A Generalization of Algebraic Surface Drawing

JAMES F. BLINN
Jet Propulsion Laboratory

The mathematical description of three-dimensional surfaces usually falls into one of two classifications: parametric and implicit. An implicit surface is defined to be all points which satisfy some equation $F(x, y, z) = 0$. This form is ideally suited for image space shaded picture drawing; the pixel coordinates are substituted for x and y, and the equation is solved for z. Algorithms for drawing such objects have been developed primarily for first- and second-order polynomial functions, a subcategory known as algebraic surfaces. This paper presents a new algorithm applicable to other functional forms, in particular to the summation of several Gaussian density distributions. The algorithm was created to model electron density maps of molecular structures, but it can be used for other artistically interesting shapes.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*display algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representations*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*visible line/surface algorithms*

General Terms: Algorithms, Performance

1. INTRODUCTION

The technology of creating realistic and visually interesting images of three-dimensional shapes is advancing on many fronts. One such front is the development of algorithms for drawing curved surfaces directly from their mathematical definitions rather than by dividing them into large numbers of polygons. Two classes of surfaces which have received attention are the quadric and the bivariate parametric surfaces. Bivariate parametric surfaces are generated by three functions of two variables (most popularly polynomials), as the variables take on different values. Algorithms dealing with such surfaces are due to Catmull [2]; Lane, Carpenter, Whitted and Blinn [7]; and Clark [3].

The research described in this paper was carried out by the Jet Propulsion Laboratory at the California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Author's address: Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, MS 201-209, Pasadena, CA 91109.

Quadric surfaces, on the other hand, are solutions to second-order functions of the coordinates:

$$\begin{aligned} ax^2 + bxy + cxz + dx \\ + ey^2 + fyz + gy \\ + hz^2 + iz \\ + j = 0. \end{aligned}$$

This class of surfaces includes such shapes as spheres, cones, hyperboloids of revolution, and so forth. Algorithms for dealing with such shapes have not been widely published but have been implemented by MAGI [5], Blinn [1], Duff [4], and General Motors research labs [10]. While this class of shapes is somewhat restricted, it can be used to advantage in modeling many useful objects such as machine tool parts, as long as the modeling system allows a fragment of a shape to be employed as well as the entire shape.

Quadrics belong to the class of curved surfaces known as "implicit" surfaces, that is, solutions to some equation

$$F(x, y, z) = 0.$$

This paper examines a more general solution to the imaging problem for such surfaces and describes in detail its application to a class of surfaces which are closely allied to quadrics but have a wider range of shapes.

2. THE MODEL

The problem which motivated this paper is the familiar one in computer graphics of displaying molecular models. This is most often done with a ball-and-stick model or a space-filling-sphere model. In either case, the model consists of a possibly intersecting collection of two basic shapes: spheres and cylinders. To draw a picture of the model, the spheres and cylinders can easily be broken down into polygons and passed to a conventional polygon-rendering algorithm. Alternatively, any of several curved surface algorithms may be employed, and, in fact, several special-purpose algorithms [6, 8, 9] have been formulated to handle efficiently just these two shapes for rapid display of large molecular structures.

In the interests both of artistic variety and scientific accuracy, a new model was sought that breaks away from the ball-and-stick/space-filling mold. It was desired to make the bonds between atoms appear more like those shown in Figure 1. This is, in fact, more like what a real electron density cloud might look like for a covalent bond. In addition, for the purposes of animation, this bond must stretch and contract in a pleasing manner as it vibrates, ultimately snapping apart as an atom is pulled completely away from the molecule. This is illustrated in Figure 2.

A conventional approach might be to model such a shape via the already familiar bicubic or quadric surfaces. This is moderately feasible for one isolated bond but becomes difficult for more elaborate molecules with several overlapping bonds (e.g., ring structures). In addition, the topological changes that must occur when a bond breaks are difficult to deal with in an automated manner. For these reasons a basic mathematical model was used which is similar in form to an

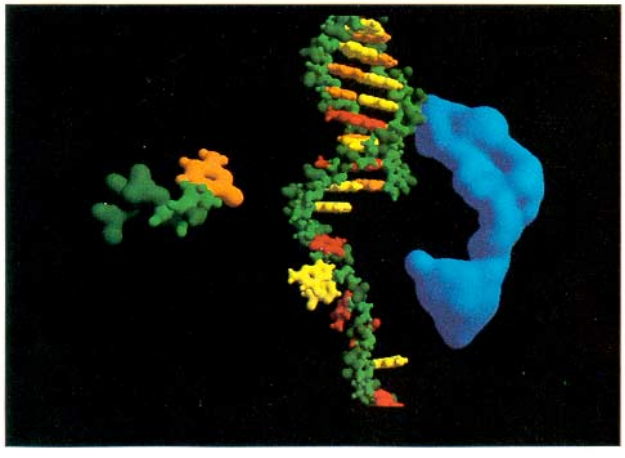


Fig. 1. Desired object appearance.

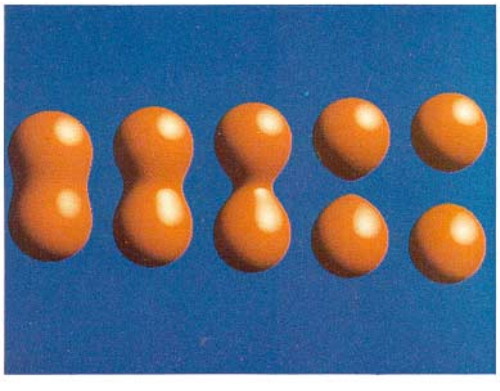


Fig. 2. Bond stretching and breaking.

actual simulation of electron density maps. Quantum mechanics represents the electron in an atom as a density function of the spatial location. A sample function for a hydrogen atom is

$$D(x, y, z) = \exp(-ar)$$

where

$$r = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2}$$

(x_1, y_1, z_1) = location of atom

We can represent this function for a collection of atoms by summing the contribution from each atom separately.

$$D(x, y, z) = \sum_i b_i \exp(-a_i r_i)$$

where r_i = distance from x, y, z to the center of atom i.

A surface can be defined as those points where this density function equals some threshold amount:

$$F(x, y, z) = D(x, y, z) - T.$$

Note that all points inside the surface have electron densities greater than T. For the purposes of computational efficiency, the function actually implemented was similarly shaped:

$$D(x, y, z) = \sum_i b_i \exp(-a_i r_i^2)$$

This does not require taking a square root. The exponential term is a simple Gaussian bump centered at r_i, with height b_i and standard devation a_i. By adjusting the a_i and b_i parameters, different effects can be achieved for the same atom arrangement. These effects alter the "blobbiness" of the object. In fact, for modeling purposes, it is more useful for a designer to specify these two parameters in terms of the radius of the atom in isolation and a blobbiness parameter. The radius of an isolated atom R_i is found by setting

$$T = b_i \exp(-a_i R_i^2) = \exp(-a_i R_i^2 + \ln b_i)$$

so the a_i can be chosen to be

$$a_i = -\frac{\ln(T/b_i)}{R_i^2}.$$

We can define the blobbiness parameter to be

$$B_i = \ln\left(\frac{T}{b_i}\right)$$

so that (solving for b_i)

$$b_i = T \exp(-B_i)$$

The net density contribution of one atom in terms of the two shape-defining parameters R_i and B_i is

$$D_i(x, y, z) = T \exp\left(\frac{\mathrm{B}_i}{\mathrm{R}_i^2}\,\mathrm{r}_i^2 - \mathrm{B}_i\right)$$

Note that B_i must be negative to ensure that the desity function goes to zero as r_i goes to infinity.

