminate point be

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

then the equation becomes $2(x - 1) + 2(y - 2) + 3(z - 3) = 0$.

In general, the equation for a plane through the point

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

and normal to the vector

$$\begin{bmatrix} A \\ B \\ C \\ 0 \end{bmatrix}$$

is $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$. This can be rewritten as $Ax + By + Cz = Ax_0 +$

$By_0 + Cz_0$. If we are working in $\mathbf{R}^3$ (instead of the standard affine 3-space), this equation just says that the dot product of

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

with

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

must be the same as the dot product of

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

with

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}.$$

The constant term (on the right side of the equation above) is precisely this second dot product. If

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

is a unit vector, then this dot product measures the length of the projection of

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

onto the unit vector, and hence tells us how far the plane is from the origin in $\mathbf{R}^3$.

In the equation for a plane, we are characterizing the plane by specifying its normal vector. This is dangerous in one sense, since normal vectors and ordinary vectors are different in a quite subtle manner, as we shall discuss later.

A.3.4 Distance Formulae

If we define a plane through P, perpendicular to $\mathbf{v}$, by the equation $(Q - P) \cdot \mathbf{v} = 0$, we can ask how far a point R is from the plane. One way to determine this distance is to compute the projection of $R - P$ onto the vector $\mathbf{v}$ (see Fig. A.6). The length of this projected vector is just the distance from R to the plane. But this length is also the dot product of $R - P$ and the vector $\mathbf{v}$, divided by the length of $\mathbf{v}$. Thus, if we consider a plane defined by the equation

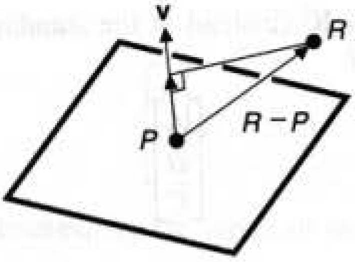


Fig. A.6 We can measure the distance from the point R to the plane by projecting R onto the normal to the plane.

$Ax + By + Cz + D = 0$, the distance from a point (r, s, t) to the plane is exactly $(Ar + Bs + Ct + D) / \sqrt{A^2 + B^2 + C^2}$. Note that the square root in the denominator is just the length of the normal vector to the plane, so if the plane was defined using a *unit* normal vector, no division is necessary.

Exercise: Suppose we are given a line in an affine space in parametric form, as the set of all points $P(t) = P_0 + t\mathbf{v}$, and a point R not on the line. What is the distance from R to the line?

Answer: The distance from a point to a line is defined as the minimum of all distances from the point to points on the line. This minimum will occur when the line from R to the point on the line is perpendicular to the line—that is, when $(R - P(t)) \cdot \mathbf{v} = 0$. Expanding this, we get

$$(R - P_0 - t\mathbf{v}) \cdot \mathbf{v} = 0,$$

$$(R - P_0) \cdot \mathbf{v} = t\mathbf{v} \cdot \mathbf{v},$$

$$t = \frac{(R - P_0) \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}}.$$

So this is the value of t at which the distance is minimized. Plugging this into the formula for points on the line, we find that the point closest to R is exactly

$$P_0 + \frac{(R - P_0) \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v}.$$

We obtain the distance from R to this point by subtracting the point from R (to get a vector) and then computing its length. The formula fails when $\mathbf{v} = \mathbf{0}$; in this case, however, $P(t) = P_0 + t\mathbf{v}$ does not define a line.

A.3.5 Intersection Formulae

Suppose that, in the standard affine plane, we have a circle, $(X - P) \cdot (X - P) = r^2$, and a line, $S(t) = Q + t\mathbf{v}$. What points lie on the intersection of these two? Well, such a point must be $S(t)$ for some value of t, and it must satisfy the equation of the circle, so we must solve

$$(S(t) - P) \cdot (S(t) - P) = r^2.$$

Algebraic manipulations reduce this expression to

$$t^2\,(\mathbf{v}\cdot\mathbf{v}) + t\,(2\,\mathbf{v}\cdot(Q-P)) + ((Q-P)\cdot(Q-P) - r^2) = 0,$$

which is a quadratic equation in t. Solving this with the quadratic formula gives the possible values for t, which can then be used in the formula for $S(t)$ to determine the actual points of intersection. Notice that this in no way depends on the number of coordinates. If we take the set of points in the standard affine 3-space defined by $(X-P)\cdot(X-P) = r^2$, which is a sphere, the same solution gives us the two points of intersection of a line with the sphere.

Exercise: Determine the intersections of the line through the point

$$\begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

(in the standard affine plane) in the direction

$$\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

with the circle of radius 5 centered at

$$\begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}.$$

Answer: Let t_1 and t_2 be $(-2 \pm 2\sqrt{31})/5$. The intersections occur at

$$\begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} + t_i \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix},$$

for $i = 1, 2$.

Suppose we are given a line and a plane in the standard affine 3-space. How can we determine their intersection? If the plane is given in point-normal form, as $(X-P)\cdot\mathbf{v} = 0$, and the line is given in parametric form, as $Q(t) = Q + t\mathbf{w}$, we can simply replace X by $Q(t)$ and solve:

$$(Q + t\mathbf{w} - P)\cdot\mathbf{v} = 0.$$

Solving this equation for t gives

$$t = \frac{(P-Q)\cdot\mathbf{v}}{\mathbf{w}\cdot\mathbf{v}},$$

and the intersection point is at

$$Q + \frac{(P-Q)\cdot\mathbf{v}}{\mathbf{w}\cdot\mathbf{v}}\,\mathbf{w}.$$

This technique for finding intersections is quite general. If we have a surface in the standard affine 3-space defined by the equation $F(x, y, z, 1) = 0$, we can substitute the point $P + t\,\mathbf{v}$ for the argument $(x, y, z, 1)$, which yields an equation in the single variable t. Solving for t gives the parametric value for the point on the ray at the intersection point. Substituting this t value into $P + t\,\mathbf{v}$ gives the actual point of intersection.

In general, an implicitly defined surface (i.e., a surface defined by an equation of the form $F(x, y, z, 1) = 0$ in the standard affine 3-space) has a surface normal vector at the point $(x, y, z, 1)$; the coordinates of this vector are given by the partial derivatives of F at the point. (The corresponding situation in the plane was discussed in the scan-conversion of ellipses in Chapter 3.) The normal vector is thus

$$\begin{bmatrix} \dfrac{\partial F}{\partial x}(x, y, z, 1) \\[2ex] \dfrac{\partial F}{\partial y}(x, y, z, 1) \\[2ex] \dfrac{\partial F}{\partial z}(x, y, z, 1) \\[2ex] 0 \end{bmatrix}.$$

A.3.6 Orthonormal Bases

Two vectors, $\mathbf{u}$ and $\mathbf{v}$, are said to be *orthogonal* if $\mathbf{u} \cdot \mathbf{v} = 0$. If $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is a basis for a vector space, and each $\mathbf{b}_i$ is a unit vector, and every two vectors in the basis are orthogonal, the basis is said to be an *orthonormal basis*. We can express these conditions more simply by saying that B is an orthonormal basis if $\mathbf{b}_i \cdot \mathbf{b}_j = 0$ unless $i = j$, in which case $\mathbf{b}_i \cdot \mathbf{b}_j = 1$.

Orthonormal bases have a number of convenient properties that other bases lack. For example, if **B** is an orthonormal basis, and we wish to write a vector $\mathbf{v}$ as a linear combination of the vectors in B, $\mathbf{v} = \alpha_1\mathbf{b}_1 + \ldots + \alpha_n\mathbf{b}_n$, it is easy to find the value of α_i: It is just $\mathbf{v} \cdot \mathbf{b}_i$.

Exercise: Show that in $\mathbf{R}^n$, the standard basis $E = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$, (where $\mathbf{e}_i$ has all entries 0 except the ith, which is 1), is an orthonormal basis. Show that the vectors

$$\begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}$$

form an orthonormal basis for $\mathbf{R}^2$. What are the coordinates of the vector $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ in this basis?

Answer: The first two parts are direct computations. The coordinates are $11/\sqrt{5}$ and $-2/\sqrt{5}$.

Because of this convenient property, it is often desirable to convert a basis into an orthonormal basis. This is done with the Gram–Schmidt process. The idea of this process is to take each vector in turn, to make it orthogonal to all the vectors considered so far, and then to normalize it. If we start with a basis $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, the process is this:

Let $\mathbf{v}_1' = \mathbf{v}_1$ (no vectors have been considered so far, so this is trivial).
Let $\mathbf{w}_1 = \mathbf{v}_1' / \| \mathbf{v}_1' \|$.
Let $\mathbf{v}_2' = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{w}_1)\mathbf{w}_1$ (this is orthogonal to $\mathbf{w}_1$).
Let $\mathbf{w}_2 = \mathbf{v}_2' / \| \mathbf{v}_2' \|$.
Let $\mathbf{v}_3' = \mathbf{v}_3 - (\mathbf{v}_3 \cdot \mathbf{w}_1)\mathbf{w}_1 - (\mathbf{v}_3 \cdot \mathbf{w}_2)\mathbf{w}_2$.
Let $\mathbf{w}_3 = \mathbf{v}_3' / \| \mathbf{v}_3' \|$.

The vectors $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$ are an orthonormal basis. The process for a larger number of vectors is similar. The last step, for the case of three vectors, can be simplified; see Exercise A.7.

A.4 MATRICES

A matrix is a rectangular array of numbers. Its elements are doubly indexed, and by convention the first index indicates the row and the second indicates the column. Mathematical convention dictates that the indices start at 1; certain programming languages use indices that start at 0. We leave it to programmers in those languages to shift all indices by 1. Thus, if $\mathbf{A}$ is a matrix, then $a_{3,2}$ refers to the element in the third row, second column. When symbolic indices are used, as in a_{ij}, the comma between them is omitted.

Elements of $\mathbf{R}^n$, which we have been writing in the form

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix},$$

can be considered to be $n \times 1$ matrices.

A.4.1 Matrix Multiplication

Matrices are multiplied according to the following rule: If $\mathbf{A}$ is an $n \times k$ matrix with entries a_{ij}, and $\mathbf{B}$ is a $k \times p$ matrix with entries b_{ij}, then $\mathbf{AB}$ is defined, and is an $n \times p$ matrix with entries c_{ij}, where $c_{ij} = \sum_{s=0}^{k} a_{is}b_{sj}$. If we think of the columns of $\mathbf{B}$ as individual vectors, $\mathbf{B}_1, \ldots, \mathbf{B}_p$, and the rows of $\mathbf{A}$ as vectors $\mathbf{A}_1, \ldots, \mathbf{A}_k$ as well (but rotated 90° to be horizontal), then we see that c_{ij} is just $\mathbf{A}_i \cdot \mathbf{B}_j$. The usual properties of multiplication hold, except that matrix multiplication is not commutative: $\mathbf{AB}$ is, in general, different from $\mathbf{BA}$. But multiplication distributes over addition: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$, and there is an identity element for multiplication—namely, the *identity matrix*, $\mathbf{I}$, which is a square matrix with all entries 0 except for 1s on the diagonal (i.e., the entries are δ_{ij}, where $\delta_{ij} = 0$ unless $i = j$, and $\delta_{ii} = 1$).

A.4.2 Determinants

The determinant of a square matrix is a single number that tells us a great deal about the matrix. The columns of the matrix are linearly independent if and only if the determinant of the matrix is nonzero. Every $n \times n$ matrix represents a transformation from $\mathbf{R}^n$ to $\mathbf{R}^n$, and the determinant of the matrix tells us the volume change induced by this transformation (i.e., it tells us how much the unit cube is expanded or contracted by the transformation).

Computing the determinant is somewhat complicated, because the definition is recursive. The determinant of the 2×2 matrix $\begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is just $ad - bc$. The determinant of an $n \times n$ matrix is defined in terms of determinants of smaller matrices. If we let A_{1i} denote the determinant of the $(n-1) \times (n-1)$ matrix gotten by deleting the first row and ith column from the $n \times n$ matrix $\mathbf{A}$, then the determinant of $\mathbf{A}$ is defined by

$$\det \mathbf{A} = \sum_{i=1}^{n} (-1)^{i+1} a_{1i} A_{1i}.$$

An alternate way to compute the determinant is to use *Gaussian elimination*. Gaussian elimination works by sequences of *row operations*. There are three types of row operations on a matrix: (1) exchanging any two rows, (2) multiplying a row by a nonzero scalar, and (3) adding a multiple of row i to row j (row i is left unchanged, and row j is replaced with (row j) + α(row i)). The algorithm for reducing an $n \times n$ matrix $\mathbf{A}$ by Gaussian elimination is simple: Arrange (by exchanging rows and scaling) that $a_{11} = 1$. For each $j \neq 1$, subtract a_{j1} times row 1 from row j, so that a_{j1} then becomes zero. Now, by exchanging the second row with subsequent rows (if necessary) and scaling, arrange that $a_{22} = 1$. For each $j \neq 2$, subtract a_{j2} times row 2 from row j. Continue this process until the matrix becomes the identity matrix.

In the course of this process, it may be impossible to make $a_{ii} = 1$ for some i (this happens when the entire column i is zero, for example); in this case, the determinant is zero. Otherwise, the determinant is computed by taking the multiplicative inverse of the product of all the scalars used in type-2 row operations in Gaussian elimination, and then multiplying the result by $(-1)^k$, where k is the number of row exchanges done during Gaussian elimination.

One special application of the determinant works in $\mathbf{R}^3$: the *cross-product*. The cross-product of two vectors

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

is computed by taking the determinant of the matrix,

$$\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix},$$

where the letters $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are treated as symbolic variables. The result is then a linear combination of the variables $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$; at this point, the variables are replaced with the vectors $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ respectively. The result is the vector

$$\begin{bmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{bmatrix},$$

which is denoted by $\mathbf{v} \times \mathbf{w}$. It has the property that it is perpendicular to the plane defined by $\mathbf{v}$ and $\mathbf{w}$, and its length is the product $\|\mathbf{v}\|\,\|\mathbf{w}\|\,|\sin \theta|$, where θ is the angle between $\mathbf{v}$ and $\mathbf{w}$. It also has the property that a matrix whose columns are $\mathbf{v}$, $\mathbf{w}$, and $\mathbf{v} \times \mathbf{w}$ will always have nonnegative determinant.

This last characteristic is an interesting one, and can be used to define *orientation*. Two bases for $\mathbf{R}^n$ are said to have the same orientation if, when the vectors in each basis are used to form the columns of a matrix, the two resulting matrices have determinants of the same sign. A basis is said to be *positively oriented* if it has the same orientation as the standard basis; it is *negatively oriented* otherwise.

Exercise: Show that the basis $\{\mathbf{e}_2, \mathbf{e}_1, \mathbf{e}_3, \mathbf{e}_4\}$ is a negatively oriented basis for $\mathbf{R}^4$.

Answer: The determinant of the corresponding matrix is -1.

Exercise: Suppose two planes are defined by the equations $(X - P) \cdot \mathbf{v} = 0$ and $(X - Q) \cdot \mathbf{w} = 0$. What is the direction vector for the line of intersection of the two planes?

Answer: Since the line of intersection lies in each plane, its direction vector must be orthogonal to the normal vectors to each plane. One such vector is the cross product $\mathbf{v} \times \mathbf{w}$. If $\mathbf{v}$ and $\mathbf{w}$ are parallel, then the planes either are identical or do not intersect at all; so, in the case where $\mathbf{v} \times \mathbf{w} = 0$, the problem is degenerate anyway.

A.4.3 Matrix Transpose

An $n \times k$ matrix can be flipped along its diagonal (upper left to lower right) to make a $k \times n$ matrix. If the first matrix has entries a_{ij} ($i = 1, \ldots, n$; $j = 1, \ldots, k$), then the resulting matrix has entries b_{ij} ($i = 1, \ldots, k$; $j = 1, \ldots, n$), with $b_{ij} = a_{ji}$. This new matrix is called the *transpose* of the original matrix. The transpose of $\mathbf{A}$ is written $\mathbf{A}^t$. If we consider a vector in $\mathbf{R}^n$ as an $n \times 1$ matrix, then its transpose is a $1 \times n$ matrix (sometimes called a row vector). Using the transpose, we can give a new description of the dot product in $\mathbf{R}^n$; namely, $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^t\mathbf{v}$.

Exercise: Compute one example indicating, and then prove in general, that if $\mathbf{A}$ is $n \times k$ and $\mathbf{B}$ is $k \times p$, then $(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t$.

Answer: We leave this problem to you.

A.4.4 Matrix Inverse

Matrix multiplication differs from ordinary multiplication in another way: A matrix may not have a multiplicative inverse. In fact, inverses are defined only for square matrices, and not even all of these have inverses. Exactly those square matrices whose determinants are nonzero have inverses.

If $\mathbf{A}$ and $\mathbf{B}$ are $n \times n$ matrices, and $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$, where $\mathbf{I}$ is the $n \times n$ identity matrix, then $\mathbf{B}$ is said to be the inverse of $\mathbf{A}$, and is written $\mathbf{A}^{-1}$. For $n \times n$ matrices with real number entries, it suffices to show that either $\mathbf{AB} = \mathbf{I}$ or $\mathbf{BA} = \mathbf{I}$—if either is true, the other is as well.

If we are given an $n \times n$ matrix, there are two basic ways to find its inverse: Gaussian elimination and Cramer's rule. Gaussian elimination is the preferred method for anything larger than 3×3.

The inverse of a matrix can be computed using Gaussian elimination by writing down both $\mathbf{A}$ and the identity matrix. As you perform row operations on $\mathbf{A}$ to reduce it to the identity, you perform the same row operations on the identity. When $\mathbf{A}$ has become the identity matrix, the identity matrix will have become $\mathbf{A}^{-1}$. If, during Gaussian elimination, some diagonal entry cannot be made 1, then, as we noted, the determinant is 0, and the inverse does not exist. This technique can be improved in numerous ways. A good reference, including working programs for implementation, is [PRESS88].

A different method for computing inverses is called *Cramer's rule*. It builds the inverse explicitly, but at the cost of computing many determinants. Here is how it works.

To compute the inverse of an $n \times n$ matrix $\mathbf{A}$ with entries a_{ij}, we build a new matrix, $\mathbf{A}'$, with entries A_{ij}. To compute A_{ij}, we delete rows i and j from the matrix $\mathbf{A}$, and then compute the determinant of the resulting $(n-1) \times (n-1)$ matrix. Multiplying this determinant by $(-1)^{i+j}$ gives the value for A_{ij}. Once $\mathbf{A}'$ is computed, the inverse of $\mathbf{A}$ is just $(1 / \det \mathbf{A})\,(\mathbf{A}')^t$.

Because of the large number of determinants involved, Cramer's rule is impractical for large matrices. For the 2×2 case, however, it is quite useful. It tells us that

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

One last special case for matrix inversion deserves mention. Suppose that $\mathbf{U}$ is a matrix whose columns form an orthonormal basis. This means that $\mathbf{u}_i \cdot \mathbf{u}_j = \delta_{ij}$ for all i and j. Consider what happens when we compute $\mathbf{U}^t\mathbf{U}$. We have noted that the ij entry of the product is the dot product of the ith row of the first factor and the jth column of the second factor. But these are just $\mathbf{u}_i$ and $\mathbf{u}_j$; hence, their dot product is δ_{ij}. This tells us that $\mathbf{U}^t\mathbf{U} = \mathbf{I}$, and hence that $\mathbf{U}^{-1} = \mathbf{U}^t$. Note, by the way, that this means that the columns of $\mathbf{U}^t$ also form an orthonormal basis!

A.5 LINEAR AND AFFINE TRANSFORMATIONS

A *linear transformation* is a map from one vector space to another that preserves linear combinations. More precisely, it is a map $\mathbf{T}$ with the property that $\mathbf{T}(\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n) = \alpha_1\mathbf{T}(\mathbf{v}_1) + \alpha_2\mathbf{T}(\mathbf{v}_2) + \ldots + \alpha_n\mathbf{T}(\mathbf{v}_n)$. Linear transformations are the ones we describe in great detail in Chapter 5.

An *affine transformation* is a map from one affine space to another that preserves affine combinations. More precisely, it is a map $\mathbf{T}$ with the property that $\mathbf{T}(P + \alpha(Q - P)) = \mathbf{T}(P) + \alpha(\mathbf{T}(Q) - \mathbf{T}(P))$. $\mathbf{T}$ extends naturally to a map on the associated vector space. We define $\mathbf{T}(\mathbf{v})$ to be $\mathbf{T}(P) - \mathbf{T}(Q)$, where P and Q are any two points with $Q - P = \mathbf{v}$. Affine transformations include translations, rotations, scales, and shearing transformations. Note that the transformations defined in Chapter 5 are both affine *and* linear transformations. They are linear transformations from $\mathbf{R}^4$ to $\mathbf{R}^4$, but they take the standard affine 3-space (the points of $\mathbf{R}^4$ whose last coordinate is 1) to itself, so that they also describe affine transformations on this affine space.

A.5.1 The Matrix for a Transformation on $\mathbf{R}^n$

Suppose we have n independent vectors, $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbf{R}^n$, and we wish to find a linear transformation $\mathbf{T}$ from them to the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$. (We have chosen this odd naming convention because the $\mathbf{b}_i$s, being independent, form a basis.) How can we do this? The simplest way to express a linear transformation on $\mathbf{R}^n$ is to give a matrix for it. That is to say, we will find an $n \times n$ matrix $\mathbf{A}$ such that $\mathbf{T}(\mathbf{v}) = \mathbf{A}\mathbf{v}$ for all $\mathbf{v}$ in $\mathbf{R}^n$.

We begin by solving a simpler problem. We find a matrix for the transformation that takes the standard basis vectors, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ to an arbitrary set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Suppose we take any $n \times n$ matrix $\mathbf{Q}$ with entries q_{ij} and multiply it by $\mathbf{e}_j$. If we let $\mathbf{r} = \mathbf{Q}\mathbf{e}_j$, then $\mathbf{r}_i = \mathbf{q}_{ij}$. That is, multiplying a matrix by the jth standard basis vector extracts the jth column of the matrix. We can reverse this observation to find the matrix that transforms the standard basis vectors into $\mathbf{v}_1, \ldots, \mathbf{v}_n$: We just use the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ as the columns of the matrix.

