# THE MATHEMATICS OF QUADRIC SURFACE RENDERING AND *SOID*

*Paul S. Heckbert*

Three Dimensional Animation Systems Group
Computer Graphics Laboratory
New York Institute of Technology

3-D Technical Memo 4
13 July 1984

## 1. INTRODUCTION

The *SOID* program, which renders quadric surface models, has been one of NYIT's principal rendering programs since it was written in 1979. Over the years it has spawned several variations, most important of which is *SSOID*, short for *simultaneous soid*, which solves a quadric equation and several quadric inequalities. The only paper about *SOID* was written in January 1980 and never finished [Duff80a]. This report is an attempt to collect and distill the notes of Tom Duff, the principal author of *SOID*, and those who have modified it over the years: Thad Beier, Mike Chou, and Paul Heckbert. To avoid obsolescence, this document will stress the mathematical techniques used by *SOID*, and possible improvements, rather than implementation details of the current version. A sprinkling of mathematical tidbits has been added for the reader's edification. An extensive bibliography on quadrics is included at the end of the paper.

## 2. HISTORY OF QUADRICS

Quadric surfaces are not an invention of computer graphics; they have been studied by mathematicians for hundreds of years and whole books have been written about them [Dresden30], [Maxwell51]. In mathematics quadrics are thought of as second degree polynomials in three variables, while in computer graphics geometric interpretations are dominant. The first computer graphic pictures of quadrics were line drawings. While silhouette curves and hidden line elimination are reasonably straightforward for quadrics, the computation of quadric-quadric intersection curves is quite complex, leading to 4th degree equations. Several methods have been tried [Dresden30], [Maxwell51], [Weiss66], [Woon71], [Levin76], [Sarraga83].

The earliest use of quadric surfaces in raster graphics was in MAGI's *Synthavision* system, as one of a number of solid modeling primitives [Davis68], [Goldstein71], [Brooks74]. Shortly thereafter, Mahl, at the University of Utah, developed a program which rendered (in scanline order) quadric patches bounded by quadric inequalities [Mahl70]. Utah researchers promoted the use of 4x4 matrices to represent both transformations and quadrics [Newell75], [Blinn78b]. Much of the notation used in

this paper comes from [Blinn80].

Ellipsoids have been used for primitive human body models [Herbison-Evans82], [Badler79]. Herbison-Evans has proposed that real-time hidden line elimination of such models (without intersection curves) is within reach. Spheres are extremely popular in molecular graphics [Knowlton77], [Max83b]. Several groups have developed hardware/firmware systems to display shaded pictures of molecules in real time [Staudhammer78], [Pique83], [Max83b].

Besides our own *MAT* [Duff80b], several other quadric modeling languages have been developed [Mahl70], [Levin80a], [Levin80b].

## 3. DEFINITIONS

*algebraic surface:* a polynomial in several variables: $F(x,y,z,...)=0$

*parametric surface:* a function defined in terms of parameters,
e.g. $x=\cos\theta$, $y=\sin\theta$, $\theta$ being the parameter

*quadratic:* a second-degree polynomial (*e.g.* $ax^2+bx+c$)

*quartic:* a fourth-degree polynomial

*conic curve:* contour of a second-degree polynomial of two variables (ellipse, parabola, hyperbola, degenerates)
(*e.g.* $ax^2+bxy+cy^2+dx+ey+f=0$)

*quadric surface:* contour of a second-degree polynomial of three variables (ellipsoid, paraboloid, cylinder, cone, hyperboloid of one or two sheets, hyperbolic paraboloid, degenerates)

*soid:* the subset of quadric surfaces which *SOID* renders (ellips*oid*, paraboloid, cylinder, cone, hyperboloid of one sheet)

*cone:* In *SOID*, a cone is a hyperboloid of one sheet, the surface of rotation about the y axis of a line segment. In this document, we keep the usual definition.

For notation, we will use lower case for scalar variables, upper case for vectors and matrices. We use the homogeneous coordinate notation, where the vector

$$P = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}, \quad \text{sometimes written} \quad \{x,y,z,w\}$$

represents the 3-D point $\{x/w, y/w, z/w\}$. Readers unfamiliar with the homogeneous representation should see [Blinn80]. Points are represented by column vectors, and planes by row vectors.

Screen space is frame buffer coordinates, and object space (sometimes called unit radius space) is the coordinate system in which quadrics have their canonical form (see section on classification below). Object space is left-handed, like *SOID*'s: $x$ points to the right, $y$ up, and $z$ away.

The *adjoint* (or *adjugate*) of a matrix is the transpose of the matrix of cofactors, and it is equal to the inverse times the determinant. Since we will be working with homogeneous coordinates, where the scale of a matrix is arbitrary, we can use the adjoint in place of the inverse. The adjoint is preferable because it is easier to calculate and it always exists, even if $Q$ is singular. Adjoints will be notated with an asterisk, e.g. $Q^*$.

## 4. QUADRICS

Quadrics are second-degree algebraic surfaces in three dimensions, that is, they are described by polynomials of degree two in the variables $x$, $y$, $z$, and $w$:

$$ax^2 + 2bxy + 2cxz + 2dxw + ey^2 + 2fyz + 2gyw + hz^2 + 2izw + jw^2 = 0$$

This is called a homogeneous quadratic form (homogeneous in the sense that each term is of the same degree). Quadrics may also be written using 4x4 matrices:

$$\begin{pmatrix} x & y & z & w \end{pmatrix} \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = 0$$

We switch from coefficients $a-j$ to subscripted array elements $q_{ij}$ for consistency with *SOID*:

$$P^T Q P = \begin{pmatrix} x & y & z & w \end{pmatrix} \begin{pmatrix} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ q_{20} & q_{21} & q_{22} & q_{23} \\ q_{30} & q_{31} & q_{32} & q_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{aligned} & q_{00}x^2 + 2q_{01}xy + 2q_{02}xz + 2q_{03}xw \\ & + q_{11}y^2 + 2q_{12}yz + 2q_{13}yw \\ & + q_{22}z^2 + 2q_{23}zw \\ & + q_{33}w^2 \end{aligned} = 0$$

The 4x4 matrix Q is called the quadric matrix. Note that it is assumed to be symmetric ($q_{ij}=q_{ji}$). We will see later that the symmetry of this matrix is preserved by homogeneous transformations. Generally we will arrange the coefficients so that $P^T Q P$ is positive outside the surface, and negative inside. The meaning of inside and outside is clear for some quadrics, such as ellipsoids and cylinders, but unclear for others, such as hyperbolic paraboloids.