Since there is a factor of T in each contributing atom term the value of the threshold is now irrelevant; we can set it to some canonical value such as 1. One can get the same effect as changing the threshold by adjusting the scale factors, b_i, of the individual terms (i.e., by adjusting the blobbiness parameters B_i). For clarity, though, we usually write the threshold as T, with the understanding that $T = 1$. A canonical threshold value of 1 is particularly convenient since its logarithm is 0. The surface defined by an isolated atom, defined by setting $D_i = T$, is then a conventional quadric surface. This is seen by taking the logarithm of both sides of the above equation yielding $0 = (B_i/R_i^2)r_i^2 - B_i$. A sample image showing a range of such parameters is shown in Figure 3.

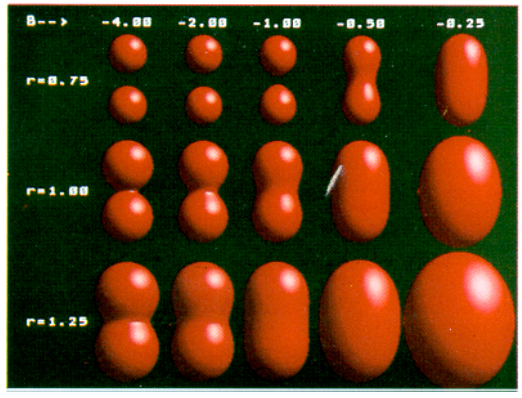


Fig. 3. Object appearance for different blobbiness and radius values.

3. THE RENDERING ALGORITHM

Surfaces defined algebraically are inherently well suited to raster conversion algorithms. The general structure of such an algorithm is straightforward: for each (x_s, y_s) pixel location the defining algebraic equation reduces to a univariate equation in z. Solutions to this equation (if any) yield the z depth of the surface at that pixel. In the case of the more common quadric surfaces this solution is easy to obtain. In the case of the more general surfaces described here, the solution must be obtained numerically. The important part of the algorithm described here is a technique for speeding up this computation.

3.1 Coordinate Systems

We begin by defining the various coordinate systems to be used. All atoms are transformed into a standard *viewing* space, with the eye looking down the $+z$ axis, the x axis pointing to the right, and the y axis pointing upward (a left-handed coordinate system). In this space the sizes and shapes of viewed objects are not altered, only their orientations and positions are adjusted according to the viewing direction.

A perspective image is then obtained with the eye positioned at the origin. This yields a homogeneous perspective matrix in terms of the field of view (Fov) and the locations of the near and far clipping planes (z_n and z_f).

$$P = \begin{bmatrix} C & 0 & 0 & 0 \\ 0 & C & 0 & 0 \\ 0 & 0 & Q & S \\ 0 & 0 & R & 0 \end{bmatrix} \qquad \begin{aligned} \text{where } C &= \cos(\text{Fov}/2) \\ S &= \sin(\text{Fov}/2) \\ Q &= -S\, z_f/(z_n - z_f) \\ R &= -Q\, z_n \end{aligned}$$

This transformation will be followed by a scale and translation in x and y (the "viewport" transformation) which converts from the canonical -1 to $+1$ visible

coordinate space into hardware screen (pixel) coordinates. Note that the x and y scale might not be equal if pixels in the screen space are not square.

$$V = \begin{bmatrix} x_w & 0 & 0 & 0 \\ 0 & y_w & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_c & y_c & 0 & 1 \end{bmatrix}$$

These two transformations are concatenated with the viewing transformation to make one net transformation directly into *perspective* space. In this space, the shapes of objects have been distorted in such a manner that a simple orthographic projection onto the $z = 0$ plane yields the correct perspective picture. It must be realized that this perspective space is just an efficiency measure. It represents removing the perspective calculations from the rendering loop; the perspective projection (e.g., of polygon vertices) is done simultaneously with the modeling and viewing transformations via one matrix multiplication. This merging of transformations also works for rational bicubic and quadric surfaces since these classes of surfaces are closed under the perspective transformation.

There are some calculations, however, which must still be done in the undistorted viewing space (e.g., light reflection calculations) since they depend on geometric distances and vector dot products. In fact, for the functions used here, the solution of the function itself can be done about as easily in viewing space as in perspective space. For this reason the merging of the perspective/viewport transformations and the modeling/viewing transformations is not used for this algorithm. All the atom centers and radii are transformed into viewing space, and pixel by pixel calculations are done by explicit rays from the origin through the pixel. A point on such a ray is defined by its z depth and the perspective and screen parameters. We find this relationship by first transforming a point on the ray into perspective space.

$$(x\ y\ z\ 1)[P][V] = (x'y'z'w') = (xCx_w + zSx_c, yCy_w + zSy_c, zQ + R, zS)$$

The pixel coordinates (x_s, y_s) are formed by the homogeneous division

$$x_s = x'/w' = (xCx_w + zSx_c)/(zS)$$
$$y_s = y'/w' = (yCy_w + zSy_c)/(zS)$$

solving for the viewing space (x, y) in terms of viewing space z yields

$$x = z(x_s - x_c)S/(Cx_w) \equiv zx_z \qquad (*)$$
$$y = z(y_s - y_c)S/(Cy_w) \equiv zy_z.$$

Thus we can define the viewing ray for a particular pixel as

$$\bar{R} = (x_z, y_z, 1).$$

Any point on this ray is a scalar multiple of this vector and can be parameterized by its z coordinate, yielding

$$(x, y, z) = z\bar{R}.$$

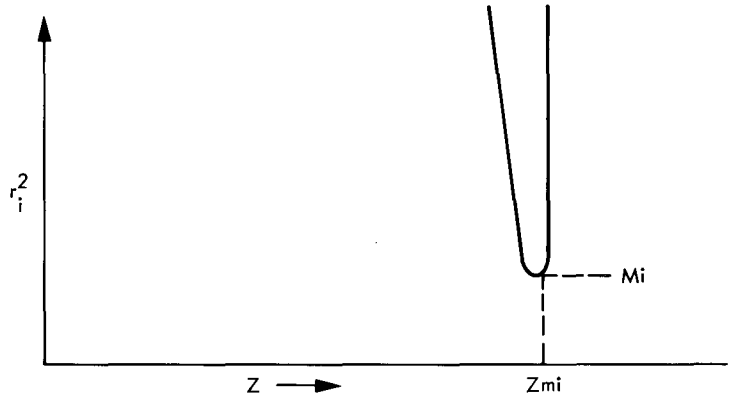


Fig. 4. Cause of round-off error.