Exercise: Find a matrix taking the standard basis of $\mathbf{R}^2$ to the vectors $\begin{bmatrix}1\\2\end{bmatrix}$ and $\begin{bmatrix}3\\3\end{bmatrix}$.
Answer: $\begin{bmatrix}1 & 3\\2 & 3\end{bmatrix}$.

To solve the original problem of this section, finding a transformation taking the $\mathbf{b}_i$s to the $\mathbf{a}_i$s, we apply the solution for the simpler problem twice. First, we find a matrix $\mathbf{B}$ (whose columns are the $\mathbf{b}_i$s) that takes the standard basis to the $\mathbf{b}_i$s; then, we find a matrix $\mathbf{A}$ that takes the standard basis to the $\mathbf{a}_i$s. The matrix $\mathbf{B}^{-1}$ will do just the opposite of $\mathbf{B}$, and take the $\mathbf{b}_i$s to the standard basis, so the matrix $\mathbf{A}\mathbf{B}^{-1}$ is the solution to the original problem. It is a matrix taking the $\mathbf{b}_i$s to the $\mathbf{a}_i$s.

Exercise: Find a matrix transformation taking $\begin{bmatrix}1\\2\end{bmatrix}$ and $\begin{bmatrix}2\\5\end{bmatrix}$ to $\begin{bmatrix}1\\1\end{bmatrix}$ and $\begin{bmatrix}3\\2\end{bmatrix}$, respectively.

Answer: The matrix taking the standard basis to the first pair of vectors is $\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix}$; the matrix taking the standard basis to the second pair is $\begin{bmatrix}1 & 3\\1 & 2\end{bmatrix}$. The solution is therefore $\mathbf{T}(\mathbf{v}) = \mathbf{Q}\mathbf{v}$, where

$$\mathbf{Q} = \begin{bmatrix}1 & 3\\1 & 2\end{bmatrix}\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix}^{-1} = \begin{bmatrix}-1 & 1\\1 & 0\end{bmatrix}.$$

A.5.2 Transformations of Points and Normal Vectors

When we apply a matrix linear transformation to the points of the standard affine n-space, how do the differences between points (i.e., the vectors of the affine space) transform? Suppose our transformation is defined by $\mathbf{T}(P) = \mathbf{A}P$, and further suppose that this transformation sends the affine plane to itself (i.e., that there is no homogenization required after the transformation—this is equivalent to saying the last row of $\mathbf{A}$ is all 0s, except the bottom-right entry, which is a 1). Then, $\mathbf{T}(Q - P) = \mathbf{A}(Q - P)$. But $Q - P$ has a 0 in its last component (since both P and Q have a 1 there). Hence, the last column of $\mathbf{A}$ has no effect in the result of the transformation. We therefore define $\mathbf{A}'$ to be the same as $\mathbf{A}$, but with its last column replaced by all 0s except the last entry, which we make 1. This matrix, $\mathbf{A}'$, can be used to transform vectors in the affine space.

We mentioned previously that the definition of a plane by its normal vector was dangerous, and here we see why. Suppose we have a plane whose points satisfy $(X - P) \cdot \mathbf{v} = 0$. When we transform this plane by $\mathbf{A}$, we will get a new plane containing $\mathbf{A}P$, so it will have an equation of the form $(Y - \mathbf{A}P) \cdot \mathbf{w} = 0$ for some vector $\mathbf{w}$. We want those points of the form $\mathbf{A}X$, where X is on the original plane, to satisfy this second equation. So we want to find a vector $\mathbf{w}$ with the property that $(\mathbf{A}X - \mathbf{A}P) \cdot \mathbf{w} = 0$ whenever $(X - P) \cdot \mathbf{v} = 0$. Expressed differently, we want

$$(\mathbf{A}X - \mathbf{A}P)^t\, \mathbf{w} = 0 \qquad \text{whenever } (X - P) \cdot \mathbf{v} = 0.$$

By distributing the transpose operator, we see this reduces to

$$(X - P) \cdot \mathbf{A}^t\mathbf{w} = 0 \qquad \text{whenever } (X - P) \cdot \mathbf{v} = 0.$$

This equation will certainly hold if $\mathbf{A}^t\mathbf{w} = \mathbf{v}$—that is, if $\mathbf{w} = (\mathbf{A}^t)^{-1}\mathbf{v}$. Thus, $(\mathbf{A}^t)^{-1}\mathbf{v}$ is the normal vector to the transformed plane. In the event that $\mathbf{A}$ is an orthogonal matrix (as it is, e.g., in the case of rotations), we know that $(\mathbf{A}^t)^{-1} = \mathbf{A}$, so the normal vector transforms in the same way as the point (but with no translation, because the last component of the vector is 0). But this is not true for general matrices. The computation of the inverse transpose of $\mathbf{A}$ can be somewhat simplified by computing instead the inverse transpose of $\mathbf{A}'$, whose effect on vectors is the same as that of $\mathbf{A}$. Since $\mathbf{A}'$ is effectively a smaller matrix (its last row and column are the same as those of the identity matrix), this is often much easier.[2]

Computing the inverse of a matrix may be difficult—and, if you use Cramer's rule, it involves dividing by the determinant. Since the normal vector, after being transformed, will probably no longer be a unit vector and will need to be normalized, leaving out this division does no harm. Thus, people sometimes use the *matrix of cofactors* for transforming normals. Entry ij of this matrix is $(-1)^{i+j}$ times the determinant of the matrix resulting from deleting row i and column j from $\mathbf{A}$.

A.6 EIGENVALUES AND EIGENVECTORS

An *eigenvector* of a transformation $\mathbf{T}$ is a vector $\mathbf{v}$ such that $\mathbf{T}(\mathbf{v})$ is a scalar multiple of $\mathbf{v}$. If $\mathbf{T}(\mathbf{v}) = \lambda\mathbf{v}$, then λ is called the *eigenvalue* associated with $\mathbf{v}$. The theoretical method for finding eigenvalues (at least for a matrix transformation $\mathbf{T}(\mathbf{v}) = \mathbf{A}\,\mathbf{v}$) is to let $\mathbf{B} = \mathbf{A} - x\mathbf{I}$, where $\mathbf{I}$ is the identity matrix, and x is an indeterminate. The determinant of $\mathbf{B}$ is then a polynomial in x, $p(x)$. The roots of p are precisely the eigenvalues. If λ is one such eigenvalue, and λ is a real number, then $\mathbf{T}(\mathbf{v}) = \lambda\mathbf{v}$ must be true for some vector $\mathbf{v}$. By rearranging, we get that $\mathbf{A}\mathbf{v} - \lambda\mathbf{v} = 0$, or $\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = 0$. Thus, finding all solutions to this last equation gives us all the eigenvectors corresponding to λ.

Although this approach is theoretically feasible, in practice it is not very useful, especially for large matrices. Instead, numerical methods based on iterating the transforma-

[2]Differential geometers refer to vectors such as the normal vector as *covectors*, since these vectors are defined by a dot-product relation with ordinary (or *tangent*) vectors. The set of all covectors is sometimes called the *cotangent* space, but this term has no relation to trigonometric functions. More complex objects, called *tensors*, can be made up from mixtures of tangent and cotangent vectors, and the rules for transforming them are correspondingly complex.

tion are used. Chief among these is Gauss–Seidel iteration; for details of this technique, see [PRES88].

We conclude this section with a particularly interesting pair of exercises.

Exercise: Show that eigenvectors of a symmetric matrix (one for which $\mathbf{M}^t = \mathbf{M}$) corresponding to distinct eigenvalues are always orthogonal. Show that for any square matrix $\mathbf{A}$, the matrix $\mathbf{A}^t\mathbf{A}$ is symmetric.

Answer: Suppose $\mathbf{Mv} = \lambda\mathbf{v}$, and $\mathbf{Mu} = \mu\mathbf{u}$. Let us compute $\mathbf{u}^t\mathbf{Mv}$ in two different ways:

$$\mathbf{u}^t\mathbf{Mv} = \mathbf{u}^t\lambda\mathbf{v} = \lambda\, \mathbf{u}^t\mathbf{v} = \lambda\, (\mathbf{u} \cdot \mathbf{v}).$$

But

$$\mathbf{u}^t\mathbf{Mv} = (\mathbf{u}^t\mathbf{M}^t)\, \mathbf{v} = (\mathbf{Mu})^t\mathbf{v} = \mu\, \mathbf{u}^t\mathbf{v} = \mu\, (\mathbf{u} \cdot \mathbf{v}).$$

Thus, $\lambda\, (\mathbf{u} \cdot \mathbf{v}) = \mu\, (\mathbf{u} \cdot \mathbf{v})$; hence, $(\lambda - \mu)(\mathbf{u} \cdot \mathbf{v}) = 0$. Since λ and μ are distinct eigenvalues, we know that $(\lambda - \mu) \neq 0$. Hence, $(\mathbf{u} \cdot \mathbf{v}) = 0$.

The transpose of $(\mathbf{A}^t\mathbf{A})$ is just $\mathbf{A}^t(\mathbf{A}^t)^t$; but the transpose of the transpose is the original matrix, so result is just $\mathbf{A}^t\mathbf{A}$. Hence, $\mathbf{A}^t\mathbf{A}$ is symmetric.

Exercise: Suppose that $\mathbf{T}(\mathbf{x}) = \mathbf{Ax}$ is a linear transformation on $\mathbf{R}^2$, and that we apply it to all points of the unit circle. The resulting set of points forms an ellipse whose center is the origin. Show that squares of the lengths of the major and minor axes of the ellipse have lengths equal to the maximum and minimum singular values of $\mathbf{A}$, where a *singular value* of $\mathbf{A}$ is defined to be an eigenvalue of $\mathbf{A}^t\mathbf{A}$.

Answer: The points on the transformed circle are of the form $\mathbf{Ax}$, where $\mathbf{x} \cdot \mathbf{x} = 1$. The square of the distance from such a transformed point to the origin is just $\mathbf{Ax} \cdot \mathbf{Ax}$, or, rewriting, $\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x}$. Let $\mathbf{u}$ and $\mathbf{v}$ be the two unit eigenvectors of $\mathbf{A}^t\mathbf{A}$, with corresponding eigenvalues λ and μ. Because they are orthogonal, they form a basis for $\mathbf{R}^2$. We can therefore write $\mathbf{x}$ as a linear combination of them: $\mathbf{x} = \cos\theta\, \mathbf{u} + \sin\theta\, \mathbf{v}$. If we now compute $\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x}$, we get

$$\begin{aligned}
\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x} &= (\cos\theta\, \mathbf{u}^t + \sin\theta\, \mathbf{v}^t)\, (\cos\theta\, \mathbf{A}^t\mathbf{Au} + \sin\theta\, \mathbf{A}^t\mathbf{Av}) \\
&= (\cos\theta\, \mathbf{u}^t + \sin\theta\, \mathbf{v}^t)\, (\cos\theta\, \lambda\mathbf{u} + \sin\theta\, \mu v) \\
&= \lambda \cos^2\theta + \mu \sin^2\theta.
\end{aligned}$$

This function has its extreme values at θ = multiples of 90°—that is, when $\mathbf{x} = \pm\mathbf{u}$ or $\pm\, \mathbf{v}$. The values at those points are just λ and μ.

A.7 NEWTON–RAPHSON ITERATION FOR ROOT FINDING

If we have a continuous function, f, from the reals to the reals, and we know that $f(a) > 0$ and $f(b) < 0$, then there must be a root of f between a and b. One way to find the root is *bisection*: We evaluate f at $(a + b)/2$; if it is positive, we search for a root in the interval between $(a + b)/2$ and b; if it is negative, we search for a root between a and $(a + b)/2$; if it is zero, we have found a root. Iterating this until the value of f is very near zero will give a good approximation of a root of f.

We can improve this slightly by taking the line between $(a, f(a))$ and $(b, f(b))$, seeing where it crosses the x axis, and using this new point as the subdivision point.

If f happens to be differentiable, we can do somewhat better than this. We can evaluate f at a point, and evaluate its derivative there as well. Using these, we can compute the equation of the tangent line to f at the point. If the graph of f is close enough to the graph of this tangent line, then the place where the tangent line crosses the x axis will be a good approximation of a root of f (see Fig. A.7). If it is not a good enough approximation, we can use it as a starting point and iterate the process (see Fig. A.8).

If the initial guess is x_0, then the equation of the tangent line is

$$y - f(x_0) = f'(x_0)\,(x - x_0).$$

This crosses the x axis when $y = 0$, which happens at the point

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

In general, we can find the next point, x_{i+1}, from the point x_i by a corresponding formula, and repeat the process until a root is found. This process is called *Newton's method* or *Newton–Raphson iteration*.

Exercise: Apply Newton's method to the function $f(x) = x^2 - 2$, starting at $x = 1$.

Answer: $x_0 = 1$, $x_1 = 1.5$, $x_2 = 1.4166$, $x_3 = 1.4142\ldots$, and so on.

The method can fail by *cycling*. For example, it is possible that $x_2 = x_0$, and then the process will repeat itself forever without getting to a better approximation. For example, the function $f(x) = x^3 - 5x$ has a root at $x = 0$, but starting this iterative technique at $x_0 = 1$ will never find that root, because the subsequent choices will be $x_1 = -1$, $x_2 = 1$, and so on.

If the function f is sufficiently nice, the method can be guaranteed to succeed. In particular, if f has everywhere positive derivative and negative second derivative, the method will certainly converge to a root.

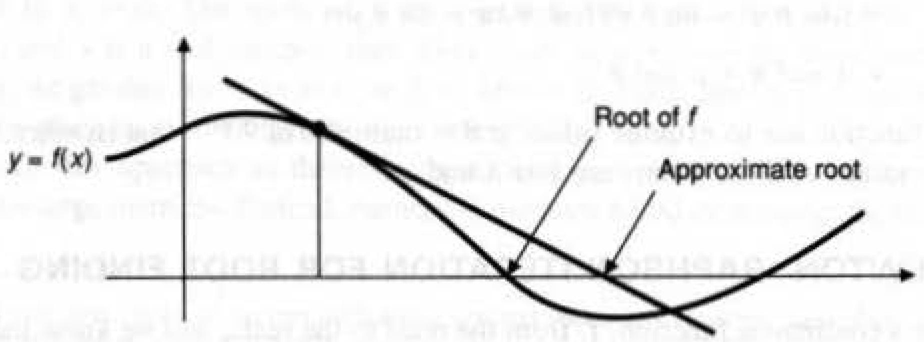


Fig. A.7 If the graph of the tangent line to a function is close enough to the graph of the function, the zero-crossing of the tangent line will be a good approximation to a zero-crossing of the function graph.

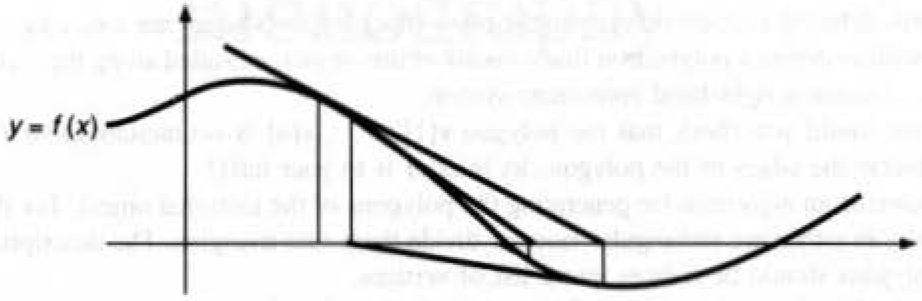


Fig. A.8 Iteration of the process described in Fig. A.7.

EXERCISES

A.1 Light reflects from a plane (in a simple model) according to the rule, "The angle of incidence is equal to the angle of reflection." If the normal to the plane is the vector **n**, and the ray from the light is described by the parametric ray $P + t\mathbf{v}$, what is the direction vector **u** for the reflected ray?

Answer: If we express **v** as a sum of two components—one in the direction of **n** and one perpendicular to **n**—we can easily describe **u**: It is the same as **v**, except with the component in the **n** direction negated. The component of **v** in the **n** direction is just $(\mathbf{v} \cdot \mathbf{n})\, \mathbf{n} / \| \mathbf{n} \|$, so the final result is that $\mathbf{u} = (\mathbf{v} - (\mathbf{v} \cdot \mathbf{n})\, \mathbf{n} / \| \mathbf{n} \|) - (\mathbf{v} \cdot \mathbf{n})\, \mathbf{n} / \| \mathbf{n} \| = \mathbf{v} - 2(\mathbf{v} \cdot \mathbf{n})\, \mathbf{n} / \| \mathbf{n} \|$. If **n** is a unit vector, this is just $\mathbf{v} - 2(\mathbf{v} \cdot \mathbf{n})\, \mathbf{n}$. (Note that the light ray is the opposite of the ray $\overline{L}$ in Chapter 16.)

A.2 Find a tranformation from the standard affine plane to itself that leaves h coordinates fixed, but transforms (x, y) coordinates so that, in any constant-h plane, the unit square $[0, 1] \times [0, 1]$ is sent to $[-1, 1] \times [-1, 1]$. What is wrong with the following purported solution to the problem?

"The space in which we are working is 3D, so we will specify the transformation by saying where three basis vectors get sent. Clearly,

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{goes to} \quad \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix},$$

and

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{goes to} \quad \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}.$$

Also,

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{goes to} \quad \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}.$$

So the matrix must be

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}."$$

Partial Answer: The assumption begin made in the purported solution is that the map is a *linear map* on $\mathbf{R}^3$. It is actually an affine map, and includes a translation.

A.3 You are given a list of vertices $v[1], \ldots, v[n]$ as xyz triples in $\mathbf{R}^3$, but with each z coordinate equal to zero, defining a closed polygon in the plane (the polygon's edges are $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$). You wish to define a polyhedron that consists of this object, extruded along the z axis from $z = 0$ to $z = 1$. Assume a right-hand coordinate system.

a. How would you check that the polygon $v[1], \ldots, v[n]$ is counterclockwise (i.e., as you traverse the edges of the polygon, its interior is to your left)?
b. Describe an algorithm for generating the polygons of the extruded object. For the "sides," you can either use rectangular faces or divide them into triangles. The descriptions of these polygons should be indices into a list of vertices.
c. Each of the faces given in part (b) consists of a list of vertices. Suppose that you walk around the boundary of a polygon, and the polygon interior is to your right. Did you choose your order for the vertices so that the exterior of the extruded polyhedron is always overhead? If not, modify your answer to part (b).
d. Each edge of the extruded polyhedron is part of two faces, so during the traversals in part (c), each edge is traversed twice. Are the two traversals always in the same direction? Always in opposite directions? Or is there no particular pattern?

A.4 Given that $P = (x_0, y_0)$ and $Q = (x_1, y_1)$ are points in the plane, show that the equation of the line between them is $(y_1 - y_0)\,x - (x_1 - x_0)\,y = y_1x_0 - x_1y_0$. This formulation is especially nice, because it provides a general form for lines in any direction, including vertical lines.

A.5 Given that $\mathbf{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ is a vector in the plane, show that $\mathbf{w} = \begin{bmatrix} -y \\ x \end{bmatrix}$ is orthogonal to it. This is sometimes called (in analogy with the corresponding case in 3D) the cross-product of a single vector in the plane. The cross-product of $n - 1$ vectors in $\mathbf{R}^n$ can also be defined.

A.6 If P, Q, and R are three points in the standard affine 3-space, then

$$\frac{1}{2}\| (\mathrm{Q} - \mathrm{P}) \times (\mathrm{R} - \mathrm{P}) \|$$

is the area of the triangle ΔPQR. If P, Q, and R all lie in the xy plane in the standard affine 3-space, then

$$\frac{1}{2} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot ((\mathrm{Q} - \mathrm{P}) \times (\mathrm{R} - \mathrm{P}))$$

gives the *signed area* of the triangle—it is positive if ΔPQR is a counterclockwise loop in the plane and negative otherwise. (Counterclockwise here means "counterclockwise as viewed from a point on the positive z axis, in a right-handed coordinate system.")

a. Find the signed area of the triangle with vertices $P = (0,0)$, $Q = (x_i, y_i)$, and $R = (x_{i+1}, y_{i+1})$.

   Answer: $\frac{1}{2}(x_i\, y_{i+1} - x_{i+1}\, y_i.)$

b. Suppose we have a polygon in the plane with vertices $v_1, \ldots, v_n$, ($v_n = v_1$), and $v_i = (x_i, y_i)$ for each i. Explain why the signed area of the polygon is exactly

$$\frac{1}{2} \sum_{i=1}^{n-1} x_i\, y_{i+1} - x_{i+1}\, y_i.$$

   Compare this with Eq. 11.2.

A.7 The Gram-Schmidt process described in Section A.3.6 for three vectors can be slightly simplified. After computing $\mathbf{w}_1$ and $\mathbf{w}_2$, we seek a third unit vector, $\mathbf{w}_3$ which is perpendicular to the first two. There are only two choices possible: $\mathbf{w}_3 = \pm\, \mathbf{w}_1 \times \mathbf{w}_2$. Show that the Gram-Schmidt process chooses the sign in this formula to be the same as the sign of $\mathbf{v}_3 \cdot (\mathbf{w}_1 \times \mathbf{w}_2)$. This implies that if you know that $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ is positively oriented, then you need not even check the sign: $\mathbf{w}_3 = \mathbf{w}_1 \times \mathbf{w}_2$.

Bibliography

What follows is an extensive bibliography in computer graphics. In addition to being a list of references from the various chapters, it is also a fine place to browse. Just looking at the titles of the books and articles can give you a good idea of where research in the field has been and where it is going.

Certain journals are referenced extremely frequently, and we have abbreviated them here. The most important of these are the ACM SIGGRAPH Conference Proceedings, published each year as an issue of *Computer Graphics*, and the *ACM Transactions on Graphics*. These two sources make up more than one-third of the bibliography.

Abbreviations

ACM TOG	*Association for Computing Machinery, Transactions on Graphics*
CACM	*Communications of the ACM*
CG & A	*IEEE Computer Graphics and Applications*
CGIP	*Computer Graphics and Image Processing*
CVGIP	*Computer Vision, Graphics, and Image Processing (formerly CGIP)*
FJCC	*Proceedings of the Fall Joint Computer Conference*
JACM	*Journal of the ACM*
NCC	*Proceedings of the National Computer Conference*
SJCC	*Proceedings of the Spring Joint Computer Conference*
SIGGRAPH 76	*Proceedings of SIGGRAPH '76 (Philadelphia, Pennsylvania, July 14–16, 1976). In Computer Graphics*, 10(2), Summer 1976, ACM SIGGRAPH, New York.
SIGGRAPH 77	*Proceedings of SIGGRAPH '77 (San Jose, California, July 20–22, 1977). In Computer Graphics*, 11(2), Summer 1977, ACM SIGGRAPH, New York.
SIGGRAPH 78	*Proceedings of SIGGRAPH '78 (Atlanta, Georgia, August 23–25, 1978). In Computer Graphics*, 12(3), August 1978, ACM SIGGRAPH, New York.

