Math: What's the Use?

Valerie A. Miller
Department of Mathematics & Statistics
Georgia State University
Atlanta, GA 30303-3083

Abstract: In this presentation we will discuss some of the different mathematics used in generating graphics images, from basic Euclidean geometry to the basics of spline interpolation.

Introduction:

Behind all great (and not so great) computer graphics images and/or animations stands a great lady. Her name? Mathematics. Whether it is a spline or parametric equations that generate the curve, the various geometries that provide the illusion of 3D, or the vector theory used in reflections, rotations, and shading, just about all aspects of a computer generated image relies on mathematics. In this paper we will discuss some of the different mathematics used in generating graphics images, from basic Euclidean geometry to the basics of spline interpolation. We begin with an overview of the simpler concepts in computer graphics, line drawing and transformations. In section 2 we examine how the illusion of 3D is created; in section 3 we will examine the mathematics involved in lighting our scene; and finally, in section 4, we will examine the creation of objects that do not have a simple geometric representation.

Section 1: The Basics

Computer graphics techniques have always relied heavily on mathematics for their implementation. Simple line and conic drawing algorithms (e.g., BRES65, BRES77, PITT67, and VANA84) show the importance of a thorough understanding of basic algebra, geometry and multivariate calculus. Calculus? Yes, gradients are used to determine vectors perpendicular to the tangents to conics other than circles. This is used when determining which pixel to "turn on" to create the most realistic-looking curve. Even pattern filling and line clipping rely on these basic concepts. Then, of course, clipping requires knowledge of the widowing system and coordinate systems in general. So we're back to geometry again. When creating an object to be represented, a mathematical model representing this object must be created. Whether it is a simple two dimensional (2D) or three dimensional (3D) geometric figure or a more complicated curve or surface, each object must have a mathematical representation. This representation is given in "real-world" coordinates, i.e., real values for which the mathematical model is valid. To create the graphics image to represent our object we must change the real-world coordinates (window or user coordinates) of our object to the normalized device coordinates and/or the viewport coordinates. This change requires what is commonly referred to as a window-to-viewport transformation. This transformation of user coordinates (x_{user}, y_{user}) to normalized device coordinates (x_{norm}, y_{norm}) is a simple linear function that is a combination of scaling and translation:

$$x_{norm} = \frac{x_{user} - w_{x\_min}}{w_{x\_max} - w_{x\_min}}\left(v_{x\_max} - v_{x\_min}\right) + v_{x\_min}$$

$$y_{norm} = \frac{y_{user} - w_{y\_min}}{w_{y\_max} - w_{y\_min}}\left(v_{y\_max} - v_{y\_min}\right) + v_{y\_min}$$

Here, w_{x_min}, w_{x_max} w_{y_min}, and w_{y_max}, are the minimum and maximum window (user) coordinates in the x and y directions and v_{x_min}, v_{x_max} , v_{y_min} , and v_{y_max} are the minimum and maximum viewport coordinates in the x and y directions.

Once we have the ability to represent our object we may wish to change its position or size or inclination. Each of these transformations of an object, whether it is a translation, rotation, or scaling is accomplished with the assistance of vectors and matrices, i.e., linear algebra. Two-dimensional (2D) transformations are accomplished as follows:

translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

scaling:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation about the origin:
(counterclockwise rotation)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos q & -\sin q \\ \sin q & \cos q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Here the point $(x, y)^T$ belongs to the original object and the point $(x', y')^T$ belongs to the transformed object. To rotate about a point other than the origin, say point P, we first translate P to the origin, rotate the object, and then translate back.

In order to treat all three transformations equally and to be able to easily compose them, we extend our two dimensional object to a 3 dimensional representation -- we do this by creating what are known as homogeneous coordinates: $(x, y)^T \rightarrow (x, y, 1)^T$. This is done so that the different functions that are to be applied to our object (or primitive) can be composed into one matrix transformation and this one transformation is applied, rather than incurring the cost and time of applying the separate transformations to the object. The individual transformation matrices now are:

translation:

$$\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

scale:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotation:

$$\begin{bmatrix} \cos q & -\sin q & 0 \\ \sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When working with three dimensional objects, we simply extend our 2D ideas to 3D. These transformations are applied to similar homogeneous representations: $(x, y, z)^T \rightarrow (x, y, z, 1)^T$
translation:

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scaling:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotations:
about z-axis:

$$\begin{bmatrix} \cos q & -\sin q & 0 & 0 \\ \sin q & \cos q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

about x-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos q & -\sin q & 0 \\ 0 & \sin q & \cos q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

about y-axis:

$$\begin{bmatrix} \cos q & 0 & -\sin q & 0 \\ 0 & 1 & 0 & 0 \\ \sin q & 0 & \cos q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, to handle the generic case of rotating by an angle θ about an arbitrary direction given by the line segment beginning at $P_1 = (x_1, y_1, z_1)^T$ and ending at $P_2 = (x_2, y_2, z_2)^T$ we apply a composition of transformations. Set $(a, b, c)^T = P_2 - P_1$ and $L = \sqrt{a^2 + b^2 + c^2}$ and $p = \sqrt{b^2 + c^2}$