3.2 Basic Algorithm

For a particular pixel, the square of the distance from the center of atom i, P_i, to a point on the viewing ray, $z\bar{R}$, reduces to a quadratic polynomial in z.

$$r_i^2 = (z\bar{R} - \bar{P}_i)\cdot(z\bar{R} - \bar{P}_i)$$
$$= z^2(\bar{R}\cdot\bar{R}) - 2z(\bar{R}\cdot\bar{P}_i) + (\bar{P}_i\cdot\bar{P}_i)$$

This expression is algebraically correct but it is unfortunately susceptible to round-off error. The reason for this can be seen from Figure 4.

The function can be a quite narrow parabola centered possibly quite far back on the z axis. While commonly encountered values for the coefficients of this equation do not themselves present problems, solution of the equation requires taking differences of their products. This can easily exceed the accuracy of single precision arithmetic. To avoid the necessity of multiple precision arithmetic we adopt a more geometrically meaningful representation

$$r_i^2 = (\bar{R}\cdot\bar{R})(z - z_{mi}^2) + M_i$$

where

$$z_{mi} = (\bar{R}\cdot\bar{P}_i)/(\bar{R}\cdot\bar{R})$$
$$M_i = (z_{mi}\bar{R} - \bar{P}_i)\cdot(z_{mi}\bar{R} - \bar{P}_i)$$

Here, z_{mi} is the z distance of the local minimum, M_i, of r_i^2. Each term in the density function is thus a Gaussian bump function of z centered at z_{mi}. The total function is the sum of several such bumps. The visible z depth value is the first location where this function exceeds the value T. This is shown in Figure 5.

If only one atom is visible, the z depth can be found analytically by setting the density term for that atom equal to the threshold value of 1.

$$T = 1 = \exp(-a_i r_i^2 - B_i)$$

Taking the logarithm of both sides and substituting our formula for r_i^2,

$$0 = a_i[(\bar{R}\cdot\bar{R})(z - z_{mi}^2) + M_i] + B_i.$$

Solving for z (note that the negative square root is taken to get the solution

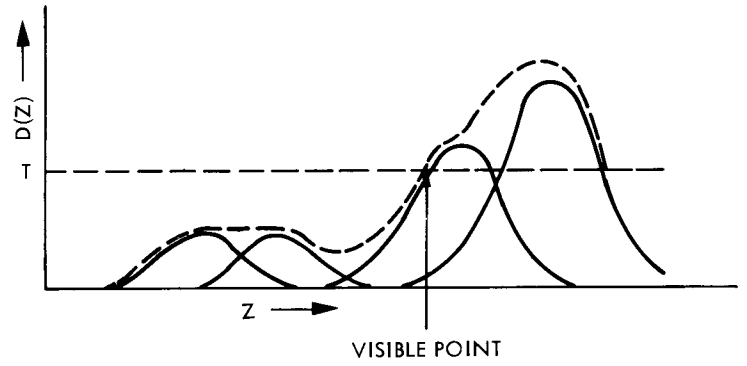


Fig. 5. Density function of z depth. (Dotted line is sum of D_i.)

closest to the eye),

$$z = z_{mi} - \sqrt{\frac{a_i M_i + B_i}{-a_i(\bar{R}\cdot\bar{R})}}.$$

3.3 Iterative Solution

If there is more than one atom, an analytic solution is not feasible and we must resort to numerical methods. Two popular methods for the iterative solution of such equations are Newton iteration and "regula falsi."

Newton iteration works by beginning with an initial guess and refining it by approximating the function D with a straight line tangent to the function at that point. Solving this linear equation yields a new guess, z_{new}.

$$z_{\text{new}} = z - \frac{D(z) - T}{D'(z)}$$

Derivatives are easily obtained from the functional form.

$$\frac{dD}{dz} = D' = \sum_i -2\, a_i(\bar{R}\cdot\bar{R})(z - z_{mi}) \exp(-a_i r_i^2 - B_i)$$

Note that this calculation uses many computations in common with the calculation of D, so evaluation of both D and D' is relatively economical.

Regula falsi begins with two initial guesses which are known to bracket the solution: z_n, where $D(z_n) < T$, and z_f, where $D(z_f) > T$. It generates a new guess by drawing a line between $(z_n, D(z_n))$ and $(z_f, D(z_f))$ and solving for T.

$$z_{\text{new}} = \frac{z_n(D(z_f) - T) - z_f(D(z_n) - T)}{D(z_f) - D(z_n)}$$

The real value of $D(z_{\text{new}})$ is then calculated. If $D(z_{\text{new}}) < T$ then z_{new} replaces z_n. Otherwise it replaces z_f. Thus the range between z_n and z_f continually contracts around the correct solution.

If the initial guess is close enough to the real solution, Newton iteration converges rapidly. If it is not close, however, it will diverge. Regula falsi is guaranteed to converge but does so more slowly. We therefore have adopted a

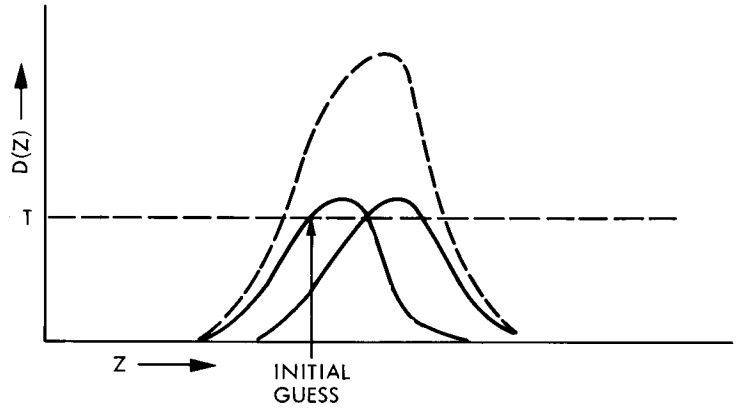


Fig. 6. Initial guess when $D_{max} > T$. (Dotted line is sum of D_i.)

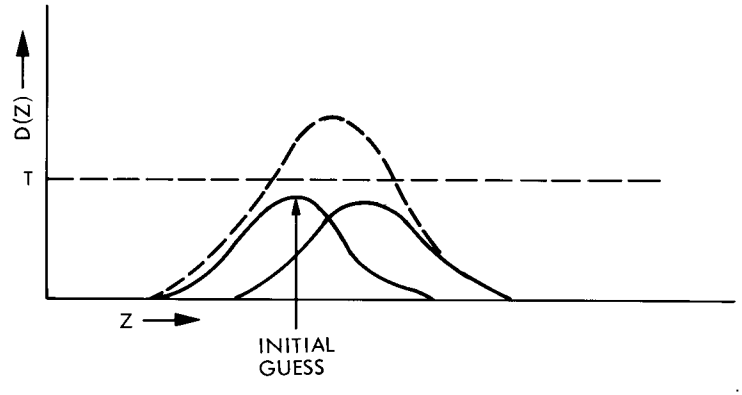


Fig. 7. Initial guess when $D_{max} < T$. (Dotted line is sum of D_i.)

hybrid solution. A value for z_{new} is calculated by Newton iteration. If this value is outside the range (z_n, z_f) then the value is recalculated from the regula falsi formula. This process is repeated until the value of $|D(z_{new}) - T|$ is less than some error tolerance t.