SIGGRAPH 79	*Proceedings of SIGGRAPH '79 (Chicago, Illinois, August 8–10, 1979). In Computer Graphics*, 13(2), August 1979, ACM SIGGRAPH, New York.
SIGGRAPH 80	*Proceedings of SIGGRAPH '80 (Seattle, Washington, July 14–18, 1980). In Computer Graphics*, 14(3), July 1980, ACM SIGGRAPH, New York.
SIGGRAPH 81	*Proceedings of SIGGRAPH '81 (Dallas, Texas, August 3–7, 1981). In Computer Graphics*, 15(3), August 1981, ACM SIGGRAPH, New York.
SIGGRAPH 82	*Proceedings of SIGGRAPH '82 (Boston, Massachusetts, July 26–30, 1982). In Computer Graphics*, 16(3), July 1982, ACM SIGGRAPH, New York.
SIGGRAPH 83	*Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25–29, 1983). In Computer Graphics*, 17(3), July 1983, ACM SIGGRAPH, New York.
SIGGRAPH 84	*Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23–27, 1984). In Computer Graphics*, 18(3), July 1984, ACM SIGGRAPH, New York.
SIGGRAPH 85	*Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26, 1985). In Computer Graphics*, 19(3), July 1985, ACM SIGGRAPH, New York.
SIGGRAPH 86	*Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18–22, 1986). In Computer Graphics*, 20(4), August 1986, ACM SIGGRAPH, New York.
SIGGRAPH 87	*Proceedings of SIGGRAPH '87 (Anaheim, California, July 27–31, 1987). In Computer Graphics*, 21(4), July 1987, ACM SIGGRAPH, New York.
SIGGRAPH 88	*Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1–5, 1988). In Computer Graphics*, 22(4), August 1988, ACM SIGGRAPH, New York.
SIGGRAPH 89	*Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989). In Computer Graphics*, 23(3), July 1989, ACM SIGGRAPH, New York.

ABIE89 Abi-Ezzi, S.S., *The Graphical Processing of B-Splines in a Highly Dynamic Environment*, Ph.D. Thesis Rensselaer Polytechnic Institute, Troy, NY, May 1989.

ABRA85 Abram, G., L. Westover, and T. Whitted, "Efficient Alias-Free Rendering Using Bit-Masks and Look-Up Tables," *SIGGRAPH 85*, 53–59.

ADOB85a Adobe Systems, Inc., *PostScript Language Tutorial and Cookbook*, Addison-Wesley, Reading, MA, 1985.

ADOB85b Adobe Systems, Inc., *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1985.

AKEL88 Akeley, K., and T. Jermoluk, "High-Performance Polygon Rendering," *SIGGRAPH 88*, 239–246.

AKEL89 Akeley, K., "The Silicon Graphics 4D/240GTX Superworkstation," *CG & A*, 9(4), July 1989, 71–83.

ALAV84 Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development," *CACM*, 27(6), June 1984, 556–563.

AMAN84 Amanatides, J., "Ray Tracing with Cones," *SIGGRAPH 84*, 129–135.

AMAN87 Amanatides, J. and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," in Maréchal, G., ed., *Eurographics 87: Proceedings of the European Computer Graphics Conference and Exhibition, Amsterdam, August 24–28, 1987*, North Holland, Amsterdam, 1987, 3–10.

AMBU86 Amburn, P., E. Grant, and T. Whitted, "Managing Geometric Complexity with Enhanced Procedural Models," *SIGGRAPH 86*, 189–195.

ANDE82 Anderson, D.P., "Hidden Line Elimination in Projected Grid Surfaces," *ACM TOG*, 1(4), October 1982, 274–288.

ANDE83 Anderson, D., "Techniques for Reducing Pen Plotting Time," *ACM TOG*, 2(3), July 1983, 197–212.

ANSI85a ANSI (American National Standards Institute), *American National Standard for Human Factors Engineering of Visual Display Terminal Workstations*, ANSI, Washington, DC, 1985.

ANSI85b ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Computer Graphics—Graphical Kernel System (GKS) Functional Description*, ANSI X3.124-1985, ANSI, New York, 1985.

ANSI88 ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, Archive File Format, Clear-Text Encoding of Archive File*, ANSI, X3.144-1988, ANSI, New York, 1988.

APGA88 Apgar, B., B. Bersack, and A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *SIGGRAPH 88*, 255–262.

APPE67 Appel, A., "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," *Proceedings of the ACM National Conference*, Thompson Books, Washington, DC, 1967, 387–393. Also in FREE80, 214–220.

APPE68 Appel, A., "Some Techniques for Shading Machine Renderings of Solids," *SJCC*, 1968, 37–45.

APPE79 Appel, A., F.J. Rohlf, and A.J. Stein, "The Haloed Line Effect for Hidden Line Elimination," *SIGGRAPH 79*, 151–157.

APPL85 Apple Computer, Inc., *Inside Macintosh*, Addison-Wesley, Reading, MA, 1985.

APPL87 Apple Computer, Inc., *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley, Reading, MA, 1987.

APT85 Apt, C., "Perfecting the Picture," *IEEE Spectrum*, 22(7), July 1985, 60–66.

ARDE89 Ardent Computer Corp. (now Stardent), *Doré Product Literature*, Sunnyvale, CA, 1989.

ARNO88 Arnold, D. B., and P.R. Bono, *CGM and CGI : Metafile and Interface Standards for Computer Graphics*, Springer-Verlag, Berlin, 1988.

ARVO86 Arvo, J., "Backward Ray Tracing," in A.H. Barr, ed., *Developments in Ray Tracing, Course Notes 12 for SIGGRAPH 86*, Dallas, TX, August 18–22, 1986.

ARVO87 Arvo, J., and D. Kirk, "Fast Ray Tracing by Ray Classification," *SIGGRAPH 87*, 55–64.

ATHE78 Atherton, P.R., K. Weiler, and D. Greenberg, "Polygon Shadow Generation," *SIGGRAPH 78*, 275–281.

ATHE81 Atherton, P.R., "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display," *SIGGRAPH 81*, 279–287.

ATHE83 Atherton, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," *SIGGRAPH 83*, 73–82.

ATKI84	Atkinson, H.H., I. Gargantini, and M.V.S. Ramanath, "Determination of the 3D Border by Repeated Elimination of Internal Surfaces," *Computing*, 32(4), October 1984, 279–295.
ATKI86	Atkinson, W., U.S. Patent 4,622,545, November 1986.
AYAL85	Ayala, D., P. Brunet, R. Juan, and I. Navazo, "Object Representation by Means of Nonminimal Division Quadtrees and Octrees," *ACM TOG*, 4(1), January 1985, 41–59.
BADL87	Badler, N.I., K.H. Manoochehri, and G. Walters, "Articulated Figure Positioning by Multiple Constraints," *CG & A*, 7(6), June 1987, 28–38.
BAEC69	Baecker, R.M., "Picture Driven Animation," *SJCC*, AFIPS Press, Montvale, NJ, 1969, 273–288.
BAEC87	Baecker, R., and B. Buxton, *Readings in Human-Computer Interaction*, Morgan Kaufmann, Los Altos, CA, 1987.
BALD85	Baldauf, D., "The Workhorse CRT: New Life," *IEEE Spectrum*, 22(7), July 1985, 67–73.
BANC77	Banchoff, T.F., and C.M. Strauss, *The Hypercube: Projections and Slicings*, Film, International Film Bureau, 1977.
BANC83	Banchoff, T., and J. Wermer, *Linear Algebra Through Geometry*, Springer-Verlag, New York, 1983.
BARN88a	Barnsley, M., A. Jacquin, F. Malassenet, L. Reuter, and A.D. Sloan, "Harnessing Chaos for Image Synthesis," *SIGGRAPH 88*, 131–140.
BARN88b	Barnsley, M., "Harnessing Chaos for Image Synthesis," Lecture at ACM SIGGRAPH '88 meeting in Atlanta, GA, August 1988.
BARR79	Barros, J., and H. Fuchs, "Generating Smooth 2–D Monocolor Line Drawings on Video Displays," *SIGGRAPH 79*, 260–269.
BARR84	Barr, A., "Global and Local Deformations of Solid Primitives," *SIGGRAPH 84*, 21–30.
BARR88	Barr, A., and R. Barzel, "A Modeling System Based on Dynamic Constraints," *SIGGRAPH 88*, 179–188.
BARR88b	Barry, P. and R. Goldman, "A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines," *SIGGRAPH 88*, 199–204.
BARR89a	Barr, A.H., ed., *Topics in Physically Based Modeling*, Addison-Wesley, Reading, MA, 1989.
BARS83	Barsky, B., and J. Beatty, "Local Control of Bias and Tension in Beta-Splines," *ACM TOG*, 2(2), April 1983, 109–134.
BARS85	Barsky, B., and T. DeRose, "The Beta2-Spline: A Special Case of the Beta-Spline Curve and Surface Representation," *CG & A*, 5(9), September 1985, 46–58. See also erratum in 7(3), March 1987, 15.
BARS87	Barsky, B., T. DeRose, and M. Dippé, *An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-spline Objects*, Report UCB/CSD 87/348, Department of Computer Science, University of California, Berkeley, CA, 1987.
BARS88	Barsky, B., *Computer Graphics and Geometric Modeling Using Beta-splines*, Springer-Verlag, New York, 1988.
BART87	Bartels, R., J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA, 1987.
BASS88	Bass, L., E. Hardy, K. Hoyt, M. Little, and R. Seacord, *Introduction to the Serpent User Interface Management System*, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, March 1988.

BAUM72 Baumgart, B.G., *Winged-edge Polyhedron Representation*, Technical Report STAN-CS-320, Computer Science Department, Stanford University, Palo Alto, CA, 1972.

BAUM74 Baumgart, B.G., *Geometric Modeling for Computer Vision*, Ph.D. Thesis, Report AIM-249, STAN-CS-74-463, Computer Science Department, Stanford University, Palo Alto, CA, October 1974.

BAUM75 Baumgart, B.G., ''A Polyhedron Representation for Computer Vision,'' *NCC 75*, 589–596.

BAUM89 Baum, D.R., H.E. Rushmeier, and J.M. Winget, ''Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors,'' *SIGGRAPH 89*, 325–334.

BAYE73 Bayer, B.E., ''An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures,'' in *Conference Record of the International Conference on Communications*, 1973, 26-11–26-15.

BEAT82 Beatty, J.C., and K.S. Booth, eds., *Tutorial: Computer Graphics*, Second Edition, IEEE Comp. Soc. Press, Silver Spring, MD, 1982.

BECK63 Beckmann, P., and A. Spizzichino, *The Scattering of Electromagnetic Waves from Rough Surfaces*, Macmillan, New York, 1963.

BEDF58 Bedford, R. and G. Wyszecki, "Wavelength Discrimination for Point Sources," *Journal of the Optical Society of America*, 48, 1958, 129-ff.

BENN84 Bennet, P.P., and S.A. Gabriel, ''System for Spatially Transforming Images,'' U. S. Patent 4,472,732, September 18, 1984.

BENT82 Benton, S.A., ''Survey of Holographic Stereograms,'' *Proceedings of SPIE*, 367, August 1982, 15–19.

BERG78 Bergeron, R., P. Bono, and J. Foley, ''Graphics Programming Using the Core System,'' *Computing Surveys (Special Issue on Graphics Standards)*, 10(4), December 1978, 389–443.

BERG86a Bergeron, P., ''A General Version of Crow's Shadow Volumes,'' *CG & A*, 6(9), September 1986, 17–28.

BERG86b Bergman, L., H. Fuchs, E. Grant, and S. Spach, ''Image Rendering by Adaptive Refinement,'' *SIGGRAPH 86*, 29–37.

BERK68 Berkeley Physics Course, *Waves*, Volume 3, McGraw-Hill, New York, 1968.

BERK82 Berk, T., L. Brownston, and A. Kaufman, ''A New Color-Naming System for Graphics Languages,'' *CG & A*, 2(3), May 1982, 37–44.

BERT81 Bertin, J., *Graphics and Graphics Information Processing*, de Gruyter, New York, 1981. Translated by Berg, W., and P. Scott from *La Graphique et le Traitement Graphique de l'Information*, Flammarion, Paris, 1977.

BERT83 Bertin, J., *Semiology of Graphics*, University of Wisconsin Press, Madison, WI, 1983. Translated by W. Berg from *Sémiologie Graphique*, Editions Gauthier-Villars, Paris; Editions Mouton & Cie, Paris-La Haye; and Ecole Pratique des Hautes Etudes, Paris, 1967.

BEWL83 Bewley, W., T. Roberts, D. Schroit, and W. Verplank, ''Human Factors Testing in the Design of Xerox's 8010 'Star' Office Workstation,'' in *Proceedings CHI'83 Human Factors in Computing Systems Conference*, ACM, New York, 1983, 72–77.

BEZI70 Bézier, P., *Emploi des Machines á Commande Numérique*, Masson et Cie, Paris, 1970. Translated by Forrest, A. R., and A. F. Pankhurst as Bézier, P., *Numerical Control —Mathematics and Applications*, Wiley, London, 1972.

BEZI74 Bézier, P., ''Mathematical and Practical Possibilities of UNISURF,'' in Barnhill, R. E., and R. F. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, New York, 1974.

BIER86a Bier, E., and M. Stone, "Snap-Dragging," *SIGGRAPH 86*, 233–240.

BIER86b Bier, E., "Skitters and Jacks: Interactive 3D Positioning Tools," in *Proceedings 1986 Workshop on Interactive 3D Graphics*, ACM, New York, 1987, 183–196.

BILL81 Billmeyer, F., and M. Saltzman, *Principles of Color Technology*, second edition, Wiley, New York, 1981.

BINF71 Binford, T., in *Visual Perception by Computer, Proceedings of the IEEE Conference on Systems and Control*, Miami, FL, December 1971.

BIRR61 Birren, R., *Creative Color*, Van Nostrand Reinhold, New York, 1961.

BISH60 Bishop, A., and M. Crook, *Absolute Identification of Color for Targets Presented Against White and Colored Backgrounds*. Report WADD TR 60-611, Wright Air Development Division, Wright Patterson AFB, Dayton, Ohio, 1960.

BISH86 Bishop, G., and D.M. Weimer, "Fast Phong Shading," *SIGGRAPH 86*, 103–106.

BLES82 Bleser, T., and J. Foley, "Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 309–314.

BLES86 Bleser, B., and J. Ward, "Human Factors Affecting the Problem of Machine Recognition of Hand-Printed Text," in *Computer Graphics '86 Conference Proceedings*, Volume 3, NCGA, Fairfax, VA, 1986, 498–514.

BLES88a Bleser, T., J. Sibert, and J. P. McGee, "Charcoal Sketching: Returning Control to the Artist," *ACM TOG*, 7(1), January 1988, 76–81

BLES88b Bleser, T., *TAE Plus Styleguide User Interface Description*, NASA Goddard Space Flight Center, Greenbelt, MD, 1988.

BLIN76 Blinn, J.F., and M.E. Newell, "Texture and Reflection in Computer Generated Images," *CACM*, 19(10), October 1976, 542–547. Also in BEAT82, 456–461.

BLIN77a Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," *SIGGRAPH 77*, 192–198. Also in FREE80, 316–322.

BLIN77b Blinn, J.F., "A Homogeneous Formulation for Lines in 3-Space," *SIGGRAPH 77*, 237–241.

BLIN78a Blinn, J.F., and M.E. Newell, "Clipping Using Homogeneous Coordinates," *SIGGRAPH 78*, 245–251.

BLIN78b Blinn, J.F., "Simulation of Wrinkled Surfaces," *SIGGRAPH 78*, 286–292.

BLIN78c Blinn, J.F., *Computer Display of Curved Surfaces*, Ph.D. Thesis, Department of Computer Science, University of Utah, Salt Lake City, UT, December 1978.

BLIN82a Blinn, J.F., "Light Reflection Functions for the Simulation of Clouds and Dusty Surfaces," *SIGGRAPH 82*, 21–29.

BLIN82b Blinn, J.F., "A Generalization of Algebraic Surface Drawing," *ACM TOG*, 1(3), July 1982, 235–256.

BLIN85 Blinn, J.F., "Systems Aspects of Computer Image Synthesis and Computer Animation," in *Image Rendering Tricks, Course Notes 12 for SIGGRAPH 85*, New York, July 1985.

BLIN88 Blinn, J.F., "Me and My (Fake) Shadow," *CG & A*, 9(1), January 1988, 82–86.

BLIN89a Blinn, J.F., "What We Need Around Here is More Aliasing," *CG & A*, 9(1), January 1989, 75–79.

BLIN89b Blinn, J.F., "Return of the Jaggy," *CG & A*, 9(2), March 1989, 82–89.

BLOO88 Bloomenthal, J., "Polygonisation of Implicit Surfaces," *Computer Aided Geometric Design*, 5, 1988, 341-355.

BOLT80 Bolt, R.A., "'Put-That-There': Voice and Gesture at the Graphics Interface," *SIGGRAPH 80*, 262–270.

BOLT84 Bolt, R.A., *The Human Interface: Where People and Computers Meet*, Lifetime Learning Press, Belmont, CA, 1984.

BORD89 Borden, B.S., "Graphics Processing on a Graphics Supercomputer," *CG & A*, 9(4), July 1989, 56–62.

BORN79 Borning, A., "Thinglab—A Constraint-Oriented Simulation Laboratory," Technical Report SSl-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, July 1979.

BORN86a Borning, A., and R. Duisberg, "Constraint-Based Tools for Building User Interfaces," *ACM TOG*, 5(4), October 1986, 345–374.

BORN86b Borning, A., "Defining Constraints Graphically," in *SIGCHI '86 Conference Proceedings*, ACM, New York, 1986, 137–143.

BOUK70a Bouknight, W.J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations," *CACM*, 13(9), September 1970, 527–536. Also in FREE80, 292–301.

BOUK70b Bouknight, W.J., and K.C. Kelly, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC*, AFIPS Press, Montvale, NJ, 1970, 1–10.

BOUV85 Bouville, C., "Bounding Ellipsoids for Ray-Fractal Intersection," *SIGGRAPH 85*, 45–52.

BOYN79 Boynton, R. M., *Human Color Vision*, Holt, Rinehart, and Winston, New York, 1979.

BOYS82 Boyse, J.W., and J.E. Gilchrist, "GMSolid: Interactive Modeling for Design and Analysis of Solids," *CG & A*, 2(2), March 1982, 27–40.

BÖHM80 Böhm, W., "Inserting New Knots into B-spline Curves," *Computer Aided Design*, 12(4), July 1980, 199–201.

BÖHM84 Böhm, W., G. Farin, and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design*, 1(1), July 1984, 1–60.

BRAI78 Braid, I.C., R.C. Hillyard, and I.A. Stroud, *Stepwise Construction of Polyhedra in Geometric Modelling*, CAD Group Document No. 100, Cambridge University, Cambridge, England, 1978. Also in K.W. Brodlie, ed., *Mathematical Methods in Computer Graphics and Design*, Academic Press, New York, 1980, 123–141.

BRES65 Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal*, 4(1), 1965, 25–30.

BRES77 Bresenham, J.E. "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *Communications of the ACM*, 20(2), February 1977, 100–106.

BRES83 Bresenham, J.E., D.G. Grice, and S.C. Pi, "Bi-Directional Display of Circular Arcs," US Patent 4,371,933, February 1, 1983.

BREW77 Brewer, H., and D. Anderson, "Visual Interaction with Overhauser Curves and Surfaces," *SIGGRAPH 77*, 132–137.

BRIG74 Brigham, E.O., *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

BRIT78 Britton, E., J. Lipscomb, and M. Pique, "Making Nested Rotations Convenient for the User," *SIGGRAPH 78*, 222–227.

BROO88 Brooktree Corporation, *Product Databook 1988*, Brooktree Corporation, San Diego, CA, 1987.

BROT84 Brotman, L.S., and N.I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *CG & A*, 4(10), October 1984, 5–12.

BROW64 Brown, R., "On-Line Computer Recognition of Hand-Printed Characters," *IEEE Trans. Computers*, Vol. EC-13(12), December 1964, 750–752.

BROW82 Brown, C.M., "PADL-2: A Technical Summary," *CG & A*, 2(2), March 1982, 69–84.

BUIT75 Bui-Tuong, Phong, "Illumination for Computer Generated Pictures," *CACM*, 18(6), June 1975, 311–317. Also in BEAT82, 449–455.

BUNK89 Bunker, M., and R. Economy, *Evolution of GE CIG Systems*, SCSD Document, General Electric Company, Daytona Beach, FL, 1989.

BURT74 Burton, R.P., and I. E. Sutherland, "Twinkle Box: A Three-Dimensional Computer Input Device," *NCC 1974*, AFIPS Press, Montvale, NJ, 1974, 513–520.

BURT76 Burtnyk, N., and M. Wein, "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation," *CACM*, 19(10), October 1976, 564–569

BUTL79 Butland, J., "Surface Drawing Made Simple," *Computer-Aided Design*, 11(1), January 1979, 19–22.

BUXT83 Buxton, W., M.R. Lamb, D. Sherman, and K.C. Smith, "Towards a Comprehensive User Interface Management System," *SIGGRAPH 83*, 35–42.

BUXT85 Buxton, W., R. Hill, and P. Rowley, "Issues and Techniques in Touch-Sensitive Tablet Input," *SIGGRAPH 85*, 215–224.

BUXT86 Buxton, W., "There's More to Interaction Than Meets the Eye: Issues in Manual Input," in Norman, D., and S. Draper, eds., *User-Centered System Design*, Lawrence Erlbaum, Hillsdale, NJ, 1986, 319–337. Also in BAEC 87, 366–375.

BYTE85 *BYTE Magazine*, 10(5), May 1985, 151–167.

CABR87 Cabral, B., N. Max, and R. Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps," *SIGGRAPH 87*, 273–281.

CACM84 "Computer Graphics Comes of Age: An Interview with Andries van Dam." *CACM*, 27(7), July 1984, 638–648.