## 5. CLASSES OF QUADRICS

A quadric is called *singular* when its quadric matrix $Q$ has a determinant of 0. The five non-singular quadrics are the ellipsoid, elliptic paraboloid, hyperbolic paraboloid, hyperboloid of one sheet, and hyperboloid of two sheets. The singular quadrics include cones, swept ellipses (cylinders), swept hyperbolas and parabolas and the degenerates: planes, lines, and points.

Although *SOID* is always told the type of each quadric by *MAT*, the program which feeds it, most of *SOID* works without that information. In some cases one might know $Q$ without knowing the quadric class, for instance, when fitting a quadric through a number of surface points. In this case, the program would have to recognize classes in order to parameterize and texture map the surface. Techniques for

classification can be found in several places [Dresden30], [Levin76], [Levin80a], [Blinn80].

Briefly, to classify a quadric one looks at: the rank of the matrix $Q$, its determinant (also known as the *discriminant*), and the signs of its four eigenvalues. Since $Q$ is symmetric, the eigenvalues are guaranteed to be real. The eigenvalues determine the dimensions of the quadric along its principal axes, and the eigenvectors give the directions of those principal axes. The eigenvalues, and hence the shape of a quadric, are invariant with translation and rotation.

This shape invariance implies that any quadric can be transformed to center on {0,0,0}, and its principal axes aligned with $x$, $y$, and $z$ putting the quadric in *canonical* (orthogonal) form:

$$ax'^2+by'^2+cz'^2+dw'^2 = 0$$

where $x'$, $y'$, $z'$ and $w'$ are linear functions of $x$, $y$, $z$, and $w$.

Whereas the general quadric equation has 10 coefficients, in this canonical form there are 4. This gives 3 degrees of freedom, since $d'$ can be arbitrarily fixed. These 3 degrees of freedom can be eliminated by scaling $x$, $y$, and $z$ differentially. The most important canonical quadrics for our purposes are (see fig. 1):

| | *UN-NORMALIZED* | | *NORMALIZED* |
|---|---|---|---|
| *ellipsoid* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = 1-\frac{y^2}{b^2}$ | *sphere* | $x^2+z^2 = 1-y^2$ |
| *elliptic paraboloid* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = 2y$ | *circular paraboloid* | $x^2+z^2 = 2y$ |
| *hyperbolic paraboloid* | $\frac{x^2}{a^2}-\frac{z^2}{c^2} = 2y$ | *hyperbolic paraboloid* | $x^2-z^2 = 2y$ |
| *hyperboloid of one sheet* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = \frac{y^2}{b^2}+1$ | *circular hyperboloid of one sheet* | $x^2+z^2 = y^2+1$ |
| *hyperboloid of two sheets* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = \frac{y^2}{b^2}-1$ | *circular hyperboloid of two sheets* | $x^2+z^2 = y^2-1$ |
| *cone* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = \frac{y^2}{b^2}$ | *circular cone* | $x^2+z^2 = y^2$ |
| *elliptic cylinder* | $\frac{x^2}{a^2}+\frac{z^2}{c^2} = 1$ | *circular cylinder* | $x^2+z^2 = 1$ |

*SOID*'s set of canonical quadrics is somewhat different:

**Sphere**

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad x^2+y^2+z^2 = 1$$

**Paraboloid**

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \end{pmatrix}, \quad x^2+z^2 = y$$

**Cone**

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad x^2+z^2 = y^2$$

**Cylinder**

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \quad x^2+z^2 = 1$$

**Hyperboloid of One Sheet** (general soid)

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & a_h & 0 & b_h \\ 0 & 0 & 1 & 0 \\ 0 & b_h & 0 & c_h \end{pmatrix}, \quad x^2+a_h y^2+2b_h y+z^2+c_h = 0$$

This formula, which *SOID*'s *cone* command uses, is general enough to include all of the previous classes:

$$\begin{aligned}
\text{Sphere:} \quad & a_h = 1 \quad b_h = 0 \quad c_h = -1 \\
\text{Paraboloid:} \quad & a_h = 0 \quad b_h = -1/2 \quad c_h = 0 \\
\text{Cone:} \quad & a_h = 1 \quad b_h = -1 \quad c_h = 0 \\
\text{Cylinder:} \quad & a_h = 0 \quad b_h = 0 \quad c_h = -1
\end{aligned}$$

## 6. TRANSFORMING QUADRICS

The 4x4 quadric matrix notation above is convenient because it's consistent with standard 3-D homogeneous coordinates. We first review transformation of points and planes:

$$P' = MP = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x_o \\ y_o \\ z_o \\ 1 \end{pmatrix} \qquad \begin{aligned} x &= \frac{x'}{w'} \\ y &= \frac{y'}{w'} \\ z &= \frac{z'}{w'} \end{aligned}$$

which maps an object space point $\{x_o, y_o, z_o\}$ through a perspective transformation to the screen space point $\{x, y, z\}$, where $M$ is the object space to screen space transformation matrix.

To represent the plane $NP = ax + by + cz + d = 0$, we use a row vector: $N = \begin{pmatrix} a & b & c & d \end{pmatrix}$. To transform a plane by a matrix $M$, we compute $N' = NM^*$ or $N'^T = M^{*T} N^T$.

Not only are lines and planes preserved by the perspective transformation, but also quadric surfaces [Newell75]. This can be seen with the following identity:

$$P^T Q P = (P^T M^T)(M^{T^*} Q M^*)(MP) = P'^T Q' P'$$

$$\textit{where:} \quad Q' = M^{T^*} Q M^* \tag{1}$$

Note that equation (1) above preserves the symmetry of the quadric matrix. Thus it is possible to take a quadric surface ($Q$) which is defined in object space and find its equation in screen space ($Q'$), *and it will still be a quadric.* This is the mathematical magic behind *SOID*. Naturally, the class of quadric may change between $Q$ and $Q'$ cylinders in perspective appear as cones, and spheres can become ellipsoids or hyperboloids of two sheets [Blinn80].