To generate the first initial guess range, we rely on some heuristics based on our knowledge of the functional form. We expect the solution to be at or near solutions to either the individual atom bumps (Figure 6) or to the local maximum of a bump if it does not itself exceed the threshold value T (Figure 7).

We therefore make a list of potential initial guess z values and sort them in ascending order of z. The sorted list is then scanned from front to back, and, for each element, the actual function value (i.e., the sum of all atom contributions) is calculated. If this is less than the threshold value, it is assumed that the local maximum of D near here does not reach the threshold and the next list item is evaluated. If it exceeds the threshold value, that z is used as the initial value of z_f and the previous list item is used as the initial value of z_n. See Figure 8.

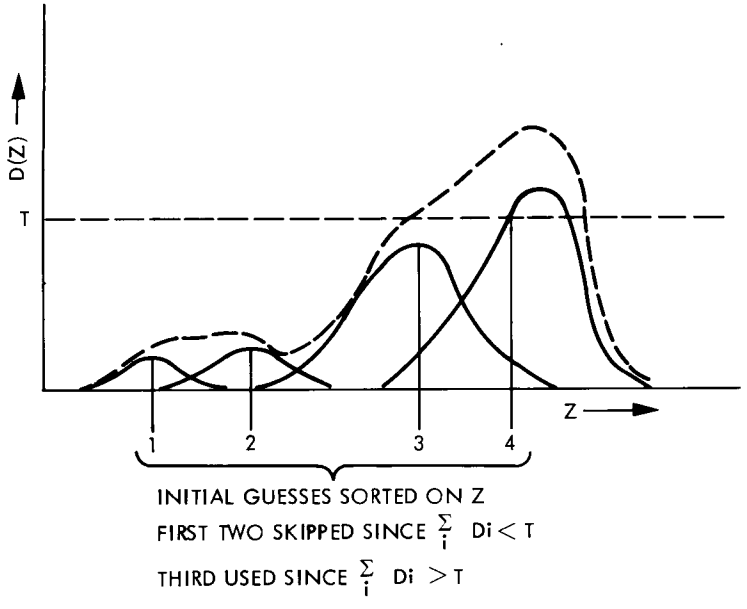


Fig. 8. Initial guess list.

3.4 Intensity Calculations

When a z solution is found, it is substituted into eq. (*) to get the x and y locations of the visible point on the surface in viewing space. To calculate an intensity, it is then necessary to find the surface normal at this point. This can be found by taking the gradient of the surface defining function, F.

$$N = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right]$$

For the function we are using here this is readily done. For example, the x component will be

$$\frac{\partial F}{\partial x} = \sum_i -2a_i(x - x_i) \exp(-a_i r_i^2 - B_i).$$

Finally, we can allow the surface reflective properties (such as color) to vary over the surface by blending them according to the contributions from each atom. This is done by taking a weighted sum of the surface property from each atom, the weight being chosen as the value of D_i from that atom. (Recall that these sum to the threshold value of 1.0.) Alternatively, as in the case of the diagrams shown here, we can find the atom with the highest value of D_i at the visible point and use its surface properties.

3.5 Optimizing the Algorithm

For images containing more than a few atoms (up to 4000 in some of the images required for this project), the summation of the D function over all the atoms is computationally out of the question. Fortunately, for any given pixel, most of the

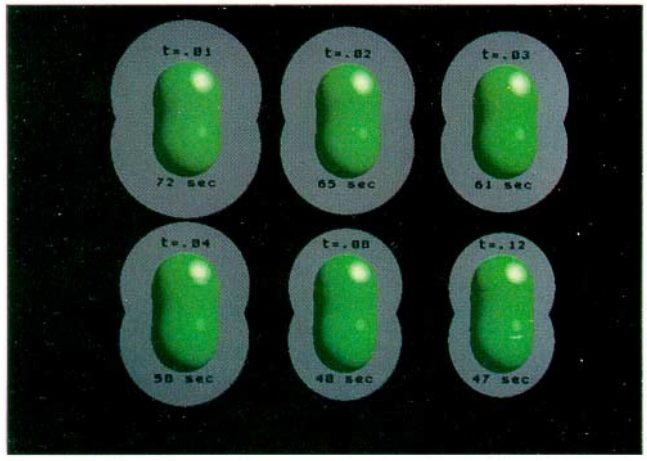


Fig. 9. Effect of different error tolerances, t.

atoms are sufficiently far away from the scan ray so that their contribution to the D function is negligible. We can therefore economize considerably by using in calculations only those atoms which are "close" to the scan ray. The term "close" is given precise meaning by enclosing each atom in a sphere defined by

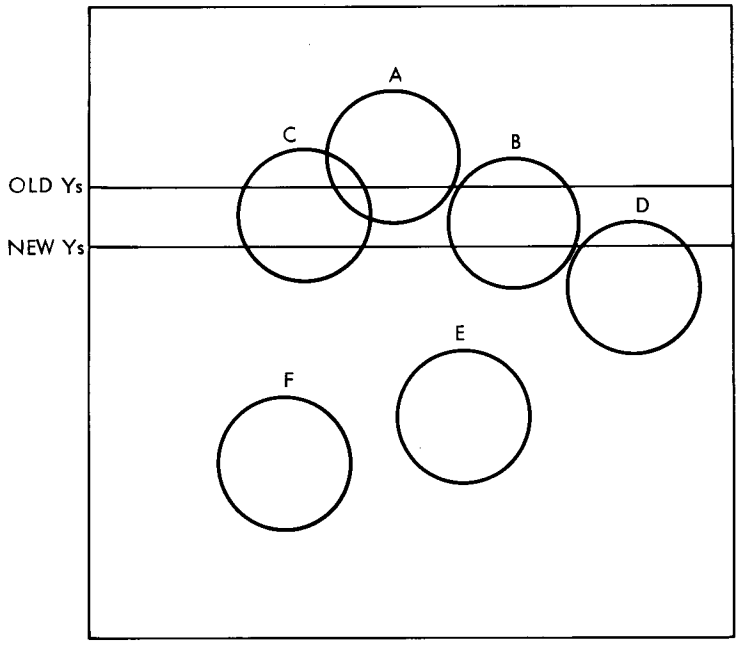
$$D_i(x, y, z) = tT,$$

where the value t is the same error tolerance used as the convergence criterion. If the scan ray intersects this sphere, there must be points along it where the contribution to D is large enough to matter. If, on the other hand, the scan ray is disjoint from the enclosing sphere, then all points on it contribute less than the error introduced by the termination conditions of the numeric solution. The atom can then be skipped.

The value of the error tolerance, of course, determines the quality of the image. A larger tolerance will be faster but the image will have some noise added. Figure 9 shows the results obtained with different values of t. The halo around each example indicates those pixels covered by the enclosing spheres of the atoms. The errors in the surface begin to be apparent with $t = 0.03$. They show up initially as a crease through the highlight at the top of the shape.

The process of maintaining a list of "close" atoms during the rendering is similar to that of maintaining a list of potentially visible polygons in more conventional poygon-rendering algorithms (or perhaps, more properly, maintaining a list of visible spheres in a sphere-drawing program).