CALL88 Callahan, J., D. Hopkins, M. Weiser, and B. Shneiderman, "An Empirical Comparison of Pie vs. Linear Menus," in *Proceedings of CHI 1988*, ACM, New York, 95–100.

CARD78 Card, S., W. English, and B. Burr, "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection of a CRT," *Ergonomics*, 21(8), August 1978, 601–613.

CARD80 Card, S., T. Moran, and A. Newell, "The Keystroke-Level Model for User Performance Time with Interactive Systems," *CACM*, 23(7), July 1980, 398–410.

CARD82 Card, S., "User Perceptual Mechanisms in the Search of Computer Command Menus," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, March 1982, 20–24.

CARD83 Card, S., T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

CARD85 Cardelli, L., and R. Pike, "Squeak: A Language for Communicating with Mice," *SIGGRAPH 85*, 199–204.

CARD88 Cardelli, L., "Building User Interfaces by Direct Manipulation," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 152–166.

CARL78 Carlbom, I., and J. Paciorek, "Planar Geometric Projections and Viewing Transformations," *Computing Surveys*, 10(4), December 1978, 465–502.

CARL85 Carlbom, I., I. Chakravarty, and D. Vanderschel, "A Hierarchical Data Structure for Representing the Spatial Decomposition of 3-D Objects," *CG & A*, 5(4), April 1985, 24–31.

CARL87 Carlbom, I., "An Algorithm for Geometric Set Operations Using Cellular Subdivision Techniques," *CG & A*, 7(5), May 1987, 44–55.

CARP84 Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method," *SIGGRAPH 84*, 103–108.

CATM72 Catmull, E., "A System for Computer Generated Movies," in *Proc. ACM Annual Conference*, ACM, New York, NY, August 1972, 422–431.

CATM74a Catmull, E., and R. Rom, "A Class of Local Interpolating Splines," in Barnhill, R., and R. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, San Francisco, 1974, 317–326.

CATM74b Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

CATM75 Catmull, E., "Computer Display of Curved Surfaces," in *Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures*, May 1975. Also in FREE80, 309–315.

CATM78a Catmull, E., "The Problems of Computer-Assisted Animation," *SIGGRAPH 78*, 348–353.

CATM78b Catmull, E., "A Hidden-Surface Algorithm with Anti-Aliasing," *SIGGRAPH 78*, 6–11. Also in BEAT82, 462–467.

CATM79 Catmull, E., "A Tutorial on Compensation Tables," *SIGGRAPH 79*, 279–285.

CATM80 Catmull, E., and A.R. Smith, "3-D Transformations of Images in Scanline Order," *SIGGRAPH 80*, 279–285.

CGA82 Special Issue on Modeling the Human Body for Animation, *CG & A*, 2(11) N. Badler, ed., November 1982, 1–81.

CHAP72 Chapanis, A., and R. Kinkade, "Design of Controls," in Van Cott, H., and R. Kinkade, eds., *Human Engineering Guide to Equipment Design*, U.S. Government Printing Office, 1972.

CHAS81 Chasen, S.H., "Historical Highlights of Interactive Computer Graphics," *Mechanical Engineering*, 103, ASME, November 1981, 32–41.

CHEN88 Chen, M., J. Mountford, and A. Sellen, "A Study in Interactive 3-D Rotation Using 2-D Control Devices," *SIGGRAPH 88*, 121–129.

CHIN89 Chin, N., and S. Feiner, "Near Real-Time Shadow Generation Using BSP Trees," *SIGGRAPH 89*, 99–106.

CHIN90 Chin, N., *Near Real-Time Object-Precision Shadow Generation Using BSP Trees*, M.S. Thesis, Department of Computer Science, Columbia University, New York, 1990.

CHRI75 Christ, R., "Review and Analysis of Color Coding for Visual Display," *Human Factors*, 17(6), December 1975, 542–570.

CHUN89 Chung, J.C., *et al.*, "Exploring Virtual Worlds with Head-Mounted Displays," *Proc. SPIE Meeting on Non-Holographic True 3-Dimensional Display Technologies*, 1083, Los Angeles, Jan. 15-20, 1989.

CLAR76 Clark, J.H., "Hierarchical Geometric Models for Visible Surface Algorithms," *CACM*, 19(10), October 1976, 547–554. Also in BEAT82, 296–303.

CLAR79 Clark, J., "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces," abstract in *SIGGRAPH 79*, 174. Also in Whitted, T., and R. Cook, eds., *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, TX, August 1986. Also in JOY88, 88–93.

CLAR80 Clark, J., and M. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *Lambda (VLSI Design)*, 1(3), Q4 1980, 40–45.

CLAR82 Clark, J., "The Geometry Engine: A VLSI Geometry System for Graphics," *SIGGRAPH 82*, 127–133.

CLEA83 Cleary, J., B. Wyvill, G. Birtwistle, and R. Vatti, "Design and Analysis of a Parallel Ray Tracing Computer," *Proceedings of Graphics Interface '83*, May 1983, 33–34.

CLEV83 Cleveland, W., and R. McGill, "A Color-Caused Optical Illusion on a Statistical Graph," *The American Statistician*, 37(2), May 1983, 101–105.

CLEV84 Cleveland, W., and R. McGill, "Graphical Perception: Theory, Experimentation and Application to the Development of Graphical Methods," *Journal of the American Statistical Association*, 79(387), September 1984, 531–554.

CLEV85 Cleveland, W., and R. McGill, "Graphical Perception and Graphical Methods for Analyzing Scientific Data," *Science*, 229, August 30, 1985, 828–833.

COHE80 Cohen, E., T. Lyche, and R. Riesenfeld, "Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics," *CGIP*, 14(2), October 1980, 87–111.

COHE83 Cohen, E., "Some Mathematical Tools for a Modeler's Workbench," *CG & A*, 3(7), October 1983, 63–66.

COHE85 Cohen, M.F., and D.P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *SIGGRAPH 85*, 31–40.

COHE86 Cohen, M.F., D.P. Greenberg, D.S. Immel, and P.J. Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis," *CG & A*, 6(3), March 1986, 26–35.

COHE88 Cohen, M.F., S.E. Chen, J.R. Wallace, and D.P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *SIGGRAPH 88*, 75–84.

CONR85 Conrac Corporation, *Raster Graphics Handbook*, second edition, Van Nostrand Reinhold, New York, 1985.

COOK82 Cook, R., and K. Torrance, "A Reflectance Model for Computer Graphics," *ACM TOG*, 1(1), January 1982, 7–24.

COOK84a Cook, R.L., "Shade Trees," *SIGGRAPH 84*, 223–231.

COOK84b Cook, R.L., T. Porter, and L. Carpenter, "Distributed Ray Tracing," *SIGGRAPH 84*, 137–145.

COOK86 Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM TOG*, 5(1), January 1986, 51–72.

COOK87 Cook, R.L., L. Carpenter, and E. Catmull, "The Reyes Image Rendering Architecture," *SIGGRAPH 87*, 95–102.

COON67 Coons, S. A., *Surfaces for Computer Aided Design of Space Forms*, MIT Project Mac, TR-41, MIT, Cambridge, MA, June 1967.

COSS89 Cossey, G., *Prototyper*, SmethersBarnes, Portland, OR, 1989.

COWA83 Cowan, W., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates," *SIGGRAPH 83*, 315–321.

CROC84 Crocker, G.A., "Invisibility Coherence for Faster Scan-Line Hidden Surface Algorithms," *SIGGRAPH 84*, 95–102.

CROW77a Crow, F.C., "Shadow Algorithms for Computer Graphics," *SIGGRAPH 77*, 242–247. Also in BEAT82, 442–448.

CROW77b Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," *CACM*, 20(11), November 1977, 799–805.

CROW78 Crow, F., "The Use of Grayscale for Improved Raster Display of Vectors and Characters," *SIGGRAPH 78*, 1–5.

CROW81 Crow, F.C., "A Comparison of Antialiasing Techniques," *CG & A*, 1(1), January 1981, 40–48.

CROW84 Crow, F.C., "Summed-Area Tables for Texture Mapping," *SIGGRAPH 84*, 207–212.

CYRU78 Cyrus, M. and J. Beck, "Generalized Two- and Three-Dimensional Clipping," *Computers and Graphics*, 3(1), 1978, 23–28.

DALL80 Dallas, W.J., "Computer Generated Holograms," in *The Computer in Optical Research*, Frieden, B.R., ed., Springer-Verlag, New York, 1980, 291–366.

DASI89 Da Silva, D., *Raster Algorithms for 2D Primitives*, Master's Thesis, Computer Science Department, Brown University, Providence, RI, 1989.

DEBO78 de Boor, C., *A Practical Guide to Splines*, Applied Mathematical Sciences Volume 27, Springer-Verlag, New York, 1978.

DECA59 de Casteljau, F., *Outillage Méthodes Calcul*, André Citroën Automobiles SA, Paris, 1959.

DEER88 Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *SIGGRAPH 88*, 21–30.

DELA88 Delany, H.C., "Ray Tracing on a Connection Machine," in *Proceedings of the 1988 International Conference on Supercomputing*, July 4–8, 1988, St. Malo, France, 659–664.

DEME80 Demetrescu, S., *A VLSI-Based Real-Time Hidden-Surface Elimination Display System*, Master's Thesis, Department of Computer Science, California Institute of Technology, Pasadena, CA, May 1980.

DEME85 Demetrescu, S., "High Speed Image Rasterization Using Scan Line Access Memories," in *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, MD, Computer Science Press, 221–243.

DEYO89 Deyo, R., and D. Ingebretson, "Notes on Real-Time Vehicle Simulation," in *Implementing and Interacting with Real-Time Microworlds, Course Notes 29 for SIGGRAPH 89*, Boston, MA, August 1989.

DIGI89 Digital Equipment Corporation, *DEC XUI Style Guide*, Digital Equipment Corporation, Maynard, MA, 1989.

DIPP84 Dippé, M. and J. Swensen, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *SIGGRAPH 84*, 149–158.

DIPP85 Dippé, M., and E.H. Wold, "Antialiasing through Stochastic Sampling," *SIGGRAPH 85*, 69–78.

DOCT81 Doctor, L., and J. Torborg, "Display Techniques for Octree-Encoded Objects," *CG & A*, 1(3), July 1981, 29–38.

DONA88 Donato, N., and R. Rocchetti, "Techniques for Manipulating Arbitrary Regions," in *Course Notes 11 for SIGGRAPH 88*, Atlanta, GA, August, 1988.

DREB88 Drebin, R.A., L. Carpenter, and P. Hanrahan, "Volume Rendering," *SIGGRAPH 88*, 65–74.

DUFF79 Duff, T., "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays," *SIGGRAPH 79*, 270–275.

DURB88 Durbeck, R., and S. Sherr, eds., *Output Hardcopy Devices*, Academic Press, New York, 1988.

DUVA90 Duvanenko, V., W.E. Robbins, R.S. Gyurcsik, "Improved Line Segment Clipping," *Dr. Dobb's Journal*, July 1990, 36–45, 98–100.

DVOŘ43 Dvořák, A., "There is a Better Typewriter Keyboard," *National Business Education Quarterly*, 12(2), 1943, 51–58.

ELLS89 Ellsworth, D., *Pixel-Planes 5 Rendering Control*, Tech. Rep. TR89-003, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1989.

EMER03 Emerson, R., "Essays: First Series—Self Reliance," in *The Complete Works of Ralph Waldo Emerson*, Houghton Mifflin, Boston, MA, 1903.

ENCA72 Encarnacao, J., and W. Giloi, ''PRADIS—An Advanced Programming System for 3-D-Display,'' *SJCC*, AFIPS Press, Montvale, NJ, 1972, 985–998.

ENDE87 Enderle, G, K. Kansy, and G. Pfaff, *Computer Graphics Programming*, second edition, Springer-Verlag, Berlin, 1987.

ENGE68 Engelbart, D.C., and W.K. English, *A Research Center for Augmenting Human Intellect*, *FJCC* , Thompson Books, Washington, D.C., 395.

ENGL89 England, N., ''Evolution of High Performance Graphics Systems,'' in *Proceedings of Graphics Interface '89*, Canadian Information Processsing Society (Morgan Kauffman in U.S.), 1989.

EVAN81 Evans, K., P. Tanner, and M. Wein, ''Tablet-Based Valuators that Provide One, Two or Three Degrees of Freedom,'' *SIGGRAPH 81*, 91–97.

EVAN89 Evans & Sutherland Computer Corporation, *The Breadth of Visual Simulation Technology*, Evans & Sutherland Computer Corporation, Salt Lake City, UT, 1989.

EYLE88 Eyles, J., J. Austin, H. Fuchs, T. Greer, and J. Poulton, ''Pixel-Planes 4: A Summary,'' in *Advances in Computer Graphics Hardware II* (1987 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, 183-208.

FARI86 Farin, G., ''Triangular Bernstein-Bézier Patches,'' *Computer Aided Geometric Design*, 3(2), August 1986, 83–127.

FARI88 Farin, G., *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, New York, 1988.

FAUX79 Faux, I. D., and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Wiley, New York, 1979.

FEIB80 Feibush, E.A., M. Levoy, and R.L. Cook, ''Synthetic Texturing Using Digital Filters,'' *SIGGRAPH 80*, 294–301.

FEIN82a Feiner, S., S. Nagy, and A. van Dam, ''An Experimental System for Creating and Presenting Interactive Graphical Documents,'' *ACM TOG*, 1(1), January 1982, 59–77.

FEIN82b Feiner, S., D. Salesin, and T. Banchoff, ''DIAL: A Diagrammatic Animation Language,'' *CG & A*, 2(7), September 1982, 43–54.

FEIN85 Feiner, S., ''APEX: An Experiment in the Automated Creation of Pictorial Explanations,'' *CG & A*, 5(11), November 1985, 29–38.

FEIN88 Feiner, S., ''A Grid-Based Approach to Automating Display Layout,'' in *Proceedings of Graphics Interface '88*, Edmonton, Canada, June 1988, 192–197.

FERG64 Ferguson, J., ''Multivariate Curve Interpolation,'' *JACM*, 11(2), April 1964, 221–228.

FIEL86 Field, D., ''Algorithms for Drawing Anti-Aliased Circles and Ellipses,'' *CGVIP*, 33(1), January 1986, 1–15.

FISH84 Fishkin, K.P., and B.A. Barsky, ''A Family of New Algorithms for Soft Filling,'' *SIGGRAPH 84*, 235–244.

FISH86 Fisher, S.S., M. McGreevy, J. Humphries, and W. Robinett, ''Virtual Environment Display System,'' in *Proceedings of the 1986 Chapel Hill Workshop on Interactive 3D Graphics*, Chapel Hill, NC, 1986, 77–87.

FITT54 Fitts, P., ''The Information Capacity of the Human Motor System in Controlling Amplitude of Motion,'' *Journal of Experimental Psychology*, 47(6), June 1954, 381–391.

FIUM89 Fiume, E.L., *The Mathematical Structure of Raster Graphics*, Academic Press, San Diego, 1989.

FLEC87 Flecchia, M., and R. Bergeron, ''Specifying Complex Dialogs in Algae,'' in *Proceedings of CHI + GI '87*, ACM, New York, 1987, 229–234.

FLOY75 Floyd, R., and Steinberg, L., "An Adaptive Algorithm for Spatial Gray Scale," in *Society for Information Display 1975 Symposium Digest of Technical Papers*, 1975, 36.

FOLE71 Foley, J., "An Approach to the Optimum Design of Computer Graphics Systems," *CACM*, 14 (6), June 1971, 380–390.

FOLE74 Foley, J., and V. Wallace, "The Art of Natural Man–Machine Communication," *Proceedings IEEE*, 62(4), April 1974, 462–470.

FOLE76 Foley, J., "A Tutorial on Satellite Graphics Systems," *IEEE Computer* 9(8), August 1976, 14–21.

FOLE82 Foley, J., and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

FOLE84 Foley, J., V. Wallace, and P. Chan, "The Human Factors of Computer Graphics Interaction Techniques," *CG & A*, 4(11), November 1984, 13–48.

FOLE87a Foley, J., "Interfaces for Advanced Computing," *Scientific American*, 257(4), October 1987, 126–135.

FOLE87b Foley, J., W. Kim, and C. Gibbs, "Algorithms to Transform the Formal Specification of a User Computer Interface," in *Proceedings INTERACT '87, 2nd IFIP Conference on Human-Computer Interaction*, Elsevier Science Publishers, Amsterdam, 1987, 1001-1006.

FOLE87c Foley, J., and W. Kim, "ICL—The Image Composition Language," *CG & A*, 7(11), November 1987, 26–35.

FOLE88 Foley, J., W. Kim, and S. Kovacevic, "A Knowledge Base for User Interface Management System," in *Proceedings of CHI '88–1988 SIGCHI Computer-Human Interaction Conference*, ACM, New York, 1988, 67-72.

FOLE89 Foley, J., W. Kim, S. Kovacevic, and K. Murray, "Defining Interfaces at a High Level of Abstraction," *IEEE Software*, 6(1), January 1989, 25–32.

FORR79 Forrest, A. R., "On the Rendering of Surfaces," *SIGGRAPH 79*, 253–259.

FORR80 Forrest, A. R., "The Twisted Cubic Curve: A Computer-Aided Geometric Design Approach," *Computer Aided Design*, 12(4), July 1980, 165–172.

FORR85 Forrest, A. R., "Antialiasing in Practice", in Earnshaw, R. A., ed., *Fundamental Algorithms for Computer Graphics,* NATO ASI Series F: Computer and Systems Sciences, Vol. 17, Springer-Verlag, New York, 1985, 113–134.

FORS88 Forsey, D.R., and R.H. Bartels, "Hierarchical B-spline Refinement," *SIGGRAPH 88*, 205–212.

FOUR82 Fournier, A., D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models, " *CACM*, 25(6), June 1982, 371–384.

FOUR86 Fournier, A., and W.T. Reeves, "A Simple Model of Ocean Waves," *SIGGRAPH 86*, 75–84.

FOUR88 Fournier, A. and D. Fussell, "On the Power of the Frame Buffer," *ACM TOG*, 7(2), April 1988, 103–128.

FRAN81 Franklin, W.R., "An Exact Hidden Sphere Algorithm that Operates in Linear Time," *CGIP*, 15(4), April 1981, 364–379.

FREE80 Freeman, H. ed., *Tutorial and Selected Readings in Interactive Computer Graphics*, IEEE Comp. Soc. Press, Silver Spring, MD, 1980.

FRIE85 Frieder, G., D. Gordon, and R. Reynolds, "Back-to-Front Display of Voxel-Based Objects," *CG & A*, 5(1), January 1985, 52–60.

FROM84 Fromme, F., "Improving Color CAD Systems for Users: Some Suggestions from Human Factors Studies," *IEEE Design and Test of Computers*, 1(1), February 1984, 18–27.

FUCH77a Fuchs, H., J. Duran, and B. Johnson, "A System for Automatic Acquisition of Three-Dimensional Data," in *Proceedings of the 1977 NCC*, AFIPS Press, 1977, 49–53.

FUCH77b Fuchs, H., "Distributing a Visible Surface Algorithm over Multiple Processors," *Proceedings of the ACM Annual Conference*, Seattle, WA, October 1977, 449–451.

FUCH79 Fuchs, H., and B. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics," *Proceedings of the 6th ACM-IEEE Symposium on Computer Architecture*, Philadelphia, PA, April 1979, 58–67.

FUCH80 Fuchs, H., Z.M. Kedem, and B.F. Naylor, "On Visible Surface Generation by A Priori Tree Structures," *SIGGRAPH 80*, 124–133.

FUCH81 Fuchs, H. and J. Poulton, "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, 2(3), Q3 1981, 20–28.

FUCH82 Fuchs, H., S.M. Pizer, E.R. Heinz, S.H. Bloomberg, L. Tsai, and D.C. Strickland, "Design of and Image Editing with a Space-Filling Three-Dimensional Display Based on a Standard Raster Graphics System," *Proceedings of SPIE*, 367, August 1982, 117–127.

FUCH83 Fuchs, H., G.D. Abram, and E.D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *SIGGRAPH 83*, 65–72.

FUCH85 Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *SIGGRAPH 85*, 111–120.

FUCH89 Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *SIGGRAPH 89*, 79–88.

FUJI85 Fujimura, K., and Kunii, T. L., "A Hierarchical Space Indexing Method," in Kunii, T. L., ed., *Computer Graphics: Visual Technology and Art , Proceedings of Computer Graphics Tokyo '85 Conference*, Springer-Verlag, 1985, 21–34.

FUSS82 Fussell, D., and B. D. Rathi, "A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons," in *Proceedings of Graphics Interface '82*, Toronto, May 1982, 373–380.

GAIN84 Gaines, B., and M. Shaw, *The Art of Computer Conversation*, Prentice-Hall International, Englewood Cliffs, NJ, 1984.

GALI69 Galimberti, R., and U. Montanari, "An Algorithm for Hidden Line Elimination," *CACM*, 12(4), April 1969, 206–211.

GARD84 Gardner, G.Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces," *SIGGRAPH 84*, 11–20.

GARD85 Gardner, G.Y., "Visual Simulation of Clouds," *SIGGRAPH 85*, 297–303.

GARG82 Gargantini, I., "Linear Octtrees for Fast Processing of Three-Dimensional Objects," *CGIP*, 20(4), December 1982, 365–374.

GARG86 Gargantini, I., T. Walsh, and O. Wu, "Viewing Transformations of Voxel-Based Objects via Linear Octrees," *CG & A*, 6(10), October 1986, 12–21.

GARR80 Garrett, M., *A Unified Non-Procedural Environment for Designing and Implementing Graphical Interfaces to Relational Data Base Management Systems*, Ph.D. dissertation, Technical Report GWU-EE/CS-80-13, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC, 1980.

GARR82 Garrett, M., and J. Foley, "Graphics Programming Using a Database System with Dependency Declarations," *ACM TOG*, 1(2), April 1982, 109–128.

GHAR88 Gharachorloo, N., S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu, and C. Zoulas, "Subnanosecond Pixel Rendering with Million Transistor Chips," *SIGGRAPH 88*, 41–49.

GHAR89 Gharachorloo, N., S. Gupta, R.F. Sproull, and I.E. Sutherland, "A Characterization of Ten Rasterization Techniques," *SIGGRAPH 89*, 355–368.

