Appendix A

Finding Normals for Analytic Surface

Analytic surfaces are smooth, differentiable surfaces that are described by a mathematical equation (or set of equations). In many cases, the easiest surfaces to find normals for are analytic surfaces for which you have an explicit definition in the following form:

V(s,t)=[X(s,t) Y(s,t) Z(s,t)]

where *s* and *t* are constrained to be in some domain, and X, Y, and Z are differentiable functions of two variables. To calculate the normal, find

$$\frac{\partial V}{\partial s} \quad and \quad \frac{\partial V}{\partial t}$$

which are vectors tangent to the surface in the *s* and *t* directions. The cross product

$$\frac{\partial V}{\partial s} \times \frac{\partial V}{\partial t}$$

is perpendicular to both and, hence, to the surface. The following shows how to calculate the cross product of two vectors. (Watch out for the degenerate cases where the cross product has zero length).

$$\lfloor v_x \; v_y \; v_z \rfloor \times \lfloor w_x \; w_y \; w_z \rfloor = \lfloor (v_y w_z - w_y \; v_z)(w_x v_z - v_x \; w_z) \; (v_x w_y - w_x \; v_y) \rfloor$$

The resulting vector should be normalized. To normalize a vector [x y z], calculate its length

$$Length = \sqrt{x^2 + y^2 + z^2}$$

and divide each component of the vector by the length.

As an example of these calculations, consider the analytic surface

$$V(s, t)=[\ s^2\ t^3\ 3\text{-}st]$$

From this

$$\frac{\partial V}{\partial s}=[2s\ 0\ -t],\quad \frac{\partial V}{\partial t}=[0\ 3t^2\ -s],\ and\ \frac{\partial V}{\partial s}\times\frac{\partial V}{\partial t}=[-3t^3\ 2s^2\ 6st^2],$$

So, for example, when s=1 and t=2, the corresponding point on the surface is (1,8,1), and the vector (-24,2,24) is perpendicular to the surface at that point. The length of this vector is 34, so the unit normal vector is (-24/34, 2/34, 24/34) = (-0.70588, 0.058823, 0.70588).

For analytic surfaces that are described implicitly, as F(x, y, z) = 0, the problem is harder. In some cases, on of the variables can be solved, z = G(x, y), and put it in the explicit form given previously:

$$V(s, t) = [s\quad t\ \ G(s, t)]$$

Then continue as described earlier.

If you can't get the surface equation in an explicit form, you might be able to make use of the fact that the normal vector is given by the gradient

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

evaluated at a particular point (x, y, z). Calculating the gradient might be easy, but finding a point that lies on the surface can be difficult. As an example of an implicitly defined analytic function, consider the equation of a sphere of radius 1 centered at the origin:

$$x^2 + y^2 + z^2 - 1 = 0$$

This means that

$$F(x, y, z) = x^2 + y^2 + z^2 - 1$$

$$z = \pm\sqrt{1 - x^2 - y^2}$$

which can be solved for z to yield

Thus, normals can be calculated from the explicit from

$$V(s,t) = \begin{bmatrix} s & t & \sqrt{1 - s^2 - t^2} \end{bmatrix}$$

as described previously.

If you could not solve for z, you could have used the gradient

$$\nabla F = [2x\ 2y\ 2z]$$

as long as you could find a point on the surface. In this case, it's not so hard to find a point --- for example, (2/3, 1/3, 2/3) lies on the surface. Using the gradient, the normal at this point is (4/3, 2/3, 4/3). The unit-length normal is (2/3, 1/3, 2/3), which is the same as the point on the surface, as expected.

Finding Normals from Polygonal Data

As mentioned previously, you often want to find normals for surfaces that are described with polygonal data such that the surfaces appear smooth rather that faceted. In most cases, the easiest way for you to do this is to calculate the normal vectors for each of the polygonal facets and then to average the normals for neighboring facets. Use the averaged normal for the vertex that the neighboring facets have in common. Figure A-1 shows a surface and its polygonal approximation.