Step 1. Translate the line and the point to be rotated to align with the z-axis:

$$T_{-P_1} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2. Rotate the line around the x-axis until it is in the xz plane. Note that right triangle trigonometry is used to determine the values for $\cos \alpha = c/p$ and $\sin \alpha = b/p$.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/p & -b/p & 0 \\ 0 & b/p & c/p & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3.Rotate once more to put the line on the z-axis:

$$R_y = \begin{bmatrix} p/L & 0 & -a/L & 0 \\ 0 & 1 & 0 & 0 \\ a/L & 0 & p/L & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4.Rotate through the desired angle θ:

$$R_z = \begin{bmatrix} \cos q & -\sin q & 0 & 0 \\ \sin q & \cos q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 5.Reverse steps 3, 2, and 1 to place the point back to its relative position.

So, the final composed transformation is:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T_{P_1} R_{-x} R_{-y} R_z R_y R_x T_{-P_1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Section 2: It's All an Illusion

The idea of trying to present the illusion of a three-dimensional picture on a two-dimensional surface had its beginnings in the Renaissance. The painters of the period utilized the idea of a "point at infinity" to represent the place where parallel lines seem to intersect. Mathematicians in their attempts to prove Euclid's parallel postulate developed non-Euclidean geometries [WYLI64, HART67] -- geometries that do not assume one or more of Euclidean Geometry's postulates -- and as a result provided firm mathematical foundations for this idea of a "point at infinity." When we attempt to represent a three-dimensional object on a graphics device we are utilizing these concepts. There are two types of projections that are commonly used to create these representations: perspective and parallel. If the distance from the center of the projection to the projection (or view) plane is finite then the projection is perspective; if the distance is infinite, the projection is parallel. Perspective projections do not preserve angles and distances, while parallel projections do. Figure 1 provides an illustration of the two basic projection types. Since it is the perspective projection that creates the more "realistic" representation, the illusion of 3D, it is this projection method that we will discuss here.

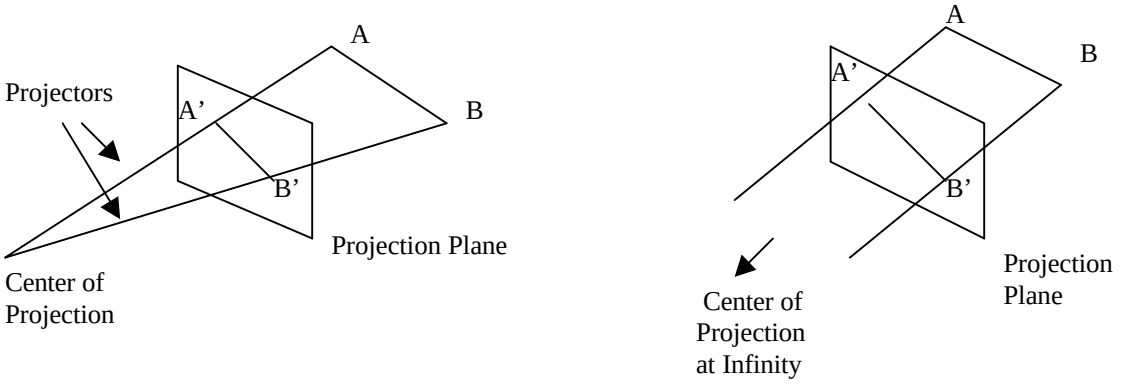

Figure 1

We start by assuming that the projection plane is perpendicular to the z axis and is located at z = d. The center of projection is at the origin. To determine the projection $P_p = (x_p, y_p, z_p)^T$ of a point $P = (x, y, z)^T$ we use similar triangles to get

$$x_p = \frac{x}{z/d}, \; y_p = \frac{y}{z/d}$$

and $z_p = d$. The division by z causes the projection of more distant objects to be smaller than that of closer objects.

A more general formulation was developed by N. Weingarten and the details can be found in FOLE90. We summarize this formulation here. We still assume that the projection plane is perpendicular to the z axis and is located at $z = z_p$, but now the center of projection is a distance Q from the point $(0,0,z_p)^T$, the intersection of the projection plane with the z axis. If the normalized direction vector from $(0,0,z_p)^T$ to the center of projection is $(d_x, d_y, d_z)^T$ then it can be shown that the projection P_p can be computed by

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\frac{d_x}{d_z} & z_p\frac{d_x}{d_z} \\ 0 & 1 & -\frac{d_y}{d_z} & z_p\frac{d_y}{d_z} \\ 0 & 0 & -\frac{z_p}{Qd_z} & \frac{z_p^2}{Qd_z}+z \\ 0 & 0 & -\frac{1}{Qd_z} & \frac{z_p}{Qd_z}+1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and

$$P_p = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} X/W \\ Y/W \\ Z/W \end{bmatrix}$$

This matrix transformation provides a one-point perspective projection. The vanishing point (or the point at infinity) is given by $(Qd_x, Qd_y, z_p)^T$.