GILO78 Giloi, W.K., *Interactive Computer Graphics—Data Structures, Algorithms, Languages*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

GINS83 Ginsberg, C.M., and D. Maxwell, "Graphical Marionette," in *Proceedings of the SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception*, Toronto, April 4–6, 1983, 172–179.

GIRA85 Girard, M., and A.A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures," *SIGGRAPH 85*, 263–270.

GIRA87 Girard, M., "Interactive Design of 3D Computer-Animated Legged Animal Motion," *CG & A*, 7(6), June 1987, 39–51.

GLAS84 Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *CG & A*, 4(10), October 1984, 15–22.

GLAS86 Glassner, A.S., "Adaptive Precision in Texture Mapping," *SIGGRAPH 86*, 297–306.

GLAS88 Glassner, A., "Spacetime Raytracing for Animation," *CG & A*, 8(2), March 1988, 60–70.

GLAS89 Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989.

GOLD71 Goldstein, R.A., and R. Nagel, "3-D Visual Simulation," *Simulation*, 16(1), January 1971, 25–31.

GOLD76 Goldberg, A., and Kay, A., *SMALLTALK-72 Instruction Manual*, Learning Research Group, Xerox Palo Alto Research Center, Palo Alto, CA, March 1976.

GOLD80 Goldstein, H., *Classical Mechanics*, Addison-Wesley, Reading, MA, 1980.

GOLD83 Goldberg, A., and D. Robson, *SmallTalk 80: The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983.

GOLD84 Goldwasser, S.M., "A Generalized Object Display Processor Architecture," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, Ann Arbor, MI, June 5–7, 1984, *SIGARCH Newsletter*, 12(3), June 1984, 38–47.

GOLD86 Goldfeather, J., J.P.M. Hultquist, and H. Fuchs, "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System," *SIGGRAPH 86*, 107–116.

GOLD87 Goldsmith, J., and J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing," *CG & A*, 7(5), May 1987, 14–20.

GOLD88 Goldwasser, S.M., R.A. Reynolds, D.A. Talton, and E.S. Walsh, "Techniques for the Rapid Display and Manipulation of 3-D Biomedical Data," *Comp. Med. Imag. and Graphics*, 12(1), 1988, 1–24.

GOLD89 Goldfeather, J., S. Molnar, G. Turk, and H. Fuchs, "Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning," *CG & A*, 9(3), May 1989, 20–28.

GONZ87 Gonzalez, R., and P. Wintz, *Digital Image Processing*, second edition, Addison-Wesley, Reading, MA, 1987.

GORA84 Goral, C.M., K.E. Torrance, D.P. Greenberg, and B. Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *SIGGRAPH 84*, 213–222.

GORI87 Goris, A., B. Fredrickson, and H.L. Baeverstad, Jr., "A Configurable Pixel Cache for Fast Image Generation," *CG & A*, 7(3), March 1987, 24–32.

GOSL89 Gosling, J., personal communication, March 1989.

GOSS88 Gossard, D., R. Zuffante, and H. Sakurai, "Representing Dimensions, Tolerances, and Features in MCAE Systems," *CG & A*, 8(2), March 1988, 51–59.

GOUR71 Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. on Computers*, C-20(6), June 1971, 623–629. Also in FREE80, 302–308.

GREE85a Green, M., "The University of Alberta User Interface Management System," *SIGGRAPH 85*, 205–213.

GREE85b Green, M., *The Design of Graphical User Interfaces*, Technical Report CSRI-170, Department of Computer Science, University of Toronto, Toronto, 1985.

GREE85c Greene, R., "The Drawing Prism: A Versatile Graphic Input Device," *SIGGRAPH 85*, 103–110.

GREE86 Greene, N., "Environment Mapping and Other Applications of World Projections," *CG & A*, 6(11), November 1986, 21–29.

GREE87a Green, M., "A Survey of Three Dialog Models," *ACM TOG*, 5(3), July 1987, 244–275.

GREE87b Greenstein, J. and L. Arnaut, "Human Factors Aspects of Manual Computer Input Devices," in Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987, 1450–1489.

GREG66 Gregory, R.L., *Eye and Brain— The Psychology of Seeing*, McGraw-Hill, New York, 1966.

GREG70 Gregory, R.L., *The Intelligent Eye*, McGraw-Hill, London, 1970.

GRIM89 Grimes, J., L. Kohn, and R. Bharadhwaj, "The Intel i860 64-Bit Processor: A General-Purpose CPU with 3D Graphics Capabilities," *CG & A*, 9(4), July 1989, 85–94.

GSPC77 Graphics Standards Planning Committee. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH." *Computer Graphics*, 11(3), Fall 1977.

GSPC79 Graphics Standards Planning Committee. "Status Report of the Graphics Standards Planning Committee." *Computer Graphics*, 13(3), August 1979.

GTCO82 GTCO Corporation, *DIGI-PAD 5 User's Manual*, GTCO Corporation, Rockville, MD, 1982.

GUPT81a Gupta, S., and R.E. Sproull, "Filtering Edges for Gray-Scale Displays," *SIGGRAPH 81*, 1–5.

GUPT81b Gupta, S., R. Sproull, and I. Sutherland, "A VLSI Architecture for Updating Raster-Scan Displays," *SIGGRAPH 81*, 71–78.

GUPT86 Gupta, S., D.F. Bantz, P.N. Sholtz, C.J. Evangelisti, and W.R. DeOrazio, *YODA: An Advanced Display for Personal Computers*, Computer Science Research Report RC11618 (–52213), IBM Thomas J. Watson Research Center, Yorktown Heights, NY, October 1986.

GURW81 Gurwitz, R., R. Fleming and A. van Dam, "MIDAS: A Microprocessor Instructional Display and Animation System," *IEEE Transactions on Education*, February, 1981.

HAEU76 Haeusing, M., "Color Coding of Information on Electronic Displays," in *Proceedings of the Sixth Congress of the International Ergonomics Association*, 1976, 210–217.

HAGE86 Hagen, M., *Varieties of Realism*, Cambridge University Press, Cambridge, England, 1986.

HAGE88 Hagen, R.E., *An Algorithm for Incremental Anti-Aliased Lines and Curves*, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, January 1988.

HAIN86 Haines, E.A., and D.P. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," *CG & A*, 6(9), September 1986, 6–16.

HAIN89 Haines, E., "Essential Ray Tracing Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989, 33–77.

HALA68 Halas, J., and R. Manvell, *The Technique of Film Animation*, Hastings House, New York, 1968.

HALA73 Halas, J., ed., *Visual Scripting*, Hastings House, New York, 1973.

HALA82 Halasz, F., and T. Moran, "Analogy Considered Harmful," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 383–386.

HALL83 Hall, R.A., and D.P. Greenberg, "A Testbed for Realistic Image Synthesis," *CG & A*, 3(8), November 1983, 10–20.

HALL86 Hall, R., "Hybrid Techniques for Rapid Image Synthesis," in Whitted, T., and R. Cook, eds., *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, TX, August 1986.

HALL89 Hall, R., *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1989.

HAMI53 Hamilton, W.R., *Lectures on Quaternions: Containing a Systematic Statement of a New Mathematical Method; of Which the Principles Were Communicated in 1843 to the Royal Irish Academy; and Which Has Since Formed the Subject of Successive Courses of Lectures, Delivered in 1848 and Subsequent Years, in the Halls of Trinity College, Dublin: With Numerous Illustrative Examples*, Hodges and Smith, Dublin, 1853.

HAML77 Hamlin, G., Jr., and C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques," *SIGGRAPH 77*, 206–213. Also in FREE80, 264–271.

HANA80 Hanau, P., and D. Lenorovitz, "Prototyping and Simulation Tools for User/Computer Dialogue Design," *SIGGRAPH 80*, 271–278.

HANR83 Hanrahan, P., "Ray Tracing Algebraic Surfaces," *SIGGRAPH 83*, 83–90.

HANR89 Hanrahan, P., "A Survey of Ray-Surface Intersection Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989, 79–119.

HANS71 Hansen, W., "User Engineering Principles for Interactive Systems," in *FJCC 1971*, AFIPS Press, Montvale, NJ, 1971, 523–532.

HARR88 Harrington, S.J., and R.R. Buckley, *Interpress: The Source Book*, Brady, New York, 1988.

HART89 Hartson, R., and D. Hix, "Human–Computer Interface Development: Concepts and Systems," *ACM Computing Surveys*, 21(1), March 1989, 5–92.

HATF89 Hatfield, D., *Anti-Aliased, Transparent, and Diffuse Curves*, IBM Technical Computing Systems Graphics Report 0001, International Business Machines, Cambridge, MA, 1989.

HAYE83 Hayes, P., and Szekely, P., "Graceful Interaction Through the COUSIN Command Interface," *International Journal of Man–Machine Studies*, 19(3), September 1983, 285–305.

HAYE84 Hayes, P., "Executable Interface Definitions Using Form-Based Interface Abstractions," in *Advances in Computer-Human Interaction*, Hartson, H.R., ed., Ablex, Norwood, NJ, 1984.

HAYE85 Hayes, P., P. Szekely, and R. Lerner, "Design Alternatives for User Interface Management Systems Based on Experience with COUSIN," in *CHI '85 Proceedings*, ACM, New York, 1985, 169–175.

HECK82 Heckbert, P., "Color Image Quantization for Frame Buffer Display," *SIGGRAPH 82*, 297–307.

HECK84 Heckbert, P.S., and P. Hanrahan, "Beam Tracing Polygonal Objects," *SIGGRAPH 84*, 119–127.

HECK86a Heckbert, P.S., "Filtering by Repeated Integration," *SIGGRAPH 86*, 315–321.

HECK86b Heckbert, P.S., "Survey of Texture Mapping," *CG & A*, 6(11), November 1986, 56–67.

HEDG82 Hedgley, D.R., Jr., *A General Solution to the Hidden-Line Problem*, NASA Reference Publication 1085, NASA Scientific and Technical Information Branch, 1982.

HEME82 Hemenway, K., "Psychological Issues in the Use of Icons in Command Menus," in *Proceedings Human Factors in Computer Systems Conference*, ACM, New York, 1982 20–23.

HERO76 Herot, C., "Graphical Input Through Machine Recognition of Sketches," *SIGGRAPH 76*, 97–102.

HERO78 Herot, C., and G. Weinzapfel, "One-Point Touch Input of Vector Information for Computer Displays," *SIGGRAPH 78*, 210–216.

HERZ80 Herzog, B., "In Memoriam of Steven Anson Coons," *Computer Graphics*, 13(4), February 1980, 228–231.

HILL83 Hill, F.S., Jr., S. Walker, Jr., and F. Gao, "Interactive Image Query System Using Progressive Transmission," *SIGGRAPH 83*, 323–333.

HILL87 Hill, R., "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction—The Sassafras UIMS," *ACM TOG*, 5(3), July 1986, 179–210.

HIRS70 Hirsch, R., "Effects of Standard vs. Alphabetical Keyboard Formats on Typing Performance," *Journal of Applied Psychology*, 54, December 1970, 484–490

HOBB89 Hobby, J. D., "Rasterizing Curves of Constant Width," *JACM*, 36(2), April 1989, 209–229.

HODG85 Hodges, L., and D. McAllister, "Stereo and Alternating-Pair Techniques for Display of Computer-Generated Images," *CG & A*, 5(9), September 1985, 38–45.

HOFF61 Hoffman, K., and R. Kunze, *Linear Algebra*, Prentice–Hall, Englewood Cliffs, NJ, 1961.

HOLL80 Holladay, T. M., "An Optimum Algorithm for Halftone Generation for Displays and Hard Copies," *Proceedings of the Society for Information Display*, 21(2), 1980, 185–192.

HOPG86a Hopgood, F., D. Duce, J. Gallop, and D. Sutcliffe, *Introduction to the Graphical Kernel System (GKS)*, second edition, Academic Press, London, 1986.

HOPG86b Hopgood, F., D. Duce, E. Fielding, K. Robinson, and A. Williams, eds., *Methodology of Window Management*, Springer-Verlag, New York, 1986.

HORN79 Horn, B.K.P., and R.W. Sjoberg, "Calculating the Reflectance Map," *Applied Optics*, 18(11), June 1979, 1770–1779.

HUBS82 Hubschman, H., and S.W. Zucker, "Frame-to-Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World," *ACM TOG*, 1(2), April 1982, 129–162.

HUDS86 Hudson, S., and R. King, "A Generator of Direct Manipulation Office Systems," *ACM Transactions on Office Information Systems*, 4(2), April 1986, 132–163.

HUDS87 Hudson, S., "UIMS Support for Direct Manipulation Interfaces," *ACM SIGGRAPH Workshop on Software Tools for User Interface Management*, in *Computer Graphics*, 21(2), April 1987, 120–124.

HUDS88 Hudson, S., and R. King, "Semantic Feedback in the Higgens UIMS," *IEEE Transactions on Software Engineering*, 14(8), August 1988, 1188–1206.

HUGH89 Hughes, J., *Integer and Floating-Point Z-Buffer Resolution*, Department of Computer Science Technical Report, Brown University, Providence, RI, 1989.

HULL87 Hull, R., and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, 19(3), September 1987, 201–260.

HUNT78 Hunter, G.M., *Efficient Computation and Data Structures for Graphics*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

HUNT79 Hunter, G.M. and K. Steiglitz, "Operations on Images Using Quad Trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2), April 1979, 145–153.

HUNT81 Hunter, G.M., *Geometrees for Interactive Visualization of Geology: An Evaluation*, System Science Department, Schlumberger-Doll Research, Ridgefield, CT, 1981.

HUNT87 Hunt, R.W., *The Reproduction of Colour*, fourth edition, Fountain Press, Tolworth, England, 1987.

HURL89 Hurley, D., and J. Sibert, "Modeling User Interface-Application Interactions," *IEEE Software*, 6(1), January 1989, 71–77.

HUTC86 Hutchins, E., J. Hollan, and D. Norman, "Direct Manipulation Interfaces," in Norman, D., and S. Draper, eds., *User Centered System Design*, Erlbaum, Hillsdale, NJ, 1986, 87–124.

IES87 Illuminating Engineering Society, Nomenclature Committee, *ANSI/IES RP-16-1986: American National Standard: Nomenclature and Definitions for Illuminating Engineering*, Illuminating Engineering Society of North America, New York, 1987.

IMME86 Immel, D.S., M.F. Cohen, and D.P. Greenberg, "A Radiosity Method for Non-Diffuse Environments," *SIGGRAPH 86*, 133–142.

INFA85 Infante, C., "On the Resolution of Raster-Scanned CRT Displays," *Proceedings of the Society for Information Display*, 26(1), 1985, 23–36.

INGA81 Ingalls, D., "The SmallTalk Graphics Kernel," *BYTE*, 6(8), August 1981.

INTE85 Interaction Systems, Inc., *TK-1000 Touch System*, Interaction Systems, Inc., Newtonville, MA, 1985.

INTE88 International Standards Organization, *International Standard Information Processing Systems—Computer Graphics—Graphical Kernel System for Three Dimensions (GKS-3D) Functional Description*, ISO Document Number 8805:1988(E), American National Standards Institute, New York, 1988.

INTE89 Intel Corporation, *i860 Microprocessor Family Product Briefs*, Intel Corporation, Santa Clara, CA, 1989.

IRAN71 Irani, K., and V. Wallace, "On Network Linguistics and the Conversational Design of Queueing Networks," *JACM*, 18, October 1971, 616–629.

ISO International Standards Organization, *Information Processing Text and Office Systems Standard Page Description Language (SPDL)*, ISO Document Number JTC1 SC18/WG8N561, American National Standards Institute, New York.

JACK64 Jacks, E., "A Laboratory for the Study of Man–Machine Communication," in *FJCC 64*, AFIPS, Montvale, NJ, 1964, 343–350.

JACK80 Jackins, C., and S.L. Tanimoto, "Oct-Trees and Their Use in Representing Three-Dimensional Objects," *CGIP*, 14(3), November 1980, 249–270.

JACO83 Jacob, R., "Using Formal Specifications in the Design of the User-Computer Interface," *CACM*, 26(4), April 1983, 259–264.

JACO85 Jacob, R., "A State Transition Diagram Language for Visual Programming," *IEEE Computer*, 18(8), August 1985, 51–59.

JACO86 Jacob, R., "A Specification Language for Direct-Manipulation User Interfaces," *ACM TOG*, 5(4), October 1986, 283–317.

JANS85 Jansen, F.W., "A CSG List Priority Hidden Surface Algorithm," in C. Vandoni, ed., *Proceedings of Eurographics 85*, North-Holland, Amsterdam, 1985, 51–62.

JANS87 Jansen, F.W., *Solid Modelling with Faceted Primitives*, Ph.D. Thesis, Department of Industrial Design, Delft University of Technology, Netherlands, September 1987.

JARV76a Jarvis, J.F., C.N. Judice, and W.H. Ninke, "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays," *CGIP*, 5(1), March 1976, 13–40.

JARV76b	Jarvis, J.F., and C.S. Roberts, "A New Technique for Displaying Continuous Tone Images on a Bilevel Display," *IEEE Trans.*, COMM-24(8), August 1976, 891–898.
JENK89	Jenkins, R.A., "New Approaches in Parallel Computing," *Computers in Physics*, 3(1), January–February 1989, 24–32.
JEVA89	Jevans, D.A., "A Review of Multi-Computer Ray Tracing," *Ray Tracing News*, 3(1), May 1989, 8–15.
JOBL78	Joblove, G.H., and D. Greenberg, "Color Spaces for Computer Graphics," *SIGGRAPH 78*, 20–27.
JOHN78	Johnson, S., and M. Lesk, "Language Development Tools," *Bell System Technical Journal*, 57(6,7), July–August 1978, 2155–2176.
JOHN82	Johnson, S.A., "Clinical Varifocal Mirror Display System at the University of Utah," in *Proceedings of SPIE*, 367, August 1982, 145–148.
JONE26	Jones, L., and E. Lowry, "Retinal Sensibility to Saturation Differences," *Journal of the Optical Society of America*, 13(25), 1926.
JOVA86	Jovanović, B., *Visual Programming of Functional Transformations in a Dynamic Process Visualization System*, Report GWU-IIST-86-22, Department of Computer Science, George Washington University, Washington, DC, 1986.
JOY86	Joy, K.I., and M.N. Bhetanabhotla, "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence," *SIGGRAPH 86*, 279–285.
JOY88	Joy, K., C. Grant, N. Max, and L. Hatfield, *Tutorial: Computer Graphics: Image Synthesis*, IEEE Computer Society, Washington, DC, 1988.
JUDD75	Judd, D., and G. Wyszecki, *Color in Business, Science, and Industry*, Wiley, New York, 1975.
JUDI74	Judice, J.N., J.F. Jarvis, and W. Ninke, "Using Ordered Dither to Display Continuous Tone Pictures on an AC Plasma Panel," *Proceedings of the Society for Information Display*, Q4 1974, 161–169.
KAJI82	Kajiya, J.T., "Ray Tracing Parametric Patches," *SIGGRAPH 82*, 245–254.
KAJI83	Kajiya, J., "New Techniques for Ray Tracing Procedurally Defined Objects," *SIGGRAPH 83*, 91–102.
KAJI84	Kajiya, J., and B. Von Herzen, "Ray Tracing Volume Densities," *SIGGRAPH 84*, 165–173.
KAJI85	Kajiya, J.T., "Anisotropic Reflection Models," *SIGGRAPH 85*, 15–21.
KAJI86	Kajiya, J.T., "The Rendering Equation," *SIGGRAPH 86*, 143–150.
KAPL85	Kaplan, G., and E. Lerner, "Realism in Synthetic Speech," *IEEE Spectrum*, 22(4), April 1985, 32–37.
KAPP85	Kappel, M.R., "An Ellipse-Drawing Algorithm for Raster Displays," in Earnshaw, R., ed. *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Springer-Verlag, Berlin, 1985, 257–280.
KASI82	Kasik, D., "A User Interface Management System," *SIGGRAPH 82*, 99–106.
KAUF88a	Kaufmann, H. E., "User's Guide to the Compositor," Computer Graphics Group Documentation, Brown University, Providence, RI, May 1988.
KAUF88b	Kaufman, A., and R. Bakalash, "Memory and Processing Architecture for 3D Voxel-Based Imagery," *CG & A*, 8(6), November 1988, 10–23.
KAWA82	Kawaguchi, Y., "A Morphological Study of the Form of Nature," *SIGGRAPH 82*, 223–232.
KAWA88	Kawaguchi, Y., film, *ACM SIGGRAPH 88 Electronic Theater and Video Review*, 26, 1988.

KAY79a Kay, D.S., *Transparency, Refraction and Ray Tracing for Computer Synthesized Images*, M.S. Thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, January 1979.

KAY79b Kay, D.S., and D. Greenberg, "Transparency for Computer Synthesized Images," *SIGGRAPH 79*, 158–164.

KAY86 Kay, T.L., and J.T. Kajiya, "Ray Tracing Complex Scenes," *SIGGRAPH 86*, 269–278.

KEDE84 Kedem, G., and J. L. Ellis, "The Raycasting Machine," in *Proceedings of the 1984 International Conference on Computer Design*, October 1984, 533–538.

KELL76 Kelly, K., and D. Judd, *COLOR—Universal Language and Dictionary of Names*, National Bureau of Standards Spec. Publ. 440, 003-003-01705-1, U.S. Government Printing Office, Washington, DC, 1976.

KELL88 Kelley, A.D., M.C. Malin, and G.M. Nielson, "Terrain Simulation Using a Model of Stream Erosion," *SIGGRAPH 88*, 263–268.

KIER85 Kieras, D., and P. Polson, "An Approach to the Formal Analysis of User Complexity," *International Journal of Man-Machine Studies*, 22(4), April 1985, 365–394.

KLEM71 Klemmer, E., "Keyboard Entry," *Applied Ergonomics*, 2(1), 1971, 2–6.

KLIN71 Klinger, A., "Patterns And Search Statistics," in Rustagi, J., ed., *Optimizing Methods in Statistics*, Academic Press, New York, 1971, 303–337.

KNOW77 Knowlton, K., and L. Cherry, "ATOMS—A Three-D Opaque Molecule System for Color Pictures of Space-Filling or Ball-and-Stick Models," *Computers and Chemistry*, 1, 1977, 161–166.

KNOW80 Knowlton, K., "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Loss-less Encoding Schemes," *Proc. of IEEE*, 68(7), 1980, 885–896.