We begin with the outer (y) loop. To initialize the loop, the enclosing spheres of all atoms are projected into screen space and the maximum and minimum visible y values are computed. The exact mathematics for this is given in the next section. The atom list is then sorted on minimum y, forming the "y-enter" list. During the y scan loop we maintain an additional "y-active" list of all atoms whose enclosing sphere includes the current scan plane. Additions to and deletions from this list are done incrementally each time through the loop. That is, each



Y-ENTER AT INITIALIZATION: A, B, C, D, E, F

Y-ENTER AT OLD Ys: D, E, F
Y-ACTIVE AT OLD Ys: A, B, C

Y-ENTER AT NEW Ys: E, F
Y-ACTIVE AT NEW Ys: B, C, D

Fig. 10. y-enter/y-active lists.

time y_s is incremented the top element on the y-enter list is examined. If it is now above the new y_s it is moved to the y-active list and the next element on the y-enter list is examined. When the top of the y-enter list is below the new y_s then we know that all entering atoms have been added. Now examine the y-active list for deletions. The $y_{\min}$ of each atom on the y-active list is tested against the new y_s, and if it is above it the element is removed. See Figure 10.

Inside the y loop there is an x loop which goes across the screen. Here we maintain an x-active list with the candidates coming from the y-active list. This is done in a manner exactly analogous to the y loop. For each element in the y-active list the enclosing sphere is projected onto the screen and the maximum and minimum x value is computed. The y-active list is then sorted on $x_{\min}$ and becomes the x-enter list. Since the y-active list is in fact identical to the x-enter list it is always kept in x-sorted order, from scan line to scan line. Any additions are merged into the list using an exchange sort. As the scan progresses from left to right the x-enter list is examined for entering atoms to add to the x-active list. The x-active list is then scanned for any exiting atoms. See Figure 11.

At this point we have, for a given pixel, a list of all atoms whose enclosing sphere intersects the current scan ray through that pixel. This list represents a significant culling of the totality of atoms to just those which are relevant to the

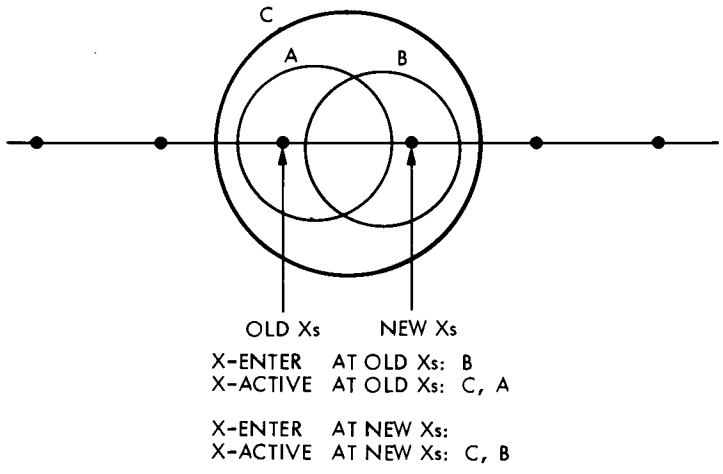


Fig. 11. x-enter/x-active lists.

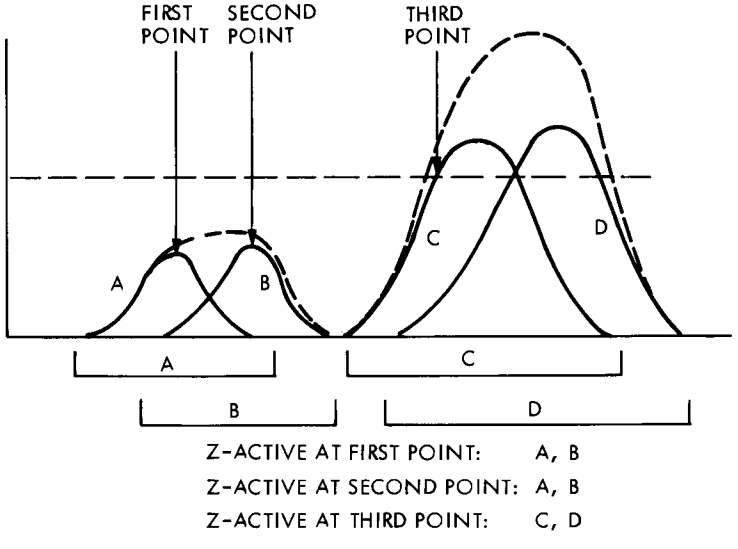


Fig. 12. Entering/exiting from z-active list.

current pixel. There is one more level of culling available, however. This is in the z direction. In Section 3.3 we described the technique for finding an initial point for the iterative solution as a scan from front to back through a z-sorted list of potential solution points. If, prior to this $z$scan, we calculate the z_{min}/z_{max} values of the intersections of the enclosing spheres with the scan ray, we can maintain a z-active list as the $z$scan progresses. In this case, the z values examined for entering/exiting tests are not equally spaced integer values as in the x and y cases. They are instead taken one at a time from the potential z list. This does not alter the basic principle, however. See Figure 12.

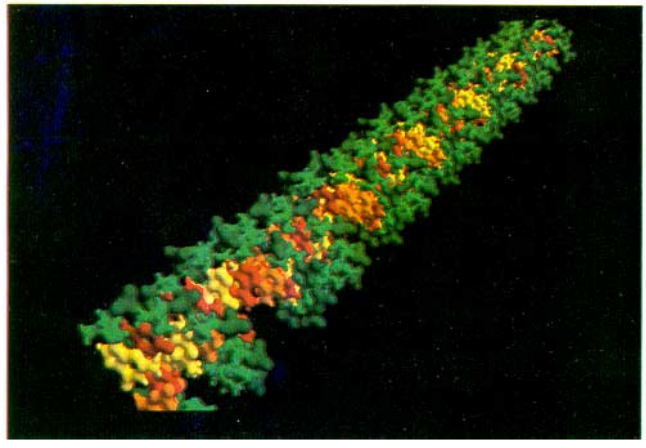


Fig. 13. Image containing 4000 atoms.

When we perform the iteration loop then the summation of the D_i will be taken over only those atoms close enough to the initial guess to matter. For example, for the image containing over 4000 atoms (Figure 13), there were at most 6 atoms on the z-active list for the iteration and usually much less.