## 7. INTERSECTION OF LINE AND QUADRIC

The simultaneous solution of a quadric $P^T Q P = 0$ and a line $P = A + \alpha D$ yields a quadratic in the line parameter $\alpha$, called *Joachimstal's equation* [Maxwell51]:

$$\begin{aligned}
P^T Q P &= (A^T + \alpha D^T) Q (A + \alpha D) \\
&= (D^T Q D)\alpha^2 + (2 D^T Q A)\alpha + (A^T Q A) = 0
\end{aligned}$$

$$\textit{where: } A = \{x_a, y_a, z_a, 1\} \quad \textit{and } D = \{x_d, y_d, z_d, 0\}$$

This can be solved for $\alpha$ and the intersection points found using the standard formula for roots of a quadratic. The above method is one approach to ray tracing quadrics. The alternative is to transform the line into the coordinate system which canonicalizes the quadric to have unit radius and solve for the intersection using formulas specific to each quadric class [Roth82].

For one-level ray tracing (no reflection or transparency), which is essentially what *SOID* and MAGI's *Synthavision* do, screen space rays are always parallel to the z axis, so the formulas above simplify significantly. If $Q$ is the screen space quadric matrix, and $D=\{0,0,1,0\}$, then

$$az^2+bz+c = 0 \qquad \textit{where: } \begin{cases} a = q_{22} \\ b = 2(q_{02}x+q_{12}y+q_{23}) \\ c = q_{00}x^2+2q_{01}xy+2q_{03}x+q_{11}y^2+2q_{13}y+q_{33} \end{cases} \tag{2}$$

Whose solutions are:

$$z = \frac{-b \pm \sqrt{\xi}}{2a} \qquad \textit{where: } \xi = b^2-4ac = \textit{discriminant}$$

Note that solving for $z$ does not involve an explicit division by $w$, as would a point transformation; the perspective is implicit in the quadric transformation formula (1). If $a=0$, the quadratic is degenerate, and *SOID* will reject it. This can happen, for example, when the eye is on the surface of the quadric, looking in a direction parallel to the surface.

## 8. NORMALS AND TANGENT PLANES

The plane tangent to the surface of the quadric can be found by taking the gradient of the quadratic form:

$$N^T = \frac{\nabla}{2}(P^T Q P) = \begin{pmatrix} q_{00}x+q_{01}y+q_{02}z+q_{03} \\ q_{10}x+q_{11}y+q_{12}z+q_{13} \\ q_{20}x+q_{21}y+q_{22}z+q_{23} \\ q_{30}x+q_{31}y+q_{32}z+q_{33} \end{pmatrix} = QP$$

$$\textit{where: } \nabla \equiv \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}, \frac{\partial}{\partial w} \right)$$

and $P$ is a point on the surface.

The first three components of $N$ define the normal vector. Thus, to find the normal to a quadric, one simply multiplies the quadric matrix by a point on the surface. If the coefficient signs of $Q$ were chosen properly, the normal will point outward.

Regardless of whether $P$ is on the surface, $N$ is called the *polar plane* of $P$, and $P$ the *pole* of the plane $N$. If $P$ is outside, the plane $N$ will intersect the quadric, but if $P$ is inside, $N$ will not intersect. To summarize,

the polar plane of a point $P$ is $N = P^T Q$

the pole of a plane $N$ is $P = Q^* N^T$

Substituting the expression for $P$ into the quadric equation,

$$P^T QP = NQ^{*T} QQ^* N^T$$

$$= NQ^* N^T = 0 \qquad (3)$$

This gives the set of planes tangent to a quadric. If we replace $Q^* N^T$ with $P$ in the equation above, we get $NP=0$, which shows that a point lies in its polar plane if it is on the surface of the quadric. Since $P^T QP$ is positive for exterior points and negative for interior ones, $NQ^* N^T$ will be positive for planes which intersect the quadric and negative for ones which don't, by the properties of poles and polars.

By substituting $N=(-1 \;\; 0 \;\; 0 \;\; x)$ into equation (3), one can solve for the $x$ extrema [Blinn80]. This provides a way to find the bounding rectangle of a quadric in screen space.

## 9. SILHOUETTE OF A QUADRIC

The silhouette of a surface is the space curve along which the z component of the normal is zero. Thus for quadrics, $q_{20}x+q_{21}y+q_{22}z+q_{23}=0$, so the silhouette curve always lies in a plane. The intersection of a plane and a quadric is a conic section whose 3x3 matrix can be found quite simply [Blinn80].

In a rendering program one would often like to find the two silhouette points on a given scan line ($x_{\min}$ and $x_{\max}$ for a given $y$). If we take equation (2) as our quadric and intersect it with a plane $y=constant$, we get a formula for $z$ as a quadratic function of $x$. The $x$ extrema occur where the quadratic has one root, *i.e.,* where $\xi=0$. $\xi$ is a quadratic function of $x$. Setting the discriminant of the $\xi$ quadratic to 0 gives us a quadratic in $y$ whose roots are the $y$ extrema of the quadric. Note that these quadratics often degenerate into linear equations, so one must be careful [Blinn82].

## 10. SOID and SSOID

*SOID* draws quadrics one at a time, using a z-buffer algorithm [Newman79] for hidden surface removal. The order of its loops is thus:

(1) step through all quadrics
(2) step through quadric's y range (do scan line)
(3) step through quadric's x range (do pixel)

To scan out a quadric, *SOID* fits a box around the object space primitive and projects it to screen space. A rectangle is fit around the eight projected vertices, and all pixels in the rectangle are checked for surface hits using the simple line-quadric intersection test above, ray tracing one object at a time. Equation (2) is solved at each pixel, yielding two, one, or no roots depending on the sign of the discriminant, $\xi$. If the discriminant is negative, the quadric does not contribute to that pixel, and the program steps to the next pixel. If non-negative, an intersection has been found, and it is checked for validity.

In *SOID*, infinite quadrics (all except ellipsoids) are clipped above and below by two bounding planes defined in object space, so that valid intersections are restricted to $y_1 \le y_o \le y_2$. In *SSOID*, the surface is additionally bounded by other quadrics. The user specifies which side of the bounding quadric surface to keep (inside or outside) at modeling time, thereby defining an inequality. Both roots of equation (2) are tested for validity, nearest first, by substitution into the formulas for the bounding planes and quadrics. In the case of a plane, the sign of $NP$ indicates one side of the plane or the other, while for quadrics, $P^T QP$ is negative on the inside and positive on the outside. If the point satisfies all of the bounding inequalities it is valid. When a valid root is found, the normal is computed, and the shading calculated based on the normal vector, direction of the eye, and direction of the lights. If neither root is valid, the ray has passed through a hole.