KNUT69 Knuth, D.E., *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

KNUT87 Knuth, D., "Digital Halftones by Dot Diffusion," *ACM TOG*, 6(4), October 1987, 245–273.

KOBA87 Kobayashi, H., T. Nakamura, and Y. Shigei, "Parallel Processing of an Object Synthesis Using Ray Tracing," *Visual Computer*, 3(1), February 1987, 13–22.

KOCA87 Koçak, H., Bisshopp, F., Laidlaw, D., and T. Banchoff, "Topology and Mechanics with Computer Graphics: Linear Hamiltonian Systems in Four Dimensions," *Advances in Applied Mathematics*, 1986, 282–308.

KOCH84 Kochanek, D., and R. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *SIGGRAPH 84*, 33–41.

KOIV88 Koivunen, M., and M. Mäntylä, "HutWindows: An Improved Architecture for a User Interface Management System," *CG & A*, 8(1), January 1988, 43–52.

KORE82 Korein, J.U., and N.I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures," *CG & A*, 2(11), November 1982, 71–81.

KORE83 Korein, J., and N. Badler, "Temporal Anti-Aliasing in Computer Generated Animation," *SIGGRAPH 83*, 377–388.

KREB79 Krebs, M., and J. Wolf, "Design Principles for the Use of Color in Displays," *Proceedings of the Society for Information Display*, 20, 1979, 10–15.

KRUE83 Krueger, M., *Artificial Reality*, Addison-Wesley, Reading, MA, 1983.

KURL88 Kurlander, D., and S. Feiner, "Editable Graphical Histories," in *Proc. 1988 IEEE Workshop on Visual Languages*, October 10–12, 1988, Pittsburgh, PA, 129–132.

KURL90 Kurlander, D., and S. Feiner, "A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands," in Chang, S., ed., *Visual Languages and Visual Programming*, Plenum Press, New York, 1990, 257–275.

LAID86 Laidlaw, D.H., W.B. Trumbore, and J.F. Hughes, "Constructive Solid Geometry for Polyhedral Objects," *SIGGRAPH 86*, 161–170.

LAND85 Landauer, T., and D. Nachbar, "Selection from Alphabetic and Numeric Menu Trees Using a Touch-Sensitive Screen: Breadth, Depth, and Width," in *Proceedings CHI '85 Human Factors in Computing Systems Conference*, ACM, New York, 1985, 73–78.

LANE79 Lane, J., and L. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *CGIP*, 11(3), November 1979, 290–297.

LANE80a Lane, J., and R. Riesenfeld, "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1), January 1980, 35–46.

LANE80b Lane, J., L. Carpenter, T. Whitted, and J. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *CACM*, 23(1), January 1980, 23–34. Also in BEAT82, 468–479.

LANT84 Lantz, K., and W. Nowicki, "Structured Graphics for Distributed Systems," *ACM TOG*, 3(1), January 1984, 23–51.

LANT87 Lantz, K., P. Tanner, C. Binding, K. Huang, and A. Dwelly, "Reference Models, Window Systems, and Concurrency," in Olsen, D., ed., "ACM SIGGRAPH Workshop on Software Tools for User Interface Management," *Computer Graphics*, 21(2), April 1987, 87–97.

LASS87 Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation," *SIGGRAPH 87*, 35–44.

LAYB79 Laybourne, K., *The Animation Book*, Crown, New York, 1979.

LEAF74 Leaf, C., *The Owl Who Married a Goose*, film, National Film Board of Canada, 1974.

LEAF77 Leaf, C., *The Metamorphosis of Mr. Samsa*, film, National Film Board of Canada, 1977.

LEE85a Lee, S., W. Buxton, and K. Smith, "A Multi-touch Three Dimensional Touch-sensitive Tablet," in *Proceedings of CHI '85 Human Factors in Computing Systems Conference*, ACM, New York, 1985, 21–25.

LEE85b Lee, M.E., R.A. Redner, and S.P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *SIGGRAPH 85*, 61–67.

LEVI76 Levin, J., "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces," *CACM*, 19(10), October 1976, 555–563.

LEVI84 Levinthal, A., and T. Porter, "Chap—a SIMD Graphics Processor," *SIGGRAPH 84*, 77–82.

LEVO78 Levoy, M., *Computer Assisted Cartoon Animation*, Master's Thesis, Department of Architecture, Cornell University, Ithaca, NY, August 1978.

LEVO82 Levoy, M., "Area Flooding Algorithms," in *Two-Dimensional Computer Animation, Course Notes 9 for SIGGRAPH 82*, Boston, MA, July 26–30, 1982.

LEVO89 Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," in *Proceedings of the Volume Visualization Workshop*, Department of Computer Science, University of North Carolina at Chapel Hill, May 18-19, 1989, 85–90.

LIAN83 Liang, Y-D., and B.A. Barsky, "An Analysis and Algorithm for Polygon Clipping," *CACM*, 26(11), November 1983, 868–877, and Corrigendum, *CACM*, 27(2), February 1984, 151.

LIAN84 Liang, Y-D., and Barsky, B., "A New Concept and Method for Line Clipping," *ACM TOG*, 3(1), January 1984, 1–22.

LIEB78 Lieberman, H., "How to Color in a Coloring Book," *SIGGRAPH 78*, 111–116.

LIEN87 Lien, S.L., M. Shantz, and V. Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces," *SIGGRAPH 87*, 111–118.

LIND68 Lindenmayer, A, "Mathematical Models for Cellular Interactions in Development, Parts I and II," *J. Theor. Biol.*, 18, 1968, 280–315.

LINT89 Linton, M., J. Vlissides, and P. Calder, "Composing User Interfaces with InterViews," *IEEE Computer*, 22(2), February 1989, 8–22.

LIPS79 Lipscomb, J.S., *Three-Dimensional Cues for a Molecular Computer Graphics System*, Ph.D. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1979.

LOUT70 Loutrel, P.P., "A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra," *IEEE Trans. on Computers*, EC-19(3), March 1970, 205–213. Also in FREE80, 221–229.

LUCA84 Lucasfilm, Ltd., *The Adventures of André and Wally B.*, film, August 1984.

MACH78 Machover, C., "A Brief Personal History of Computer Graphics," *Computer*, 11(11), November 1978, 38–45.

MACK86 Mackinlay, J., "Automating the Design of Graphical Presentation of Relational Information," *ACM TOG*, 5(2), April 1986, 110–141.

MAGI68 Mathematical Applications Group, Inc., "3-D Simulated Graphics Offered by Service Bureau," *Datamation*, 13(1), February 1968, 69.

MAGN85 Magnenat-Thalmann, N., and Thalmann, D., *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo, 1985.

MAHL72 Mahl, R., "Visible Surface Algorithms for Quadric Patches," *IEEE Trans. on Computers*, C-21(1), January 1972, 1–4.

MAHN73 Mahnkopf, P., and J.L. Encarnação, *FLAVIS—A Hidden Line Algorithm for Displaying Spatial Constructs Given by Point Sets*, Technischer Bericht Nr. 148, Heinrich Hertz Institut, Berlin, 1973.

MAMM89 Mammen, A., "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," *CG & A*, 9(4), July 1989, 43–55.

MAND77 Mandelbrot, B., *Fractals: Form, Chance and Dimension*, W.H. Freeman, San Francisco, CA, 1977.

MAND82 Mandelbrot, B., Technical Correspondence, *CACM*, 25(8), August 1982, 581–583.

MÄNT88 Mäntylä, M. *Introduction to Solid Modeling*, Computer Science Press, Rockville, MD, 1988.

MARC80 Marcus, A., "Computer-Assisted Chart Making from the Graphic Designer's Perspective," *SIGGRAPH 80*, 247–253.

MARC82 Marcus, A., "Color: A Tool for Computer Graphics Communication," in Greenberg, D., A. Marcus, A. Schmidt, and V. Gorter, *The Computer Image*, Addison-Wesley, Reading, MA, 1982, 76–90.

MARC84 Marcus, A., "Corporate Identity for Iconic Interface Design: The Graphic Design Perspective," *CG & A*, 4(12), December 1984, 24–32.

MARK80 Markowsky, G., and M.A. Wesley, "Fleshing Out Wire Frames," *IBM Journal of Research and Development*, 24(5), September 1980, 582–597.

MARS85 Marsden, J., and A. Weinstein, *Calculus I, II, and III*, second edition, Springer Verlag, New York, 1985.

MART89 Martin, G., "The Utility of Speech Input in User–Computer Interfaces," *International Journal of Man–Machine Studies*, 30(4), April 1989, 355–376.

MASS85 Massachusetts Computer Corporation (MASSCOMP), *Graphics Application Programming Manual*, Order No. M-SP40-AP, MASSCOMP, Westford, MA, 1985.

MAUL89 Maulsby, D., and I. Witten, "Inducing Programs in a Direct Manipulation Environment," in *Proceedings CHI 1989*, ACM, New York, 1989, 57–62.

MAX79 Max, N.L., "ATOMLLL: - ATOMS with Shading and Highlights," *SIGGRAPH 79*, 165–173.

MAX81 Max, N., *Carla's Island*, animation, *ACM SIGGRAPH 81 Video Review*, 5, 1981.

MAX82 Max, N., "SIGGRAPH '84 Call for Omnimax Films," Computer Graphics, 16(4), December 1982, 208–214.

MAX84 Max, N.L., "Atoms with Transparency and Shadows," *CVGIP*, 27(1), July 1984, 46–63.

MAX86 Max, N., "Atmospheric Illumination and Shadows," *SIGGRAPH 86*, 117–124.

MAXW46 Maxwell, E. A., *Methods of Plane Projective Geometry Based on the Use of General Homogeneous Coordinates*, Cambridge University Press, Cambridge, England, 1946.

MAXW51 Maxwell, E. A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, Cambridge, England, 1951.

MAYH90 Mayhew, D., *Principles and Guidelines in User Interface Design*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

MCIL83 McIlroy, M.D., "Best Approximate Circles on Integer Grids," *ACM TOG*, 2(4), October 1983, 237–263.

MCLE88 McLeod, J., "HP Delivers Photo Realism on an Interactive System," *Electronics*, 61(6), March 17, 1988, 95–97.

MCMI87 McMillan, L., "Graphics at 820 MFLOPS," *ESD: The Electronic Systems Design Magazine*, 17(9), September 1987, 87–95.

MEAG80 Meagher, D., *Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer*, Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, October 1980.

MEAG82a Meagher, D., "Geometric Modeling Using Octree Encoding," *CGIP*, 19(2), June 1982, 129–147.

MEAG82b Meagher, D., "Efficient Synthetic Image Generation of Arbitrary 3-D Objects," in *Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing*, IEEE Computer Socitey Press, Washington, DC, 1982.

MEAG84 Meagher, D., "The Solids Engine: A Processor for Interactive Solid Modeling," in *Proceedings of NICOGRAPH '84*, Tokyo, November 1984.

MEAG85 Meagher, D., "Applying Solids Processing to Medical Planning," in *Proceedings of NCGA '85*, Dallas, 1985, 101–109.

MEGA89 Megatek Corporation, *Sigma 70 Advanced Graphics Workstations*, Megatek Corporation, San Diego, CA, 1989.

MEIE83 Meier, B., *Brim*, Computer Graphics Group Documentation, Computer Science Department, Brown University, Providence, RI, 1983.

MEIE88 Meier, B., "ACE: A Color Expert System for User Interface Design," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 117–128, 1988.

MEYE80 Meyer, G.W., and D.P. Greenberg, "Perceptual Color Spaces for Computer Graphics," *SIGGRAPH 80*, 254–261.

MEYE83 Meyer, G., *Colorimetry and Computer Graphics*, Program of Computer Graphics, Cornell University, Ithaca, NY, 1983.

MEYE88 Meyer, G., and Greenberg, D., "Color-defective Vision and Computer Graphic Displays," *CG & A*, 8(5), September 1988, 28–40.

MICH71 Michaels, S., "QWERTY Versus Alphabetical Keyboards as a Function of Typing Skill," *Human Factors*, 13(5), October 1971, 419–426.

MICR89 Microsoft Corporation, *Presentation Manager*, Microsoft Corporation, Bellevue, WA, 1989.

MILL87 Miller, J.R., "Geometric Approaches to Nonplanar Quadric Surface Intersection Curves," *ACM TOG*, 6(4), October 1987, 274–307.

MILL88a Miller, G.S.P., "The Motion Dynamics of Snakes and Worms," *SIGGRAPH 88*, 169–178.

MILL88b Miller, G.S.P., "The Motion Dynamics of Snakes and Worms," lecture at ACM SIGGRAPH '88.

MILL88c Miller, P., and M. Szczur, "Transportable Application Environment (TAE) Plus Experiences in 'Object'ively Modernizing a User Interface Environment," in *Proceedings of OOPSLA '88*, 58–70.

MILL89 Miller, J.R., "Architectural Issues in Solid Modelers," *CG & A*, 9(5), September 1989, 72–87.

MINS84 Minsky, M., "Manipulating Simulated Objects with Real-World Gestures Using a Force and Position Sensitive Screen," *SIGGRAPH 84*, 195–203.

MITC87 Mitchell, D.P., "Generating Antialiased Images at Low Sampling Densities," *SIGGRAPH 87*, 65–72.

MITC88 Mitchell, D.P., and A.N. Netravali, "Reconstruction Filters in Computer Graphics," *SIGGRAPH 88*, 221–228.

MOLN88 Molnar, S., "Combining Z-Buffer Engines for Higher-Speed Rendering," 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988. To appear in Kuijk, A.A.M., ed., *Advances in Computer Graphics Hardware III*, Proceedings of 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Springer-Verlag, Berlin, 1989.

MORR86 Morris, J., M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment," *CACM*, 29(3), March 1986, 184–201.

MORT85 Mortenson, M., *Geometric Modeling*, Wiley, New York, 1985.

MUNS76 Munsell Color Company, *Book of Color*, Munsell Color Company, Baltimore, MD, 1976.

MURC85 Murch, G., "Using Color Effectively: Designing to Human Specifications," *Technical Communications*, Q4 1985, Tektronix Corporation, Beaverton, OR, 14–20.

MUSG89 Musgrave, F.K., "Prisms and Rainbows: A Dispersion Model for Computer Graphics," in *Proceedings of Graphics Interface '89*, London, Ontario, June 19–23, 1989, 227–234.

MYER68 Myer, T., and I. Sutherland, "On the Design of Display Processors," *CACM*, 11(6), June 1968, 410–414.

MYER75 Myers, A.J., *An Efficient Visible Surface Program*, Report to the National Science Foundation, Computer Graphics Research Group, Ohio State University, Columbus, OH, July 1975.

MYER84 Myers, B., "The User Interface for Sapphire," *CG & A*, 4(12), December 1984, 13–23.

MYER85 Myers, B., "The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces," in *Proceedings CHI '85*, ACM, New York, 1985, 11–17.

MYER86 Myers, B., "Creating Highly-Interactive and Graphical User Interfaces by Demonstration," *SIGGRAPH 86*, 249–257.

MYER88 Myers, B., *Creating User Interfaces by Demonstration*, Academic Press, New York, 1988.

MYER89 Myers, B., "User-Interface Tools: Introduction and Survey," *IEEE Software*, 6(1), January 1989, 15–23.

NAIM87	Naiman, A., and A. Fournier, "Rectangular Convolution for Fast Filtering of Characters," *SIGGRAPH 87*, 233–242.
NARU87	Naruse, T., M. Yoshida, T. Takahashi, and S. Naito, "SIGHT—a Dedicated Computer Graphics Machine," *Computer Graphics Forum*, 6(4), December 1987, 327–334.
NAVA89	Navazo, I., "Extended Octree Representation of General Solids with Plane Faces: Model Structure and Algorithms," *Computers and Graphics*, 13(1), January 1989, 5–16.
NAYL90	Naylor, B.F., "Binary Space Partitioning Trees as an Alternative Representation of Polytopes," *CAD*, 22(4), May 1990, 250–253.
NEMO86	Nemoto, K., and T. Omachi, "An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing," in *Proceedings of Graphics Interface '86*, 1986, 43–48.
NEWE72	Newell, M.E., R.G. Newell, and T.L. Sancha, "A Solution to the Hidden Surface Problem," in *Proceedings of the ACM National Conference 1972*, 443–450. Also in FREE80, 236–243.
NEWE74	Newell, M.E., *The Utilization of Procedure Models in Digital Image Synthesis*, Ph.D. Thesis, Technical Report UTEC-CSc-76-218, NTIS AD/A039 008/LL, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.
NEWM68	Newman, W., "A System for Interactive Graphical Programming," *SJCC*, Thompson Books, Washington, DC, 1968, 47–54.
NEWM71	Newman, W. M., "Display Procedures," *CACM*, 14(10), 1971, 651–660.
NEWM73	Newman, W., and R. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1973.
NEWM79	Newman, W., and R. Sproull, *Principles of Interactive Computer Graphics*, 2nd ed., McGraw-Hill, New York, 1979.
NICH87	Nicholl, T.M., D.T. Lee, and R.A. Nicholl, "An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis," *SIGGRAPH 87*, 253–262.
NICO77	Nicodemus, F.E., J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis, *Geometrical Considerations and Nomenclature for Reflectance*, NBS Monograph 160, U.S. Department of Commerce, Washington DC, October 1977.
NIEL86	Nielson, G., and D. Olsen, Jr., "Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices," in *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, ACM, New York, 1987, 175–182.
NISH83a	Nishimura, H., H. Ohno, T. Kawata, I. Shirakawa, and K. Omura, "LINKS-1: A Parallel Pipelined Multimicrocomputer System for Image Creation," in *Proceedings of the Tenth International Symposium on Computer Architecture, ACM SIGARCH Newsletter*, 11(3), 1983, 387–394.
NISH83b	Nishita, T. and E. Nakamae, "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources," *Proc. IEEE Computer Society International Computer Software and Applications Conference (COMPSAC)*, IEEE Computer Society, Washington, DC, November 1983, 237–241.
NISH85a	Nishita, T., and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *SIGGRAPH 85*, 23–30.
NISH85b	Nishita, T., I. Okamura, and E. Nakamae, "Shading Models for Point and Linear Sources," *ACM TOG*, 4(2), April 1985, 124–146.
NISH86	Nishita, T., and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Sky Light," *SIGGRAPH 86*, 125–132.
NISH87	Nishita, T., Y. Miyawaki, and E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources," *SIGGRAPH 87*, 303–310.

NOLL67 Noll, M., "A Computer Technique for Displaying N-dimensional Hyperobjects," *CACM*, 10(8), August 1967, 469–473.

NORM88 Norman, D., *The Psychology of Everyday Things*, Basic Books, New York, 1988.

OKIN84 Okino, N., Y. Kakazu, and M. Morimoto, "Extended Depth-Buffer Algorithms for Hidden Surface Visualization," *CG & A*, 4(5), May 1984, 79–88.

OLIV85 Oliver, M., "Display Algorithms for Quadtrees and Octtrees and their Hardware Realisation," in Kessener, L., F. Peters, and M. van Lierop, eds., *Data Structures for Raster Graphics*, Springer-Verlag, Berlin, 1986, 9–37.

OLSE83 Olsen, D., and E. Dempsey, "SYNGRAPH: A Graphical User Interface Generator," *SIGGRAPH 83*, 43–50.

OLSE84a Olsen, D., "Pushdown Automata for User Interface Management," *ACM TOG*, 3(3), July 1984, 177–203.

OLSE84b Olsen, D., W. Buxton, R. Ehrich, D. Kasik, J. Rhyne, and J. Sibert, "A Context for User Interface Management," *CG & A*, 4(12), December 1984, 33–42.

OLSE86 Olsen, D., "MIKE: The Menu Interaction Kontrol Environment," *ACM TOG*, 5(4), October 1986, 318–344.

OLSE87 Olsen, D., ed., ACM SIGGRAPH Workshop on Software Tools for User Interface Management, *Computer Graphics*, 21(2), April 1987, 71–147.

OLSE88 Olsen, D., "Macros by Example in a Graphical UIMS," *CG & A*, 8(1), January 1988, 68–78.

OLSE89 Olsen, D., "A Programming Language Basis for User Interface Management," in *Proceedings CHI '89*, ACM, New York, 1989, 171–176.

OPEN89a Open Software Foundation, *OSF/MOTIF™ Manual*, Open Software Foundation, Cambridge, MA, 1989.

OPEN89b Open Software Foundation, *OSF/MOTIF™ Style Guide*, Open Software Foundation, Cambridge, MA, 1989.

OSTW31 Ostwald, W., *Colour Science*, Winsor & Winsor, London, 1931.

PAET86 Paeth, A.W., "A Fast Algorithm for General Raster Rotation," in *Proceedings Graphics Interface '86*, Canadian Information Processing Society, 1986, 77–81.

PAET89 Paeth, A. W., "Fast Algorithms for Color Correction," *Proceedings of the Society for Information Display*, 30(3), Q3 1989, 169–175, reprinted as Technical Report CS-89-42, Department of Computer Science, University of Waterloo, Waterloo, Canada, 1989.

PAIN89 Painter, J., and K. Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *SIGGRAPH 89*, 281–288.

PALA88 Palay, A., W. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, and T. Peters, "The Andrew Toolkit: An Overview," in *Proceedings 1988 Winter USENIX*, February 1988, 9–21.

PARK80 Parke, F., "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems," *SIGGRAPH 80*, 48–56.

PARK82 Parke, F.I., "Parameterized Models for Facial Animation," *CG & A*, 2(11), November 1982, 61–68.

PARK88 Parker, R., *Looking Good in Print: A Guide to Basic Design for Desktop Publishing*, Ventana Press, Chapel Hill, NC, 1988.

PAVL81 Pavlidis, T. "Contour Filling in Raster Graphics," *CGIP*, 10(2), June 1979, 126–141.

PEAC85 Peachey, D.R., "Solid Texturing of Complex Surfaces," *SIGGRAPH 85*, 279–286.

PEAR86 Pearson, G., and M. Weiser, "Of Moles and Men: The Design of Foot Controls for Workstations," in *Proceedings CHI '86*, ACM, New York, 1986, 333–339.