3.6 Calculating the Range in x, y, and z

In this section we explicitly indicate how to calculate the x, y, and z extents of the enclosing sphere of an atom. Such enclosing spheres are defined, as described in Section 3.5, by the equation

$$tT = D_i = \exp(-a_i r_i^2 - B_i).$$

Taking the logarithm (and recalling that $T = 1$), we transform this into the equation for a quadric surface

$$0 = a_i r_i^2 + (B_i + \ln t).$$

To calculate $z_{\min}/z_{\max}$ for the range test of the zscan, we substitute the expression for r_i^2 in terms of z:

$$0 = a_i[(\bar{R} \cdot \bar{R})(z - z_{mi})^2 + M_i] + (B_i + \ln t)$$

and solve for z:

$$z_{\min} = z_{mi} - D_z$$

$$z_{\max} = z_{mi} + D_z$$

where

$$D_z = \sqrt{\frac{a_i M_i + B_i + \ln t}{-a_i(\bar{R} \cdot \bar{R})}}.$$

There will be two roots to this as long as the expression under the radical is positive. The curve defined by setting this equal to zero will then define the

projection of the silhouette outlines of the enclosing sphere onto the screen:

$$a_i M_i + B_i + \ln t = 0$$

Substituting the definition of M_i and z_{mi} we can transform this into

$$\Delta(\bar{R}\cdot\bar{R}) - (\bar{R}\cdot\bar{P}_i)^2 = 0 \qquad \text{where} \quad \Delta = (\bar{P}_i\cdot\bar{P}_i) + \frac{B_i + \ln t}{a_i}.$$

Recalling that $R = (x_z, y_z, 1)$ we get

$$\Delta(x_z^2 + y_z^2 + 1) - (x_z x_i + y_z y_i + z_i)^2 = 0.$$

Now for a particular scan line, y_z is constant. We then have a quadratic equation in x_z whose solutions give the range in x_z for that scan line.

$$x_z^2(\Delta - x_i^2) + x_z(-2x_i(y_z y_i + z_i)) + (\Delta(y_z^2 + 1) - (y_z y_i + z_i)^2) = 0$$

The solutions to this equation ($x_{z\min}$, $x_{z\max}$) must still be converted to pixel coordinates ($x_{s\min}$, $x_{s\max}$) by reference to eq. (*).

The above equation will have two roots at all values of y_z for which its discriminant is positive. The values of y_z for which this discriminant becomes zero therefore yield the maximum and minimum y_z for which the enclosing sphere is visible.

$$4x_i^2(y_z y_i + z_i)^2 - 4(\Delta - x_i^2)(\Delta(y_z^2 + 1) - (y_z y_i + z_i)^2) = 0$$

Again, the two solutions for y_z from this equation must be converted to pixel coordinates via eq. (*).

In both the x and y cases, the pixel range for the enclosing sphere must be intersected with the pixel range of the display. This effectively means that we are performing clipping in screen space. If the range of the sphere is completely outside the screen then it can be eliminated entirely.

4. TIMING

The rendering algorithm, while making some quite interesting shapes, is not terrifically fast. In an effort to see where the time is spent within the algorithm it was instrumented for timing measurements. Table 1 shows the results for the calculations involved in generating Figure 14, containing 64 atoms. Again, the halo around the molecule represents the enclosing spheres of the atoms and indicates the pixel range over which calculations are made.

Note that even though the *y*scan and *x*scan routines take quite a while (since they have to deal with larger active lists), their net contribution to the running time is small since they are called only once per image and once per potentially occupied scan line, respectively. The *z*calc routine is where the values of z_{mi}, M_i, and so forth, are calculated for all the atoms on the z-active list. This is what takes most of the time. Part of the reason for this is not so much the amount of time spent in the routine as the very large number of times it is called. It is called once for each pixel covered by any enclosing sphere, while the iteration and shading routine are called once per actually occupied pixel.

We also present histograms of the sizes of the active lists for each of the nested scans. Each bin of the histogram counts the number of times a given size of active

Table I. Timing for the calculations involved in generating Figure 14.

Routine	Number of calls	Total time (seconds)	Time/call (msec)	% of total time
*x*scan	1	0.01	99.34	0.0
*y*scan	163	9.57	58.69	6.5
*z*calc	31036	58.57	1.89	39.5
*z*scan	17926	22.63	1.26	15.3
Iteration	16762	25.20	1.50	17.0
Shading	16762	32.28	1.92	21.8
Total	—	148.25	—	—

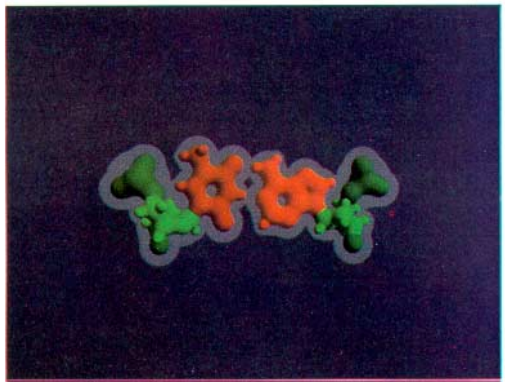


Fig. 14. Sample image for timing test.

list is passed to the routine. Note that, due to the culling process, the iteration and shading routines were usually called with active lists of length 3 or less. See Figure 15 (pages 252–253).

5. HIERARCHICAL MODELING

The implementation of this algorithm was done on a PDP-11, which allows limited address space for user programs. There is not enough user memory for a program which does both the modeling and rendering for systems of atoms of the size desired. Accordingly the process is divided into two programs which communicate via a temporary file. This file is, in fact, just the *y*-enter list sorted on y_{max} for the enclosing volume of each atom. A general purpose modeling program accepts commands controlling the positioning and blobbiness parameters of atoms, global lighting, viewing parameters, and so forth. It then writes out the sorted *y*-enter list into a temporary file. The rendering program then reads in this file, atom by atom, as the scan proceeds down the screen. This frees the rendering program from any restrictions on the total number of atoms visible. It needs only to read one atom ahead of itself in the *y*-enter list file to be able to tell when the next atom is to enter the *y*-active list. Internally it maintains only the *y*-active list and thus has restrictions only on the maximum number of atoms visible on any one scan line. (This technique has also been used by the author in polygon-

rendering programs and bicubic patch-rendering programs, the lists in those cases being of polygons or patches.)

There are still some limits on the maximum number of atoms which the modeling program can maintain. These can be relaxed by inserting a separate sorting module between the modeler and the renderer. The modeler then does not need to be able to store all the atoms in a scene. It just concerns itself with command line interpretation, setting parameters, and transforming atoms to viewing space. It writes out the atom list as it reads it from a molecule definition file. The sorter then is a small program with a large array. It reads the y-enter list file, sorts the data, and writes it out again.

For the purposes of modeling large polymerized molecules (like DNA) an alternative hierarchical modeling scheme was employed. Polymers are made up of a large collection of a few basic modules. For DNA these modules are the four nucleotides and a few free radicals used to simulate the replication process. Each module is modeled as a rigid body at an arbitrary position and orientation. The definition of a module lists its constituent atoms and the radius of an enclosing sphere for the entire module. When an image is to be made, the modules are first sorted in an order based on the minimum y_s of their enclosing spheres. Then a scan is made in the y direction to generate the entire y-enter list of atoms directly in sorted order. For each new y_s value the y-module list is examined to see if any modules have become active. When a module becomes active it is expanded into its constituent atoms which are then transformed into viewing space according to the module's orientation and position. These atoms are added to a candidate y-enter pool. When all newly activated modules have been expanded, the candidate y-enter pool is examined. Any atoms which have actually become visible on the scan line are written to the y-enter file. The advantage of this process is that the candidate y-enter pool never gets very large and can therefore handle large structures.

6. EXTENSIONS

The initial algorithm was devised for a special purpose task. We examine here some simple generalizations of the process to give a wider range of shapes which can be modeled.