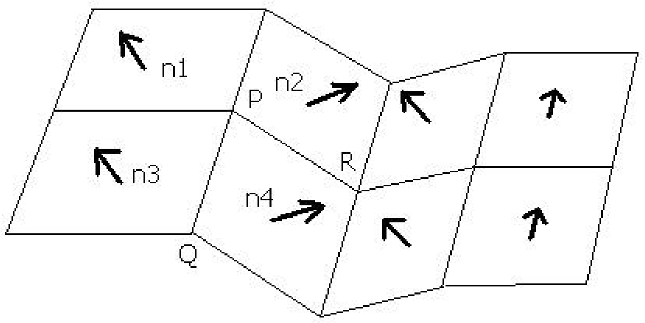

Figure A-1: Averaging and normal vecters.

To find the normal for a flat polygon, take any three vertices v1, v2, and v3 of the polygon that do not lie in a straight line. The cross product

$$[v_1 - v_2] \text{ x } [v_2 - v_3]$$

is perpendicular to the polygon. (Typically, you want to normalize the resulting vector.) Then you need to average the normals for adjoining facets to avoid giving to much weight to one of them. For instance, in the example shown in Figure A-1, if n_1, n_2, and n_3 are the normals for three polygons meeting a t point p, calculated $n_1+n_2+n_3$ and the normalize it. (you can get a better average if you weight the normals by the size of the angles at the shared intersection.) The resulting vector can be used as the normal for point P.

Homogeneous Coordinates

OpenGL commands usually deal with two-and three-dimensional vertices, but in fact all are treated internally as three-dimensional homogeneous vertices comprising four coordinates. Every column vector $(x, y, z, w)^T$ represents a homogeneous vertex if at least on of its elements is nonzero. If the real number *a* is nonzero, the $(x, y, z, w)^T$ and $(ax, ay, az, aw)^T$ represent the same homogeneous vertex. (This is just like fractions: x/y = (ax)/(ay).) A three-dimensional euclidean space point $(x, y, z)^T$ becomes the homogeneous vertex with coordinates $(x, y, z, 1)^T$, and the two-dimensional euclidean point $(x, y)^T$ becomes $(x, y, 0.0, 1.0)^T$.

As long as *w* is nonzero, the homogeneous vertex $(x, y, z, w)^T$ corresponds to the three-dimensional point $(x/w, y/w, z/w)^T$. Fi w = 0.0, it corresponds to no euclidean point, but rather to some idealized “point at infinity”. To understand this point at infinity, consider the point (1, 2, 0, 0), and note that the sequence of points (1, 2, 0, 1), (1, 2, 0, 0.01), and (1, 2.0, 0.0, 0.0001), corresponds to the euclidean points (1, 2), (100, 200) and (10000, 20000). This sequence represents points rapidly moving toward infinity along the line 2x = y. Thus, you can think of (1, 2, 0, 0) as the point at infinity in the direction of that line.

Note: OpenGL might not handle homogeneous clip coordinates with x<0 correctly. To be sure that your code is portable to all OpenGL systems, use only nonnegative *w* values.

Transforming Vertices

Vertex transformations (such as rotations, translations, scaling, and shearing) and projections (such as perspective and orthographic) can all be represented by applying an appropriate 4x4 matrix to the coordinates representing the vertex. If *v* represents a homogeneous vertex and **M** is a 4x4 transformation matrix, the **Mv** is the image of **v** under the transformation by **M**. In computer-graphics applications, the transformations used are usually nonsingular—in other words, the matrix **M** can be inverted. This is not required, but some problems arise with nonsingular transformations.

After transformation, all transformed vertices are clipped so that x, y, and z are in the range [-*w, w*] (assuming *w>0*). Note that this range corresponds in euclidean space to [-1.0, 1.0].

Transforming Normals

Normal vectors are not transformed in the same way as vertices or position vectors. Mathematically, it is better to think of normal vectors not as vectors, but as planes perpendicular to those vectors. Then, the transformation rules for normal vectors are described by the transformation rules for perpendicular planes.