PEAR88	Pearson, G., and M. Weiser, "Exploratory Evaluation of a Planar Foot-operated Cursor Positioning Device," in *Proceedings CHI '88*, ACM, New York, 1988, 13–18.
PEIT86	Peitgen, H.-O., and P.H. Richter, *The Beauty of Fractals: Images of Complex Dynamical Systems*, Springer-Verlag, Berlin, 1986.
PEIT88	Peitgen, H.-O., and D. Saupe, eds., *The Science of Fractal Images*, Springer-Verlag, New York, 1988.
PERL85	Perlin, K., "An Image Synthesizer," *SIGGRAPH 85*, 287–296.
PERL89	Perlin, K., and E. Hoffert, "Hypertexture," *SIGGRAPH 89*, 253–262.
PERR85	Perry, T., and P. Wallach, "Computer Displays: New Choices, New Tradeoffs," *IEEE Spectrum*, 22(7), July 1985, 52–59.
PERR89	Perry, T., and J. Voelcker, "Of Mice and Menus: Designing the User-Friendly Interface," *IEEE Spectrum*, 26(9), September 1989, 46–51.
PERS85	Personics Corporation, *View Control System*, Concord, MA, 1985.
PETE86	Peterson, J.W., R.G. Bogart, and S.W. Thomas, *The Utah Raster Toolkit*, University of Utah, Department of Computer Science, Salt Lake City, UT, 1986.
PHIG88	PHIGS+ Committee, Andries van Dam, chair, "PHIGS+ Functional Description, Revision 3.0," *Computer Graphics*, 22(3), July 1988, 125–218.
PIKE83	Pike, R., "Graphics in Overlapping Bitmap Layers," *ACM TOG*, 17(3), July 83, 331–356.
PIKE84	Pike, R., "Bitmap Graphics," in *Course Notes 4 for SIGGRAPH 84*, Minneapolis, MN, July 23–27,1984.
PINK83	Pinkham, R., M. Novak, and K. Guttag, "Video RAM Excels at Fast Graphics," *Electronic Design*, 31(17), Aug. 18, 1983, 161–182.
PITT67	Pitteway, M.L.V., "Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter," *Computer J.*, 10(3), November 1967, 282–289.
PITT80	Pitteway, M.L.V., and D. J. Watkinson, "Bresenham's Algorithm with Grey-Scale," *CACM*, 23(11), November 1980, 625–626.
PITT87	Pitteway, M.L.V., "Soft Edging Fonts," Computer Graphics Technology and Systems, in *Proceedings of the Conference Held at Computer Graphics '87*, London, October 1987, Advanced computing series, 9, Online Publications, London, 1987.
PIXA86	Pixar Corporation, *Luxo, Jr.*, film, Pixar Corporation, San Rafael, CA, 1986.
PIXA88	Pixar Corporation, *The RenderMan Interface*, Version 3.0, Pixar Corporation, San Rafael, CA, May 1988.
PLAT81	Platt, S.M., and N.I. Badler, "Animating Facial Expressions," *SIGGRAPH 81*, 245–252.
PLAT88	Platt, J.C., and A.H. Barr, "Constraint Methods for Flexible Models," *SIGGRAPH 88*, 279–288.
PORT79	Porter, T., "The Shaded Surface Display of Large Molecules," *SIGGRAPH 79*, 234–236.
PORT84	Porter, T., and T. Duff, "Compositing Digital Images," *SIGGRAPH 84*, 253–259.
POSC89	Posch, K.C., and W.D. Fellner, "The Circle-Brush Algorithm," *ACM TOG*, 8(1), January 1989, 1–24.
POTM82	Potmesil, M., and I. Chakravarty, "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM TOG*, 1(2), April 1982, 85–108.
POTM83	Potmesil, M., and I. Chakravarty, "Modeling Motion Blur in Computer-Generated Images," *SIGGRAPH 83*, 389–399.
POTM89	Potmesil, M., and E. Hoffert, "Pixel Machine: A Parallel Image Computer," *SIGGRAPH 89*, 69–78.

POTT88 Potter, R., L. Weldon, and B. Shneiderman, "Improving the Accuracy of Touch Screens: An Experimental Evaluation of Three Strategies," in *Proceedings CHI '88*, ACM, New York, 27–32.

PRAT84 Pratt, M., "Solid Modeling and the Interface Between Design and Manufacture," *CG & A*, 4(7), July 1984, 52–59.

PRAT85 Pratt, V., "Techniques for Conic Splines," *SIGGRAPH 85*, 151–159.

PREP85 Preparata, F. P., and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

PRES88 Press, W.H., B.P. Flannery, S.A. Teukolskym, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1988.

PRIN71 Prince, D., *Interactive Graphics for Computer Aided Design*, Addison-Wesley, Reading, MA, 1971.

PRIT77 Pritchard, D.H., "U.S. Color Television Fundamentals—A Review," *IEEE Transactions on Consumer Electronics*, CE-23(4), November 1977, 467–478.

PRUS88 Prusinkiewicz, P., A. Lindenmayer, and J. Hanan, "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," *SIGGRAPH 88*, 141–150.

PUTN86 Putnam, L.K., and P.A. Subrahmanyam, "Boolean Operations on n-Dimensional Objects," *CG & A*, 6(6), June 1986, 43–51.

QUIN82 Quinlan, K.M., and J.R. Woodwark, "A Spatially-Segmented Solids Database—Justification and Design," in *Proc. CAD '82 Conf.*, Fifth International Conference and Exhibit on Computers in Design Engineering, Mar. 30–Apr 1, 1982, Butterworth, Guildford, Great Britain, 1982, 126–132.

RATL72 Ratliff, F., "Contour and Contrast," *Scientific American*, 226(6), June 1972, 91–101. Also in BEAT82, 364–375.

REDD78 Reddy, D., and S. Rubin, *Representation of Three-Dimensional Objects*, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

REEV81 Reeves, W.T., "Inbetweening for Computer Animation Utilizing Moving Point Constraints," *SIGGRAPH 81*, 263–269.

REEV83 Reeves, W.T., "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *SIGGRAPH 83*, 359–376.

REEV85 Reeves, W.T., and R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems," *SIGGRAPH 85*, 313–322.

REEV87 Reeves, W.T., D.H. Salesin, and R.L. Cook, "Rendering Antialiased Shadows with Depth Maps," *SIGGRAPH 87*, 283–291.

REFF88 de Reffye, P., C. Edelin, J. Françon, M. Jaeger, and C. Puech, "Plant Models Faithful to Botanical Structure and Development," *SIGGRAPH 88*, 151–158.

REIS82 Reisner, P., "Further Developments Toward Using Formal Grammar as a Design Tool," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 304–308.

REQU77 Requicha, A.A.G., *Mathematical Models of Rigid Solids*, Tech. Memo 28, Production Automation Project, University of Rochester, Rochester, NY, 1977.

REQU80 Requicha, A.A.G., "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, 12(4), December 1980, 437–464.

REQU82 Requicha, A.A.G., and H.B. Voelcker, "Solid Modeling: A Historical Summary and Contemporary Assessment," *CG & A*, 2(2), March 1982, 9–24.

REQU83 Requicha, A.A.G., and H.B. Voelcker, "Solid Modeling: Current Status and Research Directions," *CG & A*, 3(7), October 1983, 25–37.

REQU84 Requicha, A.A.G., "Representation of Tolerances in Solid Modeling: Issues and Alternative Approaches," in Pickett, M., and J. Boyse, eds., *Solid Modeling by Computers*, Plenum Press, New York, 1984, 3–22.

REQU85 Requicha, A.A.G., and H.B. Voelcker, "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE*, 73(1), January 1985, 30–44.

REYN82 Reynolds, C.W., "Computer Animation with Scripts and Actors," *SIGGRAPH 82*, 289–296.

REYN87 Reynolds, C.W., "Flocks, Herds and Schools: A Distributed Behavioral Model," *SIGGRAPH 87*, 25–34.

RHOD89 Rhoden, D., and C. Wilcox, "Hardware Acceleration for Window Systems," *SIGGRAPH 89*, 61–67.

RHYN87 Rhyne, J., "Dialogue Management for Gestural Interfaces," *Proceedings ACM SIGGRAPH Workshop on Tools for User Interface Management*, in *Computer Graphics*, 21(2), April 1987, 137–145.

ROBE63 Roberts, L.G., *Machine Perception of Three Dimensional Solids*, Lincoln Laboratory, TR 315, MIT, Cambridge, MA, May 1963. Also in Tippet, J.T., *et al.*, eds., *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, MA, 1964, 159–197.

ROBE65 Roberts, L.G., *Homogeneous Matrix Representations and Manipulation of* N-*Dimensional Constructs*, Document MS 1405, Lincoln Laboratory, MIT, Cambridge, MA, 1965.

ROGE85 Rogers, D.F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.

ROGO83 Rogowitz, B., "The Human Visual System: A Guide for the Display Technologist," *Proceedings Society for Information Display*, 24(3), 1983.

ROMN69 Romney, G.W., G.S. Watkins, and D.C. Evans, "Real Time Display of Computer Generated Half-Tone Perspective Pictures," in *Proceedings 1968 IFIP Congress*, North Holland Publishing Co., 1969, 973–978.

ROSE83 Rosenthal, D., "Managing Graphical Resources," *Computer Graphics*, 17(1), January 1983, 38–45.

ROSE85 Rose, C., B. Hacker, R. Anders, K. Wittney, M. Metzler, S. Chernicoff, C. Espinosa, A. Averill, B. Davis, and B. Howard, *Inside Macintosh*, I, Addison-Wesley, Reading, MA, 1985, I-35–I-213.

ROSS86 Rossignac, J.R., and A.A.G. Requicha, "Depth-Buffering Display Techniques for Constructive Solid Geometry," *CG & A*, 6(9), September 1986, 29–39.

ROSS89 Rossignac, J., and H. Voelcker, "Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms," *ACM TOG*, 8(1), January 1989, 51–87.

ROTH76 Rothstein, J., and C.F.R. Weiman, "Parallel and Sequential Specification of a Context Sensitive Language for Straight Lines on Grids," *CGIP*, 5(1), March 1976, 106–124.

ROTH82 Roth, S., "Ray Casting for Modeling Solids," *CGIP*, 18(2), February 1982, 109–144.

RUBE83 Rubel, A., "Graphic Based Applications—Tools to Fill the Software Gap," *Digital Design*, 3(7), July 1983, 17–30.

RUBE84 Rubenstein, R., and H. Hersh, *The Human Factor—Designing Computer Systems for People*, Digital Press, Burlington, MA, 1984.

RUBE88 Rubenstein, R., *Digital Typography*, Addison-Wesley, Reading, MA, 1988.

RUBI80 Rubin, S.M., and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *SIGGRAPH 80*, 110–116.

RUSH86 Rushmeier, H.E., *Extending the Radiosity Method to Transmitting and Specularly Reflecting Surfaces*, M.S. Thesis, Mechanical Engineering Department, Cornell University, Ithaca, NY, 1986.

RUSH87 Rushmeier, H., and K. Torrance, "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium," *SIGGRAPH 87*, 293–302.

SABE88 Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields," *SIGGRAPH 88*, 51–58.

SALE85 Salesin, D., and R. Barzel, *Two-Bit Graphics*, Computer Graphics Project, Computer Division, Lucasfilm, Ltd., San Rafael, CA, 1985; also in CG & A, 6(6), June 1986, 36–42.

SALM96 Salmon, G., *A Treatise on Conic Sections*, Longmans, Green, & Co., 10th edition, London 1896.

SALV87 Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987.

SAME84 Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Comp. Surv.*, 16(2), June 1984, 187–260.

SAME88a Samet, H., and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals," *CG & A*, 8(3), May 1988, 48–68.

SAME88b Samet, H. and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications," *CG & A*, 8(4), July 1988, 59–75.

SAME89a Samet, H., "Neighbor Finding in Images Represented by Octrees," *CGVIP*, 46(3), June 1989, 367–386.

SAME89b Samet, H., "Implementing Ray Tracing with Octrees and Neighbor Finding," *Computers and Graphics*, 13(4), 1989, 445–460.

SAME90a Samet, H., *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

SAME90b Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1990.

SARR83 Sarraga, R.F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID," *CVGIP*, 22(2), May 1983, 222–238.

SCHA83 Schachter, B., *Computer Image Generation*, Wiley, New York, 1983.

SCHE86 Scheifler, R., and J. Gettys, "The X Window System," *ACM TOG*, 5(2), April 1986, 79–109.

SCHE88a Scheifler, R.W., J. Gettys, and R. Newman, *X Window System*, Digital Press, 1988.

SCHE88b Scherson, I.D., and E. Caspary, "Multiprocessing for Ray-Tracing: A Hierarchical Self-balancing Approach," *Visual Computer*, 4(4), October 1988, 188–196.

SCHM83 Schmid, C., *Statistical Graphics: Design Principles and Practice*, Wiley, New York, 1983.

SCHM86 Schmucker, K., "MacApp: An Application Framework," *Byte*, 11(8), August 1986, 189–193.

SCHU69 Schumacker, R., B. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-Generated Images to Visual Simulation*, Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX, September 1969.

SCHU80 Schumacker, R., "A New Visual System Architecture," in *Proceedings of the Second Interservice/Industry Training Equipment Conference*, Salt Lake City, UT, 16-20 November 1980.

SCHU85 Schulert, A., G. Rogers, and J. Hamilton, "ADM—A Dialog Manager," in *CHI '85 Proceedings*, San Francisco, CA, Apr 14–18, 1985, 177–183.

SCHW82 Schweitzer, D., and E. Cobb, ''Scanline Rendering of Parametric Surfaces,'' *SIGGRAPH 82*, 265–271.

SCHW87 Schwarz, M., W. Cowan, and J. Beatty, ''An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models,'' *ACM TOG*, 6(2), April 1987, 123–158.

SECH82 Sechrest, S., and D.P. Greenberg, ''A Visible Polygon Reconstruction Algorithm,'' *ACM TOG*, 1(1), January 1982, 25–42.

SEDE84 Sederberg, T.W., and D.C. Anderson, ''Ray Tracing of Steiner Patches,'' *SIGGRAPH 84*, 159–164.

SEDE86 Sederberg, T.W., and S.R. Parry, ''Free-Form Deformation of Solid Geometric Models,'' *SIGGRAPH 86*, 151–160.

SEDG88 Sedgewick, R., *Algorithms*, second edition, Addison-Wesley, Reading, MA, 1988.

SELF79 Selfridge, P., and K. Sloan, *Raster Image File Format (RIFF): An Approach to Problems in Image Management*, TR61, Department of Computer Science, University of Rochester, Rochester, NY, 1979.

SELI89 Seligmann, D. and S. Feiner, ''Specifying Composite Illustrations with Communicative Goals,'' *Proceedings of ACM UIST '89*, ACM, New York, 1989, 1–9.

SEQU89 Séquin, C.H. and E.K. Smyrl, "Parameterized Ray Tracing," *SIGGRAPH 89*, 307–314.

SHAN87 Shantz, M., and S. Lien, "Shading Bicubic Patches," *SIGGRAPH 87*, 189–196.

SHAN89 Shantz, M. and S. Chang, "Rendering Trimmed NURBS with Adaptive Forward Differencing," *SIGGRAPH 89*, 189–198.

SHAO88 Shao, M.Z., Q.S. Peng, and Y.D. Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis," *SIGGRAPH 88*, 93–101.

SHAW91 Shaw, C.D., M. Green, and J. Schaeffer, "A VLSI Architecture for Image Composition," 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988. In Kuijk, A.A.M., ed., *Advances in Computer Graphics Hardware III*, Eurographics Seminars, Springer-Verlag, Berlin, 1991, 183–199.

SHER79 Sherr, S., *Electronic Displays*, Wiley, New York, 1979.

SHIN87 Shinya, M., T. Takahashi, and S. Naito, "Principles and Applications of Pencil Tracing," *SIGGRAPH 87*, 45–54.

SHIR86 Shires, G., "A New VLSI Graphics Coprocessor—The Intel 82786," *CG & A*, 6(10), October 1986, 49–55.

SHNE83 Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer*, 16(8), August 1983, 57–69.

SHNE86 Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1986.

SHOE85 Shoemake, K., "Animating Rotation with Quaternion Curves," *SIGGRAPH 85*, 245–254.

SHOU79 Shoup, R.G., "Color Table Animation," *SIGGRAPH 79*, 8–13.

SHOW89 Marketing Department, *Brochure*, Showscan Film Corp., Culver City, CA, 1989.

SIBE86 Sibert, J., W. Hurley, and T. Bleser, "An Object-Oriented User Interface Management System," *SIGGRAPH 86*, 259–268.

SIEG81 Siegel, R., and J. Howell, *Thermal Radiation Heat Transfer*, second edition, Hemisphere, Washington, DC, 1981.

SIG85 *Introduction to Image Processing. Course Notes 26 for SIGGRAPH 85*, San Francisco, California, July 1985.

SILL89 Sillion, F., and C. Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection," *SIGGRAPH 89*, 335–344.

SIMP85 Simpson, C., M. McCauley, E. Roland, J. Ruth, and B. Williges, "System Design for Speech Recognition and Generation," *Human Factors*, 27(2), April 1985, 115–142.

SIMP87 Simpson, C., M. McCauley, E. Roland, J. Ruth, and B. Williges, "Speech Control and Displays," in Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987, 1490–1525.

SIOC89 Siochi, A., and H. R. Hartson, "Task-Oriented Representation of Asynchronous User Interfaces," in *Proceedings CHI '89*, ACM, New York, 183–188.

SKLA90 Sklar, D., "Implementation Issues for SPHIGS (Simple PHIGS)," Technical Report, Computer Science Department, Brown University, Providence, RI, August 1990.

SLOA79 Sloan, K.R., and S.L. Tanimoto, "Progressive Refinement of Raster Images," *IEEE Transactions on Computers*, C-28(11), November 1979, 871–874.

SMIT78 Smith, A.R., "Color Gamut Transform Pairs," *SIGGRAPH 78*, 12–19.

SMIT79 Smith, A.R., "Tint Fill," *SIGGRAPH 79*, 276–283.

SMIT82 Smith, D., R. Kimball, B. Verplank, and E. Harslem, "Designing the Star User Interface," *Byte*, 7(4), April 1982, 242–282.

SMIT84 Smith, A.R., "Plants, Fractals and Formal Languages," *SIGGRAPH 84*, 1–10.

SMIT87 Smith, A.R., "Planar 2-pass Texture Mapping and Warping," *SIGGRAPH 87*, 263–272.

SMIT88 Smith, D.N., "Building Interfaces Interactively," in *Proceedings ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 144–151.

SMIT89 Smith, A.R., "Geometry vs. Imaging," in *Proceedings of NCGA '89*, Philadelphia, PA, April 1989, 359–366.

SNOW83 Snowberry, K., S. Parkinson, and N. Sisson, "Computer Display Menus," *Ergonomics*, 26(7), July 1983, 699–712.

SNYD85 Snyder, H., "Image Quality: Measures and Visual Performance," in TANN85, 70–90.

SNYD87 Snyder, J.M. and A.H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *SIGGRAPH 87*, 119–128.

SPAR78 Sparrow, E.M., and R.D. Cess, *Radiation Heat Transfer*, Hemisphere, Washington, DC, 1978.

SPRO82 Sproull, R.F., "Using Program Transformations to Derive Line-Drawing Algorithms," *ACM TOG*, 1(4), October 1982, 259–273.

SRIH81 Srihari, S., "Representation of Three-Dimensional Digital Images," *ACM Computing Surveys*, 13(4), December 1981, 399–424.

STAU78 Staudhammer, J., "On Display of Space Filling Atomic Models in Real Time," *SIGGRAPH 78*, 167–172.

STEI89 Steinhart, J., ed., *Introduction to Window Management*, *Course Notes 11 for SIGGRAPH 89*, Boston, MA, August 1989.

STER83 Stern, G., "Bbop—A System for 3D Keyframe Figure Animation," in *Introduction to Computer Animation*, *Course Notes 7 for SIGGRAPH 83*, New York, July 1983, 240–243.

STIN78 Stiny, G., and J. Gips, *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts*, University of California Press, Berkeley, 1978.

STON88 Stone, M., W. Cowan, and J. Beatty, "Color Gamut Mapping and the Printing of Digital Color Images," *ACM TOG*, 7(3), October 1988, 249–292.

STOV82 Stover, H., "True Three-Dimensional Display of Computer Data," in *Proceedings of SPIE*, 367, August 1982, 141–144.

STRA88 Strauss, P., *BAGS: The Brown Animation Generation System*, Ph.D. Thesis, Technical Report CS-88-22, Computer Science Department, Brown University, Providence, RI, May 1988.

SUKA88 Sukaviriya, P., "Dynamic Construction of Animated Help from Application Context," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 190–202.

SUKA90 Sukaviriya, P., and L. Moran, "User Interface for Asia," in Neilsen, J., ed., *Designing User Interfaces for International Use*, Elsevier, Amsterdam, 1990.

SUN86a Sun Microsystems, *Programmer's Reference Manual for the Sun Window System*, Sun Microsystems, Mountain View, CA, 1986.

SUN86b Sun Microsystems, *SunView™ Programmer's Guide*, Sun Microsystems, Mountain View, CA, 1986.

SUN87 Sun Microsystems, *NeWS™ Technical Overview*, Sun Microsystems, Mountain View, CA, 1987.

SUN89 Sun Microsystems, *OPEN LOOK Graphical User Interface*, Sun Microsystems, Mountain View, CA, 1989.

SUNF86 Sun Flex Corporation, *Touchpen*, Sun Flex, Novato, CA, 1986.

SUTH63 Sutherland, I.E., "Sketchpad: A Man–Machine Graphical Communication System," in *SJCC*, Spartan Books, Baltimore, MD, 1963.

SUTH65 Sutherland, I.E., "The Ultimate Display," in *Proceedings of the 1965 IFIP Congress*, 2, 1965, 506–508.

SUTH68 Sutherland, I.E., "A Head-Mounted Three Dimensional Display," in *FJCC 1968*, Thompson Books, Washington, DC, 757–764.

SUTH74a Sutherland, I.E., R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *ACM Computing Surveys*, 6(1), March 1974, 1–55. Also in BEAT82, 387–441.

SUTH74b Sutherland, I.E., and Hodgman, G.W., "Reentrant Polygon Clipping," *CACM*, 17(1), January 1974, 32–42.

SUTT78 Sutton, J., and R. Sprague, "A Survey of Business Applications," in *Proceedings American Institute for Decision Sciences 10th Annual Conference, Part II*, Atlanta, GA, 1978, 278.

SUYD86 Suydham, B., "Lexidata Does Instant Windows," *Computer Graphics World*, 9(2), February 1986, 57–58.