To speed *SOID*'s inner loop, the calculation of the coefficients a, b, and c can be done incrementally using the method of finite differences. Below we sketch out the inner loop of *SOID*'s ''ball.c''.

```
a = q22
Δb = 2q02                                   ←b(x+1)−b(x)
ΔΔc = 2q00                                  ←Δc(x+1)−Δc(x)
for (y=ymin; y≤ymax; y++) {
    [initialize b and c at x=xmin]
    Δc = q00*(2*xmin+1)+2*(q01*y+q03)       ←c(x+1)−c(x)
    tx = w00*xmin+w01*y+w03
    ty = w10*xmin+w11*y+w13
    tz = w20*xmin+w21*y+w23
    for (x=xmin; x≤xmax; x++) {
        ξ = b*b-4a*c
        if (ξ≥0) {
            z = (-b±sqrt(ξ))/2a             ←might have to check both roots
            [read z-buffer and check bounding planes/quadrics]
            {nx,ny,nz} = {tx,ty,tz}+{w02,w12,w22}*z
            {nx,ny,nz} /= sqrt(nx*nx+ny*ny+nz*nz) ←unit normal
            [compute texture and shade for pixel]
            [write image and z-buffer]
        }
        b += Δb
        c += Δc
        Δc += ΔΔc
        {tx,ty,tz} += {w00,w10,w20}
    }
}
```

The matrix $W$ transforms screen space points into world space planes. Using $\{tx, ty, tz\}$, the world space normal can be computed from the screen coordinates using only 3 multiplies and 6 adds per pixel [Duff80a]. World space, which is intermediate to object and screen spaces, is the coordinate system in which shading is done. $W$ is

found using the plane transformation and tangent plane formulas:

$$W = M^{*T}_{screen \rightarrow world}\ Q_{screen}\ , \qquad \begin{pmatrix} nx \\ ny \\ nz \\ nw \end{pmatrix} = W \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The current version (v2.2) of *SOID*, running on a VAX 780 with floating point accelerator, takes $70\mu s$ per pixel to calculate and test the discriminant, and $900\mu s$ to do the square roots and cheap shading without texture mapping. Since testing a pixel is so much faster than rendering one, extremely tight bounds would not speed *SOID* much. The **sqrt** subroutine in *libnm* takes about $100\mu s$, so square roots take up a large fraction of *SOID*'s time. It is possible to eliminate the square root for $z$ using a conic drawing algorithm [Porter78], but that approach is slow when using high-resolution z-buffers [Duff80a].

The discriminant calculations above can be optimized further by manipulating equation (2) into the form $z^2 - 2\alpha z + \beta = 0$, which has the solutions $z = \alpha \pm \sqrt{\alpha^2 - \beta}$. The discriminant is quadratic in $x$, and $\alpha$ linear in $x$, so both can be incrementally updated in three additions, just like $a$, $b$, and $c$. By calculating the discriminant (rather than $b$ and $c$) incrementally, we save several multiplies over the current method. Also, there is no $a$ to divide by.

Perhaps a more fruitful optimization effort would focus on fast, approximate shading and square root calculation. Determining the precision required for these calculations (is double-precision floating point needed?) could also lead to significant speedups, especially for processors without floating point hardware, such as an Ikonas [Hart68], [Farmwald81], [Hanrahan83].

## 11. BETTER BOUNDS

*SOID*'s method for finding the bounding rectangle has flaws. First of all, if the eye is close to or inside a quadric, the bounding box can surround the eye, and some of the vertices fall behind it. When this happens, more complex tests are required if one wants to avoid scanning out the entire screen. Secondly, the bounding rectangle defined by the box does not fit the silhouette of rotated ellipsoids tightly. A *minimal* bounding rectangle can be found from the extrema of the silhouette conic curve.

Another case in which the bounding rectangle leads to inefficiency is in the rendering of narrow cylinders which cut diagonally across the screen. The bounding rectangle is a very poor approximation to the outline of the quadric in this case. Alternatives to the bounding rectangle, such as the convex hull and silhouettes, deserve investigation. Finding minimal bounds in *SSOID*, where they are more crucial than in *SOID*, would entail computing quadric-quadric intersection curves, which is a complex mathematical problem [Levin76]. One alternative approach is to approximate each quadric with a polyhedron, use constructive solid geometry techniques to find the intersection or difference of the two solids, and construct approximate bounds from that. A third method is similar, but works entirely in screen space. The silhouette conics of

the primary quadric and all of the bounds are computed, and their intersection, union, or difference (whatever is appropriate) found.

The bounds tests which *SSOID* currently uses are quite straightforward. At each pixel inside the primary quadric's bounding rectangle it checks against each of the bounding quadrics. Since tight bounds are not used to minimize the area scanned, it is especially important to optimize the evaluation of the bounding quadrics, since they are computed in the inner loop. The first optimization is to order the list of bounding quadrics so the ones most likely to invalidate a root will be checked first. A simple scheme for this keeps track of the last quadric to throw out a root, and checks that one first. Another optimization comes from the $x$ and $y$ coherence of the quadratic form. Since the ordering optimization above means we'll often test only one of the bounding quadrics per pixel, incrementalizing the quadratic form in $x$ does not make much sense. Doing that would force us to step through each of the bounding quadrics just to update their incremental variables. A better idea is to exploit $y$ coherence, so that updates need be done only once per scan line. While straightforward quadric evaluation requires 9 multiplies and 9 adds, using $y$ coherence it can be done in 5 multiplies and 5 adds per test, plus a little overhead for each scan line:

$$(q_{00}\cdot x+2q_{02}\cdot z+\{2q_{01}y+2q_{03}\})\cdot x+(q_{22}\cdot z+\{2q_{12}y+2q_{23}\})\cdot z+\{q_{11}y^2+2q_{13}y+q_{33}\}$$

where precalculated quantities are shown in braces.

## 12. PARAMETRIC QUADRICS

Whereas algebraic surfaces are defined implicitly, parametric surfaces are specified explicitly by three functions of two parameters $u$ and $v$:

$$\begin{aligned} x &= x(u,v) \\ y &= y(u,v) \\ z &= z(u,v) \end{aligned}$$

Although the parameterization is not explicit when quadrics are modeled, one can be found using a projection. Note that the subset of quadrics which *SOID* renders are all surfaces of revolution about the y axis. This allows a simple parameterization equivalent to a cylindrical projection. When rendering, the screen space coordinates $\{x,y,z\}$ of a point on the surface can be mapped to object space, where the parameterization is defined:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = M^* \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \begin{cases} x_o = \dfrac{x'}{w'} = r\sin u \\ y_o = \dfrac{y'}{w'} = -v \\ z_o = \dfrac{z'}{w'} = -r\cos u \end{cases} \qquad \begin{cases} u = -\tan^{-1}\dfrac{x_o}{z_o} = -\tan^{-1}\dfrac{x'}{z'} \\ v = -y_o = -\dfrac{y'}{w'} \end{cases}$$

$$\textit{where:}\quad r^2 = x_o^2+z_o^2 = -a_h v^2+2b_h v-c_h$$

So $u$ is the angle around the y axis, and $v$ the height. The signs are chosen so that

$u,v=0$ is at the top rear of the cylinder, $u$ points to the right on the front of the cylinder, and $v$ points down (just like $x$ and $y$ in a frame buffer). These parameters, which have range $-\pi \le u \le \pi$, $-1 \le v \le 1$, are scaled and offset before indexing the texture array.