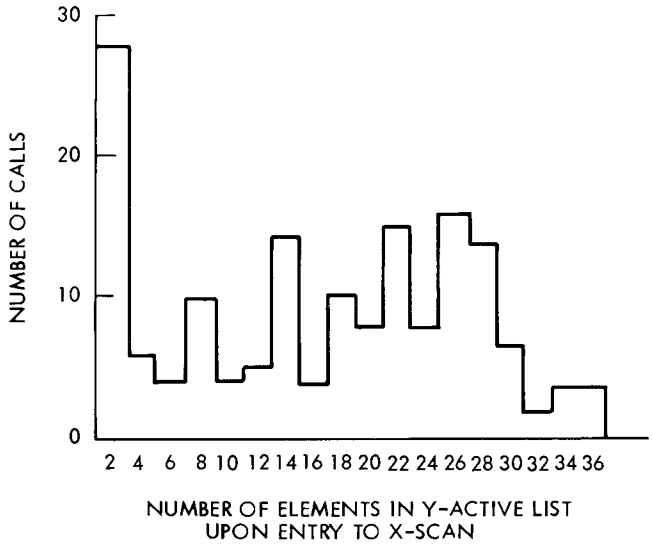
6.1 General Quadric Seeds

One obvious extension to the shapes defined so far is to provide for nonspherical primitive shapes. Recall the original defining equation consisted of terms

$$\exp(-a_i r_i^2 - B_i).$$

The exponent of e is just a special case quadric:

$$\begin{aligned} -a_i r_i^2 - B_i &= -a_i((x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2) - B_i \\ &= (xyz1)(-a_i)\begin{bmatrix} 1 & 0 & 0 & -x_i \\ 0 & 1 & 0 & -y_i \\ 0 & 0 & 1 & -z_i \\ -x_i & -y_i & -z_i & P \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \end{aligned}$$



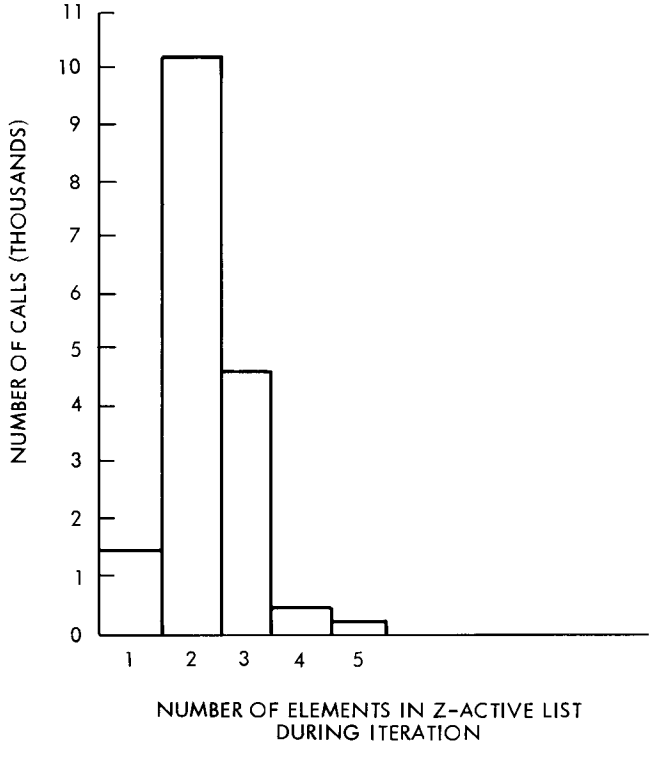
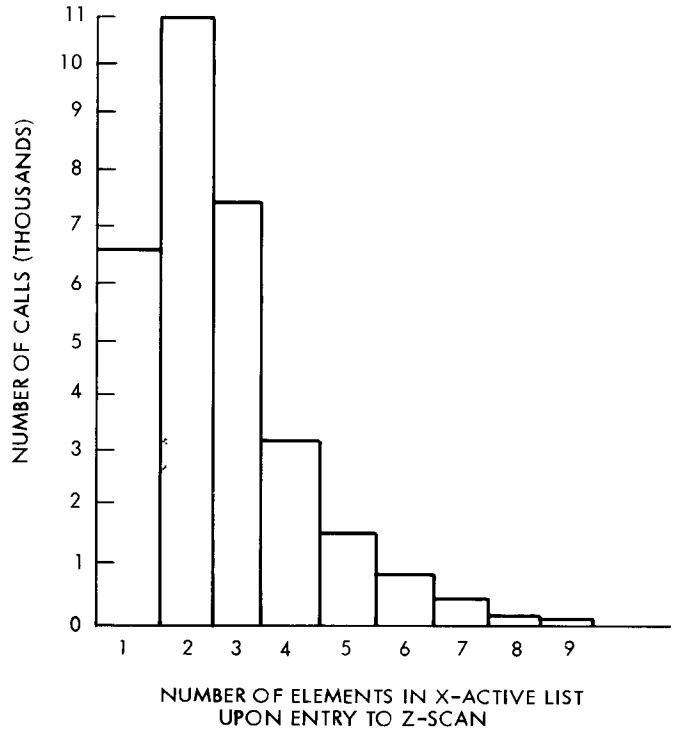
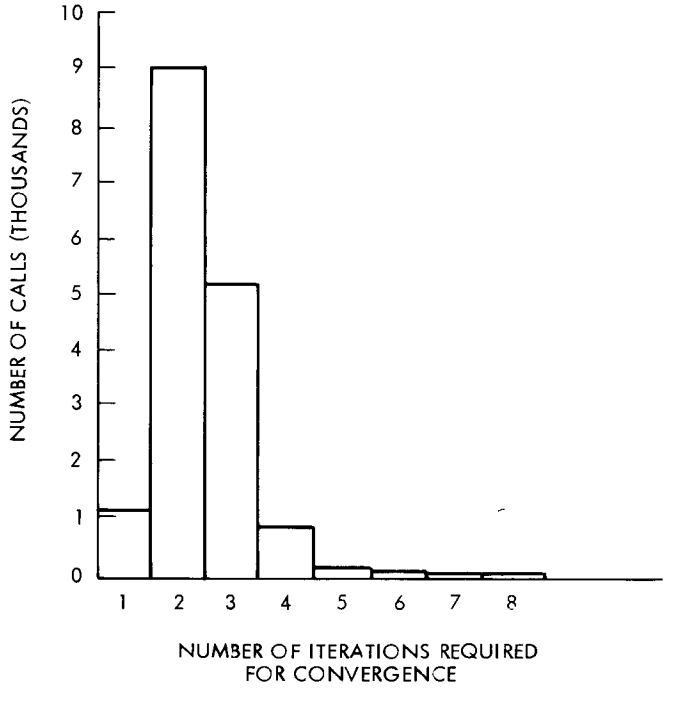


Fig. 15. Histograms of lengths of active lists.

Figure 15 continued on next page.