SWAN86 Swanson, R., and L. Thayer, "A Fast Shaded-Polygon Renderer," *SIGGRAPH 86*, 95–101.

SYMB85 Symbolics, Inc., *S-Dynamics*, Symbolics, Inc., Cambridge, MA, 1985.

TAMM82 Tamminen, M. and R. Sulonen, "The EXCELL Method for Efficient Geometric Access to Data," in *Proc. 19th ACM IEEE Design Automation Conf.*, Las Vegas, June 14–16, 1982, 345–351.

TAMM84 Tamminen, M., and H. Samet, "Efficient Octree Conversion by Connectivity Labeling," *SIGGRAPH 84*, 43–51.

TANA86 Tanaka, A.M., M. Kameyama, S. Kazama, and O. Watanabe, "A Rotation Method for Raster Images Using Skew Transformation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, June 1986, 272–277.

TANI77 Tanimoto, S.L., "A Graph-Theoretic Real-Time Visible Surface Editing Technique," *SIGGRAPH 77*, 223–228.

TANN85 Tannas, L. Jr., ed., *Flat-Panel Displays and CRTs*, Van Nostrand Reinhold, New York, 1985.

TEIT64 Teitelman, W., "Real-Time Recognition of Hand-Drawn Characters," in *FJCC 1964, AFIPS Conf. Proc.*, 24, Spartan Books, Baltimore, MD, 559.

TEIT86 Teitelman, W., "Ten Years of Window Systems—A Retrospective View," in Hopgood, F.R.A., *et al.*, eds., *Methodology of Window Management*, Springer-Verlag, New York, 1986, 35–46.

TERZ88 Terzopoulos, D., and K. Fleischer, "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," *SIGGRAPH 88*, 269–278.

TESL81 Tesler, L., "The Smalltalk Environment," *Byte*, 6(8), August 1981, 90–147.

TEXA89 Texas Instruments, Inc., *TMS34020 and TMS34082 User's Guide*, Texas Instruments, Dallas, TX, March 1989.

THIB87 Thibault, W.C., and B.F. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *SIGGRAPH 87*, 153–162.

THOM84 Thomas, S.W., *Modeling Volumes Bounded by B-Spline Surfaces*, Ph.D. Thesis, Technical Report UUCS-84-009, Department of Computer Science, University of Utah, Salt Lake City, UT, June 1984.

THOM86 Thomas, S.W., "Dispersive Refraction in Ray Tracing," *The Visual Computer*, 2(1), January 1986, 3–8.

THOR79 Thornton, R.W., "The Number Wheel: A Tablet-Based Valuator for Three-Dimensional Positioning," *SIGGRAPH 79*, 102–107.

TILB76 Tilbrook, D., *A Newspaper Page Layout System*, M.Sc. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1976. Also see *ACM SIGGRAPH Video Tape Review*, 1, May 1980.

TILL83 Tiller, W., "Rational B-Splines for Curve and Surface Representation," *CG & A*, 3(6), September 1983, 61–69.

TILO80 Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," *IEEE Trans. on Computers*, C-29(10), October 1980, 847–883.

TORB87 Torborg, J., "A Parallel Processor Architecture for Graphics Arithmetic Operations," *SIGGRAPH 87*, 197–204.

TORR66 Torrance, K.E., E.M. Sparrow, and R.C. Birkebak, "Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces," *J. Opt. Soc. Am.*, 56(7), July 1966, 916–925.

TORR67 Torrance, K., and E.M. Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces," *J. Opt. Soc. Am.*, 57(9), September 1967, 1105–1114.

TOTH85 Toth, D.L., "On Ray Tracing Parametric Surfaces," *SIGGRAPH 85*, 171–179.

TOUL70 Touloukian, Y.S., and D.P. DeWitt, eds., *Thermophysical Properties of Matter: The TPRC Data Series, Vol. 7 (Thermal Radiative Properties: Metallic Elements and Alloys)*, Plenum, New York, 1970.

TOUL72a Touloukian, Y.S., and D.P. DeWitt, eds., *Thermophysical Properties of Matter: The TPRC Data Series,Vol. 8 (Thermal Radiative Properties: Nonmetallic Solids)*, Plenum, New York, 1972.

TOUL72b Touloukian, Y.S., D.P. DeWitt, and R.S. Hernicz, eds., *Thermophysical Properties of Matter: The TPRC Data Series, Vol. 9 (Thermal Radiative Properties: Coatings)*, Plenum, New York, 1972.

TRAU67 Traub, A.C., "Stereoscopic Display Using Rapid Varifocal Mirror Oscillations," *Applied Optics*, 6(6), June 1967, 1085–1087.

TRIC87 Tricoles, G., "Computer Generated Holograms: an Historical Review," *Applied Optics*, 26(20), October 1987, 4351–4360.

TROW75	Trowbridge, T.S., and K.P. Reitz, "Average Irregularity Representation of a Rough Surface for Ray Reflection," *J. Opt. Soc. Am.*, 65(5), May 1975, 531–536.
TUFT83	Tufte, E., *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT, 1983.
TURK82	Turkowski, K., "Anti-Aliasing Through the Use of Coordinate Transformations," *ACM TOG*, 1(3), July 1982, 215–234.
TURN84	Turner, J.A., *A Set-Operation Algorithm for Two and Three-Dimensional Geometric Objects*, Architecture and Planning Research Laboratory, College of Architecture, University of Michigan, Ann Arbor, MI, August 1984.
ULIC87	Ulichney, R., *Digital Halftoning*, MIT Press, Cambridge, MA, 1987.
UPSO88	Upson, C., and M. Keeler, "V-BUFFER: Visible Volume Rendering," *SIGGRAPH 88*, 59–64.
UPST89	Upstill, S., *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*, Addison-Wesley, Reading, MA, 1989.
VANA84	Van Aken, J. R., "An Efficient Ellipse-Drawing Algorithm," *CG&A*, 4(9), September 1984, 24–35.
VANA85	Van Aken, J.R., and M. Novak, "Curve-Drawing Algorithms for Raster Displays, " *ACM TOG*, 4(2), April 1985, 147–169.
VANA89	Van Aken, J., personal communication, January 1989.
VANC72	Van Cott, H., and R. Kinkade, *Human Engineering Guide to Equipment Design*, 008-051-00050-0, U.S. Government Printing Office, Washington, DC, 1972.
VAND74	van Dam, A., G.M. Stabler, and R.J. Harrington, "Intelligent Satellites for Interactive Graphics," *Proceedings of the IEEE*, 62(4), April 1974, 483–492.
VERB84	Verbeck, C.P., and D.P. Greenberg, "A Comprehensive Light-Source Description for Computer Graphics," *CG & A*, 4(7), July 1984, 66–75.
VERS84	Versatron Corporation, *Footmouse*, Versatron Corporation, Healdsburg, CA, 1984.
VITT84	Vitter, J., "US&R: A New Framework for Redoing," *IEEE Software*, 1(4), October 1984, 39–52.
VOSS85a	Voss, R., "Random Fractal Forgeries," in Earnshaw, R.A., ed., *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Berlin, 1985; NATO ASI series F, volume 17, 805–835.
VOSS85b	Vossler, D., "Sweep-to-CSG Conversion Using Pattern Recognition Techniques," *CG & A*, 5(8), August 1985, 61–68.
VOSS87	Voss, R., "Fractals in Nature: Characterization, Measurement, and Simulation," in *Course Notes 15 for SIGGRAPH 87*, Anaheim, CA, July 1987.
WALD64	Wald, G., "The Receptors for Human Color Vision," *Science*, 145, 1964, 1007–1017.
WALL87	Wallace, J.R., M.F. Cohen, and D.P. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods," *SIGGRAPH 87*, 311–320.
WALL89	Wallace, J.R., K.A. Elmquist, and E.A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity," *SIGGRAPH 89*, 315–324.
WAN88	Wan, S., K. Wong, and P. Prusinkiewicz, "An Algorithm for Multidimensional Data Clustering," *ACM Transactions on Mathematical Software*, 14(2), June 1988, 153–162.
WARD85	Ward, J., and B. Blesser, "Interactive Recognition of Handprinted Characters for Computer Input," *CG & A*, 5(9), September 1985, 24–37.
WARD88	Ward, G.J., F.M. Rubinstein, and R.D. Clear, "A Ray Tracing Solution for Diffuse Interreflection," *SIGGRAPH 88*, 85–92.

WARE87 Ware, C., and J. Mikaelian, "An Evaluation of an Eye Tracker as a Device for Computer Input," in *Proceedings of CHI + GI 1987*, ACM, New York, 183–188.

WARE88 Ware, C., and J. Beatty, "Using Color Dimensions to Display Data Dimensions," *Human Factors*, 20(2), April 1988, 127–42.

WARN69 Warnock, J., *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, UT, June 1969.

WARN83 Warn, D.R., "Lighting Controls for Synthetic Images," *SIGGRAPH 83*, 13–21.

WASS85 Wasserman, A., "Extending Transition Diagrams for the Specification of Human-Computer Interaction," *IEEE Transactions on Software Engineering*, SE-11(8), August 1985, 699–713.

WATE87 Waters, K., "A Muscle Model for Animating Three-Dimensional Facial Expressions," *SIGGRAPH 87*, 17–24.

WATK70 Watkins, G.S., *A Real Time Visible Surface Algorithm*, Ph.D. Thesis, Technical Report UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, University of Utah, Salt Lake City, UT, June 1970.

WEGH84 Weghorst, H., G. Hooper, and D.P. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM TOG*, 3(1), January 1984, 52–69.

WEIL77 Weiler, K. and P. Atherton, "Hidden Surface Removal Using Polygon Area Sorting," *SIGGRAPH 77*, 214–222.

WEIL80 Weiler, K., "Polygon Comparison Using a Graph Representation," *SIGGRAPH 80*, 10–18.

WEIL85 Weiler, K., "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments," *CG & A*, 5(1), January 1985, 21–40.

WEIL86 Weil, J., "The Synthesis of Cloth Objects," *SIGGRAPH 86*, 49–54.

WEIL87 Weil, J., "Animating Cloth Objects," personal communication, 1987.

WEIL88 Weiler, K., "The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Modeling," in Wozny, M. J., H. McLaughlin, and J. Encarnação, eds., *Geometric Modeling for CAD Applications, IFIP WG5.2 Working Conference, Rensselaerville, NY, 12–14 May 1986*, North-Holland, 1988, 3–36.

WEIM80 Weiman, C.F.R., "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *SIGGRAPH 80*, 286–293.

WEIN81 Weinberg, R., "Parallel Processing Image Synthesis and Anti-Aliasing," *SIGGRAPH 81*, 55–61.

WEIN87 Weingarten, N., personal communication, 1987.

WEIN88 Weinand, A., E. Gamma, and R. Marty, "ET++—An Object Oriented Application Framework in C++," *OOPSLA 1988 Proceedings*, ACM-SIGPLAN Notices, 23(11), November 1988, 46–57.

WEIS66 Weiss, R.A., "BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces," *JACM*, 13(2), April 1966, 194–204. Also in FREE80, 203–213.

WELL76 Weller, D., and R. Williams, "Graphic and Relational Data Base Support for Problem Solving," *SIGGRAPH 76*, 183–189.

WELL89 Wellner, P., "Statemaster: A UIMS Based on Statecharts for Prototyping and Target Implementation," in *Proceedings of CHI '89*, ACM, New York, 177–182.

WERT39 Wertheimer, M., "Laws of Organization in Perceptual Forms," in Ellis, W.D., ed., *A Source Book of Gestalt Psychology*, Harcourt Brace, New York, 1939.

WESL81 Wesley, M.A., and G. Markowsky, "Fleshing Out Projections," *IBM Journal of Research and Development*, 25(6), November 1981, 934–954.

WEST89 Westover, L., "Interactive Volume Rendering," in *Proceedings of Volume Visualization Workshop*, Department of Computer Science, University of North Carolina at Chapel Hill, May 18-19, 1989, 9–16.

WHEL82 Whelan, D., "A Rectangular Area Filling Display System Architecture," *SIGGRAPH 82*, 147–153.

WHIT80 Whitted, T., "An Improved Illumination Model for Shaded Display," *CACM*, 23(6), June 1980, 343–349.

WHIT82 Whitted, T., and S. Weimer, "A Software Testbed for the Development of 3D Raster Graphics Systems," *ACM TOG*, 1(1), January 1982, 43–58.

WHIT83 Whitted, T., "Anti-Aliased Line Drawing Using Brush Extrusion," *SIGGRAPH 83*, 151–156.

WHIT84 Whitton, M., "Memory Design for Raster Graphics Displays," *CG & A*, 4(3), March 1984, 48–65.

WHIT85 Whitted, T., "The Hacker's Guide to Making Pretty Pictures," in *Image Rendering Tricks, Course Notes 12 for SIGGRAPH 85*, New York, July 1985.

WILH87 Wilhelms, J., "Using Dynamic Analysis for Realistic Animation of Articulated Bodies," *CG & A*, 7(6), June 1987, 12–27.

WILL72 Williamson, H., "Algorithm 420 Hidden-Line Plotting Program," *CACM*, 15(2), February 1972, 100–103.

WILL78 Williams, L., "Casting Curved Shadows on Curved Surfaces," *SIGGRAPH 78*, 270–274.

WILL83 Williams, L., "Pyramidal Parametrics," *SIGGRAPH 83*, 1–11.

WITK87 Witkin, A., K. Fleischer, and A. Barr, "Energy Constraints on Parameterized Models," *SIGGRAPH 87*, 225–232.

WITK88 Witkin, A., and M. Kass, "Spacetime Constraints," *SIGGRAPH 88*, 159–168.

WOLB88 Wolberg, G., *Geometric Transformation Techniques for Digital Images: A Survey*, Technical Report CUCS-390-88, Department of Computer Science, Columbia University, New York, December 1988. To appear as Wolberg, G., *Digital Image Warping*, IEEE Computer Society, Washington, DC, 1990.

WOLB89 Wolberg, G., and T.E. Boult, *Separable Image Warping with Spatial Lookup Tables*, SIGGRAPH 89, 369–378.

WOLB90 Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

WOLF87 Wolf, C., and P. Morel-Samuels, "The Use of Hand-Drawn Gestures for Text Editing," *International Journal of Man-Machine Studies*, 27(1), July 1987, 91-102.

WOLF90 Wolff, L., and D. Kurlander, "Ray Tracing with Polarization Parameters," *CG&A*, 10(6), November 1990, 44–55.

WOO85 Woo, T., "A Combinatorial Analysis of Boundary Data Structure Schemata," *CG & A*, 5(3), March 1985, 19–27.

WOOD70 Woods, W., "Transition Network Grammars for Natural Language Analysis," *CACM*, 13 (10), October 1970, 591–606.

WOOD76 Woodsford, P. A., "The HRD-1 Laser Display System," *SIGGRAPH 76*, 68–73.

WOON71 Woon, P.Y., and H. Freeman, "A Procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces," in *IFIP 1971*, North-Holland Pub. Co., Amsterdam, 1971, pp. 1120–1125. Also in FREE80, 230–235.

WRIG73 Wright, T.J., "A Two Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables," *IEEE Trans. on Computers*, 22(1), January 1973, 28–33. Also in FREE80, 284–289.

WU89 Wu, X., and J.G. Rokne, "Double-Step Generation of Ellipses," *CG & A*, 9(3), May 1989, 56–69.

WYLI67 Wylie, C., G.W. Romney, D.C. Evans, and A.C. Erdahl, "Halftone Perspective Drawings by Computer," *FJCC 67*, Thompson Books, Washington, DC, 1967, 49–58.

WYSZ82 Wyszecki, G., and W. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second edition, Wiley, New York, 1982.

WYVI86 Wyvill, G., C. McPheeters, and B. Wyvill, "Data Structures for Soft Objects," *The Visual Computer*, 2(4), April 1986, 227–234.

WYVI88 Wyvill, B., "The Great Train Rubbery," *ACM SIGGRAPH 88 Electronic Theater and Video Review*, 26, 1988.

YEAG86 Yeager, L., and C. Upson, "Combining Physical and Visual Simulation—Creation of the Planet Jupiter for the Film '2010'," *SIGGRAPH 86*, 85–93.

ZDON90 Zdonik, S.B., and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, CA, 1990.

ZELT82 Zeltzer, D., "Motor Control Techniques for Figure Animation," *CG & A*, 2(11), November 1982, 53–59.

ZIMM87 Zimmerman, T., J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A Hand Gesture Interface Device," in *Proceedings of the CHI* + GI 1987 Conference, ACM, New York, 189–192.

Index

Graphics Software Packages: SRGP and SPHIGS

The SRGP and SPHIGS graphics packages described in this book are archived in multiple formats, and are available on the World Wide Web free for your use.

http//:www.aw.com/cseng/authors/foley/compgrafix/compgrafix.sup.html

These formats allow you to run the packages on many PC, Apple Macintosh, and UNIX platforms. The files are identical across platforms except for the method used in compressing or archiving them. The website indicated above includes specific directions for accessing each format.

PLEASE NOTE BELOW THE SPECIFIC PLATFORMS REQUIRED FOR EACH FORMAT. THE SOFTWARE MAY NOT INSTALL OR RUN PROPERLY ON ANY OTHER PLATFORM—UNDER SOME OTHER C COMPILER, FOR EXAMPLE. WE REGRET THAT WE ARE UNABLE TO OFFER ANY USER SUPPORT IN SUCH CASES.

Requirements:

UNIX Worksations: Requires a workstation running UNIX and the X Window System; X11 release R4 or later; an ANSII C Compiler (gcc is recommended); v4.3 or 4.4 BSD, System V UNIX, or Solaris 2.0.

Apple Macintosh: Requires any model Apple Macintosh with a minimum of 1 megabyte of RAM; 2 megabytes of RAM are required to run the debugger; System Software v7.0 or later; Metrowerks CodeWarrior v.10 or later.

Microsoft Windows for the PC Family: Requires any PC using an 80826 or higher microprocessor with a minimum of 1 megabyte of RAM (combined conventional and extended memory); Hercules monochrome adapter, or EGA color monitor or better; Microsoft Mouse or compatible pointing device; Microsoft Windows v3.1, Windows95, or DOS v5.0 or later; Microsoft Software Development Kit for Windows; Borland Turbo C v2.0 or later.

Instructors Note:

Instructors who adopt this book may obtain a free copy of the Apple Macintosh or Microsoft Windows files on a diskette. Contact your local Addison Wesley Longman representative, send e-mail to aw.cse@aw.com, or (in the U.S.) call 1-800-322-1377. Be sure to specify the format you need.

Plate E Ray-traced radiosity simulation, by K. Howie, B. Trumbore, and D.P. Greenberg, Program of Computer Graphics, Cornell University. (Copyright © 1989 Cornell University, Program of Computer Graphics.)

Plate G "Onlyville," by S. Snibbe, and D. Robbins. (Copyright © 1989 Brown University Computer Graphics Group.)

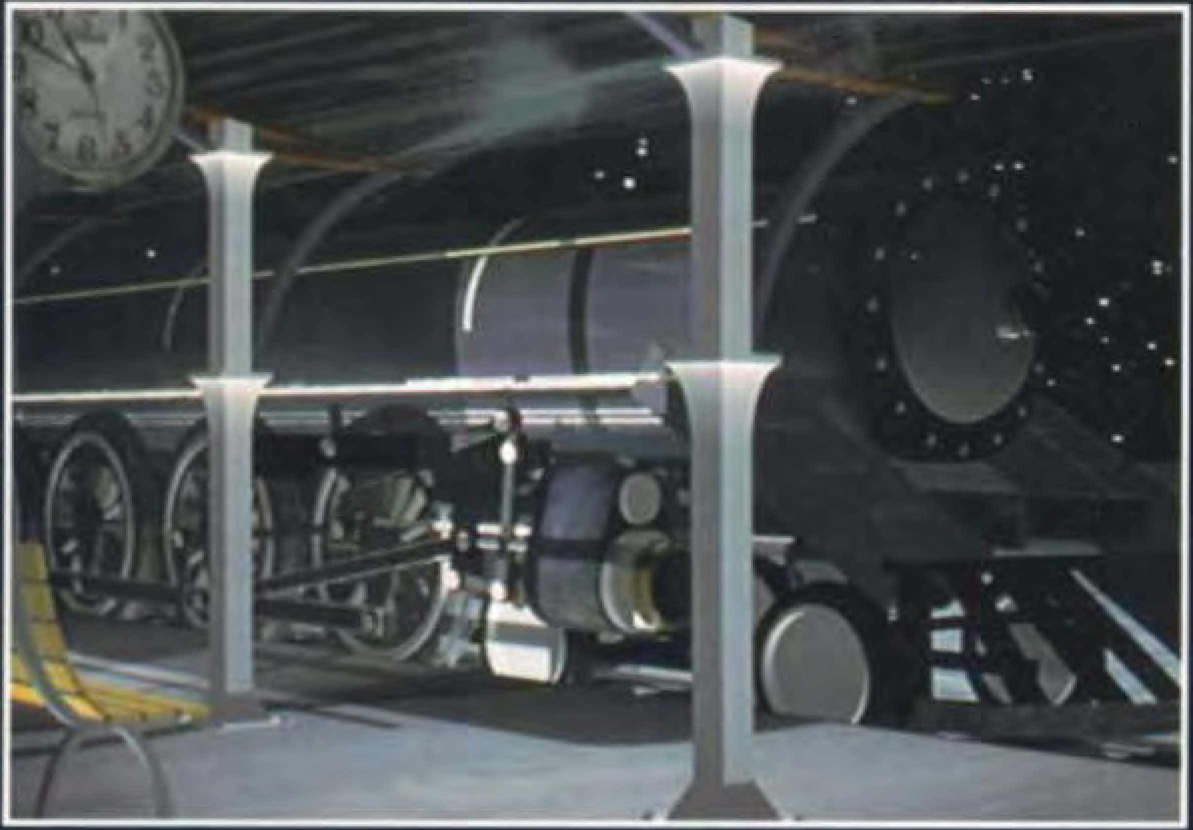


Plate F "Red's Dream," by J. Lasseter, E. Ostby, W. Reeves, and H.B. Siegel. (Copyright © 1987 Pixar.)

Plate H "Color Choreography." Four color spaces are shown: HSV hexcone, the Ostwald double cone, the RGB cube, and the CIE color space. (Image by D. Laidlaw and B. Meier; Copyright © Laidlaw/Meier, Brown University Computer Graphics Group.)

ISBN 0-201-84840-6