## 13. TEXTURE MAPPING IN *SOID*

When using the *mip* texture filtering technique [Dungan78], [Williams83], a measure of the texture space area covered by each screen space pixel must be computed. When a rectangular pixel is mapped back to texture space, a curvilinear quadrilateral results. Since *mip* only filters squares, we must approximate this quadrilateral. Empirical tests [Heckbert83] have shown that a good formula for the side length of a square approximating a parallelogram is:

$$d = \max\left\{ \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2, \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right\}^{\frac{1}{2}} \tag{4}$$

The four partials are the texture coordinate increments when one steps down or to the right by one pixel in screen space. One way to find them is simply to subtract the $u$'s and $v$'s of adjacent pixels (the neighbors above and to the left, for instance) from those of the current pixel. This is facilitated by keeping one or two scan line's worth of $(u,v)$ pairs in an array. This method works fine for the interiors of quadrics, but gets a bit tricky when the ray for one of the neighboring pixels doesn't intersect the surface. Then one must check the other neighbors, make a wild guess, or find the increments analytically.

The current implementation of *SOID* finds the four partials analytically. It is a big numerical crunch, but more straightforward than the difference method. The partials are functions of the object space coordinates, the matrix $T=M^*$, and the screen space $z$ partials. For example:

$$\frac{\partial u}{\partial x} = -\frac{\partial}{\partial x}\tan^{-1}\frac{x'}{z'} = \frac{x\frac{\partial z'}{\partial x} - z\frac{\partial x'}{\partial x}}{x'^2+z'^2}$$

$$\frac{\partial x'}{\partial x} = t_{00}+t_{02}\frac{\partial z}{\partial x}, \qquad \frac{\partial z'}{\partial x} = t_{20}+t_{22}\frac{\partial z}{\partial x}, \qquad \textit{etc.}$$

$$\textit{so} \qquad \frac{\partial u}{\partial x} = \frac{x't_{20} - z't_{00} + (x't_{22} - z't_{02})\frac{\partial z}{\partial x}}{x'^2+z'^2}$$

The partials of $z$ can be derived from equation (2). Note that they depend on which root is used:

$$\frac{\partial z}{\partial x} = -\frac{q_{02}}{a} \pm \frac{bq_{02} - 2a(q_{00}x + q_{01}y + q_{03})}{a\sqrt{\xi}}$$

$$\frac{\partial z}{\partial y} = -\frac{q_{12}}{a} \pm \frac{bq_{12} - 2a(q_{10}x + q_{11}y + q_{13})}{a\sqrt{\xi}}$$

The four resulting formulas are:

$$\frac{\partial u}{\partial x} = \frac{x'_{20} - z'_{00} + \phi\frac{\partial z}{\partial x}}{x'^2 + z'^2} \qquad \frac{\partial v}{\partial x} = \frac{y'_{30} - w'_{10} + \psi\frac{\partial z}{\partial x}}{w'^2}$$

$$\frac{\partial u}{\partial y} = \frac{x'_{21} - z'_{01} + \phi\frac{\partial z}{\partial y}}{x'^2 + z'^2} \qquad \frac{\partial v}{\partial y} = \frac{y'_{31} - w'_{11} + \psi\frac{\partial z}{\partial y}}{w'^2}$$

*where:* $\phi = x'_{22} - z'_{02}$ *where:* $\psi = y'_{32} - w'_{12}$

## 14. BUMP MAPPING IN *SOID*

Bump mapping uses a texture to perturb the normal vectors of a surface so that the shading gives the illusion of relief. Following [Blinn78a], $P$ is an object space point on the quadric, $N$ the surface normal, $f$ the relief function, and $P' = P + f(u,v)\hat{N}$ the perturbed surface point. To compute the perturbed surface normal, which is necessary for shading calculations, we find tangent vectors along the $u$ and $v$ parameter directions and take their cross product:

we use subscripts for partial derivatives, e.g. $f_u = \frac{\partial f}{\partial u}$

$$P'_u = \frac{\partial}{\partial u}(P + f\hat{N}) \approx P_u + f_u\hat{N}$$

$$P'_v = \frac{\partial}{\partial v}(P + f\hat{N}) \approx P_v + f_v\hat{N}$$

$$N = P_u \times P_v, \qquad N' = P'_u \times P'_v = N + D$$

$$\textit{where: } D = \frac{f_u N \times P_v - f_v N \times P_u}{|N|}$$

For *SOID*'s parameterization,

$$P = \{r\sin u, -v, -r\cos u\} = \{x_o, y_o, z_o\}$$

$$P_u = \{r\cos u, 0, r\sin u\} \qquad |P_u| = r$$

$$P_v = \{\alpha\sin u, -1, -\alpha\cos u\} \qquad |P_v| = \sqrt{1+\alpha^2}$$

$$N = \{r\sin u, r\alpha, -r\cos u\} \qquad |N| = r\sqrt{1+\alpha^2}$$

*where:* $r = \sqrt{x_o^2 + z_o^2} = \sqrt{-a_h v^2 + 2b_h v - c_h}$ = *radius at a given* $y_o$

$$\alpha = \frac{\partial r}{\partial v} = \frac{-a_h v + b_h}{r}$$

Using the perpendicularity of the $u$ and $v$ tangent vectors ($P_u \cdot P_v = 0$), we find:

$$D = \frac{-f_u |P_v|^2 P_u - f_v |P_u|^2 P_v}{|N|}$$

$$= -f_u \sqrt{1+\alpha^2}\{\cos u,\ 0,\ \sin u\} - f_v \frac{r}{\sqrt{1+\alpha^2}}\{\alpha \sin u,\ -1,\ -\alpha \cos u\}$$

$$N' = N + D = \left\{\beta x_o + \gamma z_o,\ \ r(\alpha + \frac{1-\beta}{\alpha}),\ \ \beta z_o - \gamma x_o\right\}$$

$$\textit{where:}\quad \beta = 1 - \frac{f_v \alpha}{\sqrt{1+\alpha^2}}, \qquad \gamma = \frac{f_u \sqrt{1+\alpha^2}}{r}$$

These formulas apply to hyperboloids of one sheet, and hence to spheres, paraboloids, cones, and cylinders as well.