NUMBER OF CALLS (THOUSANDS)
0
1
2
3
4
5
6
7
8
9
10
11
1
2
3
4
5
6
7
8
9
NUMBER OF ELEMENTS IN X-ACTIVE LIST
UPON ENTRY TO Z-SCAN


NUMBER OF CALLS (THOUSANDS)
0
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
NUMBER OF ITERATIONS REQUIRED
FOR CONVERGENCE

where

$$P = x_i^2 + y_i^2 + z_i^2 + \frac{B_i}{a_i}.$$

By allowing general quadrics here, we can have ellipsoids, cylinders, planes, and so forth, as modeling primitives. This is done by beginning with a canonical unit sphere at the origin defined by

$$0 = (xyz1)\begin{bmatrix} -B_i & 0 & 0 & 0 \\ 0 & -B_i & 0 & 0 \\ 0 & 0 & -B_i & 0 \\ 0 & 0 & 0 & B_i \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

This is then scaled, rotated, and translated to the desired location by the standard transformation techniques for quadrics [1], that is, multiply on the left by the inverse of the transformation matrix and on the right by the transpose of the inverse.

$$Q' = T^{-1}QT^{-1t} \qquad \text{where } T \text{ is transformation such that}$$

$$(x'y'z'w') = (xyzw)T$$

This expression is chosen to preserve the relationship:

$$\text{if} \quad (xyzw)Q(xyzw)^t = 0$$

$$\text{then} \quad (x'y'z'w')Q'(x'y'z'w')^t = 0$$

The reader can verify that by scaling the matrix by R_i and translating it to (x_i, y_i, z_i) one obtains just the special case formulation we used above for pure spheres. The generalization of most of the other equations in the preceding discussion can be performed by making the following replacements:

$a_i(\bar{R}\cdot\bar{R})$	becomes	$\bar{R}Q'\bar{R}^t$
$a_i(\bar{P}_i\cdot\bar{R})$	becomes	$\bar{R}Q'\bar{W}^t$
$a_i(\bar{P}_i\cdot\bar{P}_i) + B_i$	becomes	$\bar{W}Q'\bar{W}^t$

where

Q' = transformed (4×4) quadric matrix
$\bar{W} = (0 \quad 0 \quad 0 \quad 1)$
$\bar{R} = (x_z \quad y_z \quad 1 \quad 0)$

A sample picture appears in Figure 16.

For shapes of infinite extent, such as cylinders, the calculation of max/min values in x_s or y_s may yield an infinite range. This occurs when there are no solutions to the range determining quadratic equations of Section 3.6. To handle these situations properly we must recall that we are solving these equations not so much to find their zeros as to find the region where the quadratic polynomial is positive (since its square root is required later). In the general case this may yield one finite span (for ellipsoids), one infinite span (the axis of a cylinder) or two semiinfinite spans (hyperboloids). With proper care in examining the polynomial these cases can be readily distinguished.

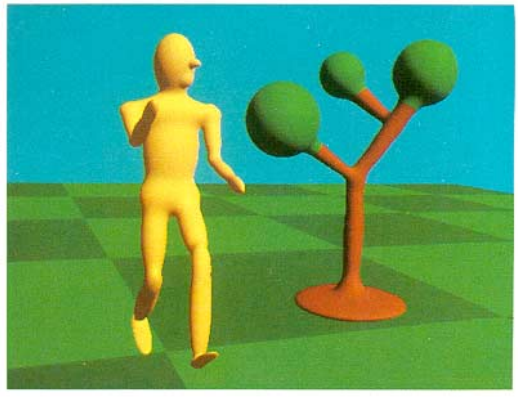


Fig. 16. Image generated using general quadric exponents.

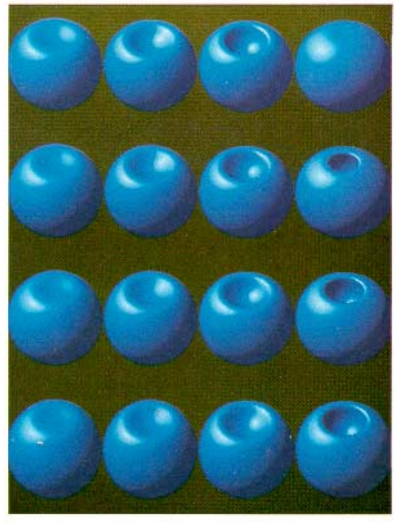


Fig. 17. Effects of negative volumes.

6.2 Negative Volumes

Another extension is to allow negative values for b_i. This effectively gives negative volumes. They are not visible themselves, but when placed near normal objects they make dents. This is because their density contributions are being subtracted from the summation. A sample image appears in Figure 17.

In fact, with some adjustments to the algorithm it should be possible to allow complex values for the b_i. This would prove useful for molecular simulations since quantum wave functions are actually complex valued functions. We could therefore represent antibonding orbitals.

6.3 Hyperellipsoids

A more general surface shape can be provided by allowing exponents other than 2 on the terms of the exponent.

6.4 Other Bump Functions

The final extension considered here involves alterations to the exponential function. We can use any shape that has the same general form. In fact, the implementation of the algorithm here uses a table lookup procedure with interpolation for rapid evaluation of the exponential function. By placing different values in the table we can easily change the shape of the bump function. Care should be taken not to defeat the heuristics for the selection of initial guesses for the numerical solution. Basically the function must equal 1 at $f(0)$ and increase monotonically over the range used.

7. CONCLUSIONS

We have presented an algorithm which simultaneously models and renders a class of surfaces which have an interesting visual appearance and should prove useful for a variety of applications. Some simple extensions to this process show promise of generating other interesting shapes.

All raster scan image synthesis algorithms must address the problem of anti-aliasing (e.g., area sampling). Algorithms based on algebraic surfaces are fairly entrenched in point sampling. This creates problems mostly at silhouette edges since the main body of the surface shape does not have any high frequencies to alias. No explicit anti-aliasing has yet been attempted on the images presented here. This would be a fruitful topic for further research.

REFERENCES

1. BLINN, J.F. Geometric representations in computer graphics. Workshop on representation of three-dimensional objects, Univ. of Pennsylvania, Philadelphia, Pennsylvania, May 1979.
2. CATMULL, E.E. Computer display of curved surfaces. In *Proc. of the Conf. on Computer Graphics, Pattern Recognition, and Data Structure* (Los Angeles, Calif., May 14–16), 1975, IEEE, New York, pp. 11–17.
3. CLARK, J.H. A fast scan-line algorithm for rendering parametric surfaces. Unpublished.
4. DUFF, T. The soid and roid manual. New York Institute of Technology, Sept. 1980.
5. GOLDSTEIN, E. AND NAGLE, R. 3D visual simulation. *Simulation 16*, 1 (Jan. 1971), 25–31.
6. KNOWLTON, K. AND CHERRY, L. Atoms, a 3D opaque molecule system. *Comput. Chem. 1*, 3 (1977), 161–166.
7. LANE, J.M., CARPENTER, L.C., WHITTED, T., AND BLINN, J.F. Scan line methods for displaying parametrically defined surfaces. *Commun. ACM 23*, 1 (Jan. 1980), 23–34.
8. MAX, N.L. Atom LLL:—Atoms with shading and highlights. *Computer Gr. 13*, 2 (Aug. 1979), 165–173.
9. PORTER, T. Spherical shading. *Computer Gr. 12*, 3 (Aug. 1978), 282–285.
10. ROTH, S. Ray casting as a method for solid modeling. *Comput. Gr. Image Process 18*, 2 (Feb. 1982), 109–144.

Received March 1982; revised May 1982; accepted May 1982.