A homogeneous plane is denoted by the row vector (a, b, c, d), where at least one of a, b, c, or d is nonzero. If q is a nonzero real number, then (a, b, c, d) and (qa, qb, qc, qd) represent the same plane. A point $(x, y, z, w)^T$ is not the plane (a, b, c, d) if ax+by+cz+dw=0. (If w=1, this is the standard description of a euclidean plane.) In order for (a, b, c, d) to represent a euclidean plane, at least one of a, b, or c must be nonzero. If they are all zero, then (0, 0, 0, d) represents the "plane at infinity," which contains all the "points at infinity."

If **p** is a homogeneous plane and **v** is an homogeneous vertex, then the statement "**v** lies on plane **p**" is written mathematically as **pv=0**, where **pv** is normal matrix multiplication. If **M** is a nonsingular vertex transformation (that is, a 4x4 matrix that has an inverse $\mathbf{M}^{-1}$), then **pv=0** is equivalent to $\mathbf{pM}^{-1}\mathbf{Mv=0}$, so **Mv** lies on the plane $\mathbf{pM}^{-1}$. Thus, $\mathbf{pM}^{-1}$ is the image of the plane under the vertex transformation **M.**

If you like to think of normal vectors as vectors instead of as the planes perpendicular to them, let **v** and **n** be vectors such that **v** is perpendicular to **n.** Then, $\mathbf{n}^{\mathbf{T}}\mathbf{v=0}$, thus, for an arbitrary nonsingular transformation **M,** $\mathbf{n}^{\mathbf{T}}\mathbf{M}^{-1}\mathbf{Mv=0}$, which means that $\mathbf{n}^{\mathbf{T}}\mathbf{M}^{-1}$ is the transpose of the transformed normal vector. Thus, the transformed normal vector is $(\mathbf{M}^{-1})^{\mathbf{T}}\mathbf{n}$. In order words, normal vectors are transformed by the inverse transpose of the transformation that transforms points.

Transformation Matrices

Although any nonsingular matrix **M** represents a valid projective transformation, a few special matrices are particularly useful. These matrices are listed in the following subsections.

Translation

The call **glTranslate***(x, y, z) generates **T**, where

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad and \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

The call **glScale***(x, y, z) generates **S**, where

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad and \quad S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Notice that S^{-1} is defined only if x, y, and z are all nonzero.

Rotation

The call **glRotate***(a, x, y, z) generates R as follows:

Let v = $(x, y, z)^T$, and u = v/||v|| = $(x', y', z')^T$.

Also let

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

and M = uu^T + (cos a)(I-uu^T) + (sin a) S

Then

$$R = \begin{bmatrix} m & m & m & 0 \\ m & m & m & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where *m* represents elements from M, which is a 3x3 matrix.

The **R** matrix is always defined. If x = y = z =0, then **R** is the identity matrix. You can obtain the inverse of **R, R^{-1}**, by substituting –*a* for *a*, or by transposition.

The **glRotate*()** command generates a matrix for rotation about an arbitrary axis.

The corresponding matrices are as follows:

$$glRotate*(a,1,0,0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$glRotate*(a,0,1,0): \begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$glRotate*(a,0,0,1): \begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As before, the inverses are obtained by transposition.

Perspective Projection

The call glFrustum(*l, r, b, t, n, f*) generates **R**, where

$$R = \begin{bmatrix} \frac{2n}{r-1} & 0 & \frac{r+1}{r-1} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad and \quad R^{-1} = \begin{bmatrix} \frac{r-1}{2n} & 0 & 0 & \frac{r+1}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{bmatrix}$$

R is defined as long as *l* ¹ *r*, *t* ¹ *b,* and *n* ¹ *f*.

Orthographic Projection

The call **glOrtho(*l, r, b, t, n, f*)** generates **R**, where

$$R = \begin{bmatrix} \frac{2}{r-1} & 0 & 0 & -\frac{r+1}{r-1} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad and \quad R^{-1} = \begin{bmatrix} \frac{r-1}{2} & 0 & 0 & \frac{r+1}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t+b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R is defined as long as *l* ¹ *r*, *t* ¹ *b,* and *n* ¹ *f*.