Since the parameterization is defined in terms of the canonical quadrics in object space, the bump mapping will be scale-invariant. Consequently, bumps will not flatten out as the object is scaled up, as they did for [Blinn78a], [Blinn78b]. Also, one can use the same formula for $d$, the filter area, as one would for texture mapping. As Blinn points out, antialiasing by low-pass filtering of $f$ is not equivalent to low-pass filtering after shading, but it's cheap, and it's much better than point-sampling, so that's the way *SOID* does it.

## 15. SHADING IN *SOID*

We will briefly cover *SOID*'s shading techniques. Note that the methods described here are not limited to quadrics.

A cheap way to do shading is to precompute a *light table* containing a coarse sampling of the brightness of the sky in a number of directions [Blinn78b], [Duff80a], [Duff82]. *SOID* makes a miniature (currently 49x49) table indexed by the $x$ and $y$ components of the screen space normal vector. The light table is analogous to a picture of a white sphere illuminated by the light sources. To shade a surface, *SOID* looks up the approximate intensity for each normal using bilinear interpolation in the light table. Since a single table is used for all objects, this method does not correctly simulate lights or cameras which are local to the scene. However, it can be significantly faster than straightforward use of a shading formula especially if there are many lights or the shading formula is slow. If the table is very coarse, say 16x16, the surface orientations can be coded into an 8-bit pixel value so that lighting can be changed in real time using colormap manipulations [Blinn79], [Sloan79].

If the table is very fine, say 512x512, one can also put reflections of objects in the scene into the table [Blinn76], [Blinn78b]. This *reflection mapping*, as it is often called, is good for simulating chrome surfaces. For highly curved surfaces, one will get *highlight aliasing* unless the appropriate area of the light table is filtered for each

pixel [Williams83]. The *miplight* option of *SOID* does such highlight antialiasing using *mip*. Computation of the filter area is quite easy because $u$ and $v$ for the light ''texture'' are simply the $x$ and $y$ components of the screen space surface normal. Formula (4) is used again.

## 16. PARAMETERIZATIONS FOR RULED QUADRICS

If all points of a surface belong to lines which lie entirely on the surface, it is said to be *ruled*. An ellipsoid is not a ruled surface but cones, cylinders, hyperboloids of one sheet, and hyperbolic paraboloids are (fig. 2).

The general technique for ruled quadric parameterization makes use of complex numbers [Maxwell51], [Pogorelov80], [VNR75]. Taking a quadric in canonical form:

$$ax^2+by^2+cz^2+dw^2 = 0$$

this can be written as

$$(x\sqrt{a}+iw\sqrt{d})(x\sqrt{a}-iw\sqrt{d}) + (y\sqrt{b}+iz\sqrt{c})(y\sqrt{b}-iz\sqrt{c}) = 0$$

and using the linear transformation of coordinates

$$x' = x\sqrt{a}+iw\sqrt{d}, \qquad y' = y\sqrt{b}+iz\sqrt{c}$$
$$w' = x\sqrt{a}-iw\sqrt{d}, \qquad z' = -y\sqrt{b}+iz\sqrt{c}$$

the equation becomes

$$x'w' = y'z'$$

We can define

$$x' = uz', \quad y' = uw'$$
$$x' = vy', \quad z' = vw'$$

Since they are linear in $x', y', z'$ and $w'$, each of these equations defines a plane and the intersection of each pair of planes a line of constant $u$ or $v$. These are called the *generators* of the ruled surface. If we set $x' = uv\lambda$, $y' = u\lambda$, $z' = v\lambda$, and $w' = \lambda$, for any constant $\lambda$, a parameterization for any quadric can be found. By inverting the transformation of coordinates we can find parametric formulas for $x$, $y$, $z$, and $w$:

$$x = \frac{x'+w'}{\sqrt{a}} = \frac{uv+1}{\sqrt{a}}$$
$$y = \frac{y'-z'}{\sqrt{b}} = \frac{u-v}{\sqrt{b}}$$
$$z = \frac{y'+z'}{i\sqrt{c}} = \frac{u+v}{i\sqrt{c}}$$
$$w = \frac{x'-w'}{i\sqrt{d}} = \frac{uv-1}{i\sqrt{d}}$$

These formulas turn out to be real (non-complex) if the surface is ruled [Pogorelov80].

As an example, for the hyperboloid of one sheet

$$x^2+y^2-z^2-w^2 = 0$$

we have $\sqrt{a}=\sqrt{b}=1$ and $\sqrt{c}=\sqrt{d}=i$, so

$$P(u,v) = \{1+uv,\ u-v,\ -u-v,\ 1-uv\}$$

Any quadric (ruled or unruled) can be expressed as a *rational biquadratic parametric* patch [Sommerville34], [Faux79], [Blinn78b], [Potmesil82] (*biquadratic* means quadratic in each of two variables, *rational* means each coordinate is a ratio of polynomials). Biquadratic patches are analogous to bicubics, except they have degree two instead of three. Surface points of a biquadratic can be written as $P(u,v)$, where:

$$x(u,v) = \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & r_{02} \\ 0 & r_{11} & r_{01} \\ r_{20} & r_{10} & r_{00} \end{pmatrix} \begin{pmatrix} v^2 \\ v \\ 1 \end{pmatrix}$$

with similar expressions for $y$, $z$, and $w$. The ruled quadric parameterization described above uses only terms of $uv$, $u$, $v$, and 1, so $r_{20}=r_{02}=0$ in this matrix, and the patch is actually *bilinear*.

*SOID* could texture map onto more types of quadrics if it were not limited to cylindrical projection parameterizations. This ruled surface technique is a possible alternative. It remains to be seen whether this parameterization can be made intuitive enough for modelers, however.

## 17. BIBLIOGRAPHY

### 17.1. Math


Dresden, Arnold, *Solid Analytical Geometry and Determinants*, Dover, New York, 1930.

Maxwell, E. A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, London, 1951.

Sommerville, D. M. Y., *Analytical Geometry of Three Dimensions*, Cambridge University Press, 1934.

Pogorelov, A. V., *Analytical Geometry*, Mir Publishers, Moscow, 1980.

*VNR Concise Encyclopedia of Mathematics*, Van Nostrand Reinhold, NY, 1975, pp. 543-547.


### 17.2. Numerical Analysis


Farmwald, Paul Michael, *On The Design of High Performance Digital Arithmetic Units*, PhD thesis, CS Dept, Stanford U., Aug. 1981.

Field, David A., and Alexander P. Morgan, *A Quick Method for Testing if a Second Degree Polynomial Has Solutions in a Given Box*, GMR-3656, General Motors Research Laboratories, April 1981.

Field, David A., and Alexander P. Morgan, ''A Quick Method for Testing Whether a Second-Degree Polynomial Has Solutions in a Given Box'', *IEEE Computer Graphics and Applications*, May 1982, pp. 65-68.

Hart, John F. et al, *Computer Approximations,* Kreiger Publishing Co., Malabar, Florida, 1968.

Morgan, Alexander P., *A Method for Computing All Solutions to Systems of Polynomial Equations,* GMR-3651, General Motors Research Laboratories, July 1981.

## 17.3. Computer Graphics

Bier, Eric Allan, *Solidviews, An Interactive Three-Dimensional Illustrator,* BS & MS thesis, Dept. of EE&CS, Massachusetts Institute of Technology, May 1983.

Blinn, James F., and Martin E. Newell, ''Texture and Reflection in Computer Generated Images'', *C.A.C.M.*, vol. 19, no. 10, Oct. 1976, pp. 542-547. (reprinted in *Tutorial: Computer Graphics*, Kellogg Booth, ed., IEEE catalog no. EHO 147-9, 1979.)

Blinn, James F., ''Simulation of Wrinkled Surfaces'', SIGGRAPH '78 Proceedings, *Computer Graphics,* vol. 12, no. 3, Aug. 1978, pp. 286-292.

Blinn, James F., *Computer Display of Curved Surfaces,* PhD thesis, CS Dept., Univ. of Utah, 1978, p. 15-17.

Blinn, James F., ''Raster Graphics'', *Tutorial: Computer Graphics,* Kellogg Booth, ed., IEEE catalog no. EHO 147-9, 1979.

Blinn, James F., ''Geometric Representations in Computer Graphics, Part I'', *SIGGRAPH '80 Advanced 3D Raster Graphics seminar notes,* July, 1980, and *SIGGRAPH '81 Advanced Image Synthesis seminar notes,* Aug. 1981.

Blinn, James F., ''A Generalization of Algebraic Surface Drawing'', *ACM Transactions on Graphics,* vol. 1, no. 3, July 1982, pp. 235-256.

Brooks, Joan, *Extension and Adjuncts to the BRL-COMGEOM Program,* for Ballistic Research Laboratories, Aug. 1974, AD/A-000 897, National Technical Information Service, U.S. Dept. of Commerce, Springfield, VA 22161.

Davis, J. R., R. Nagel, and W. Guber, ''A Model Making and Display Technique for 3-D Pictures'', *Proceedings of the 7th Annual Meeting of UAIDE,* San Francisco, Oct. 1968, pp. 47-72.

Duff, Tom, *Quadric Surfaces with Parametric Mapping,* NYIT Computer Graphics Lab, Jan. 1980, (unfinished).

Duff, Tom, *The Mat Manual,* second edition, NYIT Computer Graphics Lab, July 1980.

Duff, Tom, Paul Heckbert, and Michael Chou, *The SOID, SSOID, and ROID Manual,* third edition, NYIT Computer Graphics Lab, Sept. 1982.

Dungan, William, Jr., Anthony Stenger, and George Sutty, ''Texture Tile Considerations for Raster Graphics'', SIGGRAPH '78 Proceedings, *Computer Graphics,* vol. 12, no. 3, Aug. 1978, pp. 130-134.

Faux, I. D., and M. J. Pratt, *Computational Geometry for Design and Manufacture,* Ellis Horwood Ltd., Chichester, England, 1979.

Goldman, Ronald N., ''Quadrics of Revolution'', *IEEE Computer Graphics and Applications,* March-April 1983, pp. 68-76.

Goldman, Ronald N., ''Two Approaches to a Computer Model for Quadric Surfaces'', *IEEE Computer*

*Graphics and Applications,* Sept. 1983, pp. 21-24.

Goldstein, R. A., and R. Nagel, ‘‘3-D Visual Simulation’’, *Simulation,* vol. 16, no. 1, Jan. 1971, pp. 25-31.

Hanrahan, Pat, ‘‘Ray Tracing Algebraic Surfaces’’, SIGGRAPH ’83 Proceedings, *Computer Graphics,* vol. 17, no. 3, July 1983, pp. 83-90.

Heckbert, Paul, *Texture Mapping Polygons in Perspective,* Technical Memo No. 13, NYIT Computer Graphics Lab, April 1983.

Herbison-Evans, Don, ‘‘Animated Cartoons by Computers Using Ellipsoids’’, *Proc. 6th Australian Computer Conf.,* 1974, pp. 811-823.

Herbison-Evans, Don, ‘‘NUDES 2: A Numeric Utility Displaying Ellipsoid Solids, Version 2’’, SIGGRAPH ’78 Proceedings, *Computer Graphics,* vol. 12, no. 3, Aug. 1978, pp. 354-356.

Herbison-Evans, Don, ‘‘Rapid Raster Ellipsoid Shading’’, *Computer Graphics,* vol. 13, no. 4, Feb. 1980, pp. 355-361.

Herbison-Evans, Don, ‘‘Real-Time Animation of Human Figure Drawings with Hidden Lines Omitted’’, *IEEE Computer Graphics and Applications,* vol. 2, no. 9, Nov. 1982, pp. 27-33.

Levin, Joshua Z., *A Parametric Algorithm for Drawing Pictures of Solid Objects Bounded by Quadric Surfaces,* Tech. Rep. CRL-46, School of Eng., Rensselaer Polytechnic Institute, Troy, NY, March 1976.

Levin, Joshua Z., ‘‘A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces’’, *C.A.C.M.,* vol. 19, no. 10, Oct. 1976, pp. 555-563.

Levin, Joshua Z., *Mathematical Models for Determining the Intersections of Quadric Surfaces,* Tech. Rep. CRL-61, Rensselaer Polytechnic Inst, Troy, NY, 1978. (reprinted in *Computer Graphics and Image Processing,* vol. 11, 1979, pp. 73-87.)

Levin, Joshua Z., *QUISP: A Computer Processor for the Design and Display of Quadric-Surface Bodies,* PhD thesis, Tech. Rep. IPL-TR-80-003, Image Processing Lab, Rensselaer Polytechnic Inst., Troy, NY, Jan. 1980.

Levin, Joshua Z., ‘‘QUADRIL: A Computer Language for the Description of Quadric-Surface Bodies’’, SIGGRAPH ’80 Proceedings, *Computer Graphics,* vol. 14, July 1980, pp. 86-92.

Mahl, R., *Visible Surface Algorithms for Quadric Patches,* Tech. Report UTEC-CSc-70-111, CS Dept. Univ. of Utah, Dec. 1970.

Mahl, R., ‘‘Visible Surface Algorithm for Quadric Patches’’, *IEEE Trans. Computers C-21,* Jan. 1972, pp. 1-4.

Newell, Martin E., *The Utilization of Procedure Models in Digital Image Synthesis,* PhD thesis, CS Dept. Univ. of Utah, Summer 1975, p. 7-8.

Newman, William M., and Robert F. Sproull, *Principles of Interactive Computer Graphics,* 2nd ed., McGraw-Hill, New York, 1979.

Potmesil, Michael, *Generating Three-Dimensional Surface Models of Solid Objects from Multiple Projections,* PhD thesis, IPL-TR-033, Oct. 1982, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY.

Roth, Scott D., ‘‘Ray Casting for Modeling Solids’’, *Computer Graphics and Image Processing*, vol. 18 no. 2, Feb. 1982, pp. 109-144. Preliminary version, ‘‘Ray Casting as a Method for Solid Modeling’’, General Motors Research Publication GMR-3466, appeared in *SIGGRAPH ’81 Solid Modeling seminar notes.*

Sarraga, Ramon F., *Algebraic Methods for Intersections of Quadric Surfaces in GMSolid*, GMR-3944, Computer Science Department, General Motors Research Labs, Jan. 1982.

Sarraga, Ramon F., ‘‘Algebraic Methods for Intersections of Quadric Surfaces’’, *Computer Graphics and Image Processing*, vol. 22, no. 2, May 1983, pp. 222-238.

Sloan, Kenneth R., and Christopher M. Brown, ‘‘Color Map Techniques’’, *Computer Graphics and Image Processing*, vol. 10, Aug. 1979, pp. 297-317.

Weiss, Ruth A., ‘‘BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Views of Combinations of Plane and Quadric Surfaces’’, *J. ACM*, vol. 13, no. 2, April 1966, pp. 194-204. (reprinted in *Tutorial and Selected Readings in Interactive Computer Graphics*, Herbert Freeman, ed., IEEE catalog no. EHO 156-0, 1980.)

Williams, Lance, ‘‘Pyramidal Parametrics’’, SIGGRAPH ’83 Proceedings, *Computer Graphics*, vol. 17, no. 3, Aug. 1983, pp. 1-11.

Woon, Peter Y., *On the Computer Drawing of Solid Objects Bounded by Quadric Surfaces*, New York University, Report TR-403-3, June 1969.

Woon, Peter Y., *A Computer Procedure for Generating Visible-Line Drawings of Solids Bounded by Quadric Surfaces*, PhD thesis, Tech. Rep. 403-15, Dept. EE, School of Eng. and Sci., New York U., Dec. 1970. Also AD-724 744 National Technical Information Service, U.S. Dept. of Commerce, Springfield, VA 22161. from N.T.I.S., U.S. Dept. Commerce.

Woon, Peter Y., and Herbert Freeman, ‘‘A Computer Procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces’’, *Proceedings IFIP Congress, Information Processing 71*, vol. 2, North-Holland Pub. Co., Amsterdam, 1971, pp. 1120-1125. (reprinted in *Tutorial and Selected Readings in Interactive Computer Graphics*, Herbert Freeman, ed., IEEE catalog no. EHO 156-0, 1980.)

## 17.4. Spheres and Molecules

Badler, Norman, Joseph O’Rourke, and Hasida Tolzis, ‘‘A Spherical Representation of a Human Body for Visualizing Human Movement’’, *Proceedings of the IEEE*, vol. 67, no. 10, Oct. 1979, pp. 1397-1403.

Connolly, Michael L., ‘‘Computer Display of the Outer Surfaces of Large Molecules’’, *Nicograph ’83 Proceedings*, 1983. pp. 313-322.

Knowlton, Ken, and Lorinda Cherry, ‘‘ATOMS, a Three-D Opaque Molecule System’’, *Computers and Chemistry*, vol. 1, no. 3, 1977, pp. 161-166.

Knowlton, Ken, ‘‘Computer-Aided Definition, Manipulation and Depiction of Objects Composed of Spheres’’, *Computer Graphics*, vol. 15, no. 1, April 1981, pp. 48-71.
Also *Computer Graphics*, vol. 15, no. 4, Dec. 1981, pp. 352-375.

Max, Nelson, ‘‘ATOMLLL - ATOMS with Shading and Highlights’’, SIGGRAPH ’79 Proceedings, *Computer Graphics*, vol. 13, no. 2, Aug. 1979, pp. 165-173.

Max, Nelson, ‘‘ATOMLLL with Transparency and Shadows’’, *Proc. 15th Annual Hawaiian Intl. Conf.*

*Systems Sciences*, 1983, p. 407. (reprinted in *SIGGRAPH '82 Advanced Image Synthesis seminar notes*, July 1982.)

Max, Nelson, ''Computer Representation of Molecular Surfaces'', *IEEE Computer Graphics and Applications*, vol. 3, no. 7, Aug. 1983, pp. 21-29. (reprinted in *Nicograph '83 Proceedings*, 1983. pp. 323-331.)

Pique, Michael E., *Fast 3-D Display of Space-Filling Molecular Models*, UNC CS Tech. Report 83-004, U. of North Carolina, 1983.

Porter, Tom, ''Spherical Shading'', SIGGRAPH '78 Proceedings, *Computer Graphics*, vol. 12, no. 3, Aug. 1978, pp. 282-285.

Porter, Tom, ''The Shaded Surface Display of Large Molecules'', SIGGRAPH '79 Proceedings, *Computer Graphics*, vol. 13, no. 2, Aug. 1979, pp. 234-236.

Staudhammer, John, ''On Display of Space Filling Atomic Models in Real-Time'', SIGGRAPH '78 Proceedings, *Computer Graphics*, vol. 12, no. 3, Aug. 1978, pp. 167-172.