There are many variations of this type or projection, for example, those that allow for multiple vanishing points. A summary of the mathematical effort required to transform a three-dimensional world coordinate into a two-dimensional device coordinate is given in Figure 2 and, again, the details may be found in FOLE90.

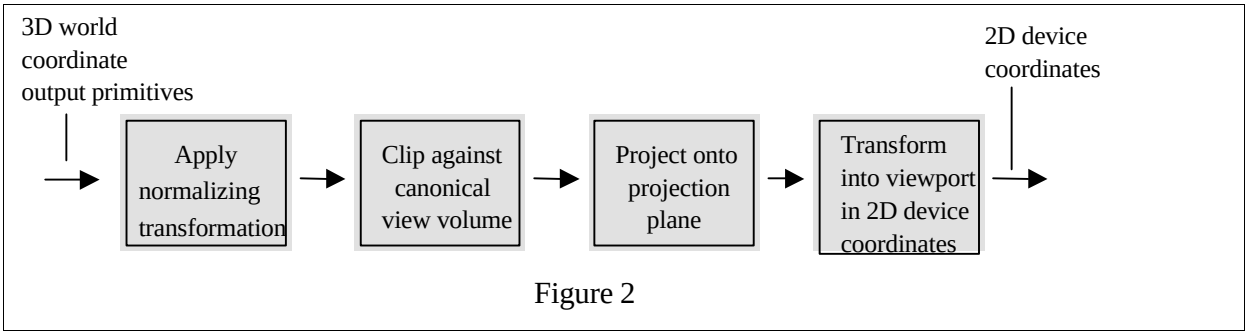

Figure 2

Section 3: A Little Light on the Subject

When considering the lighting of a scene one must consider the interaction of light with the surface of an object in the scene and model this interaction. One must consider the light emitting sources (a point source or distributed source) as well as the light reflecting sources (ambient light or background light). There are also two types of reflection to be considered: diffuse and specular. In diffuse reflections, what incoming light that is not absorbed is reflected off the surface in random directions, while in specular reflections the light reflects in a nearly fixed direction without any absorption.

In order to compute the overall illumination of a given point $P = (x, y, z)^T$, we need the following information:

1. the direction of the light source **L** (there could be more than one):
2. the normal vector **N**
3. the reflection vector **R** (there could be more than one)
4. the viewing vector **V**

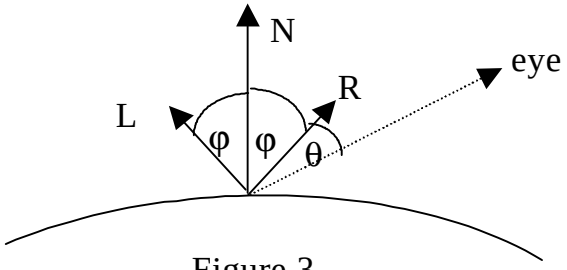

Figure 3

If $\mathbf{S} = (S_x,S_y,S_z)^T$ is the position of the light source and $\mathbf{E} = (E_x,E_y,E_z)^T$ is the position of the eye, then **L** = **S** - **P** and **V** = **E** - **P**. Each of **L** and **V** must be normalized, i.e., made to have length one. The computation of **N** and **R** can be complicated depending on the type of surface. **N** and **R** must also be nomalized. Once **L**, **V**, **N**, and **R** have been determined then $\cos\varphi = \mathbf{L}\cdot\mathbf{N}$ and $\cos\theta = \mathbf{V}\cdot\mathbf{R}$ and the Phong (PHON75) model of illumination can be computed by (POKO89):

$$I_* = I_{a*}k_{d*} + I_{p*}\left(k_{d*}\cos j/d + k_s\cos^n q\right)$$

where

* = r, g, or b (for red, green, or blue)
I_{a*} = intensity of ambient light for *
k_{d*} = diffuse reflectivity of *
k_S = a constant that estimates the specular reflection coefficient ($0 \le k_S \le 1$)
I_{p*} = intensity of point source of light for *
n = measure of "shininess" of the surface -- very shiny = large n (> 150)
d = distance to the point source or distributed source of light

There are many other models that could be considered, for example, GOUR71, WARN83, VERB84, and NISH85.

Section 4: The Spline's the Thing

When trying to represent an object mathematically, quite often it is impossible to construct an appropriate model using only simple geometric shapes. If this is the case, then arbitrary curves need to be created to model the object. The curves of choice are splines and other parametric curves. These curves are used in animation to generate in-between frames, or to create curves when only a few data points describing the curve are known. Yet another application of these curves is the generation of arbitrary surfaces. Each of these applications has essentially the same premise: from a few data points, generate a smooth curve based on this data. This curve should be simple to compute and easy to modify if a change in data is necessary.

Parametric equations are used to represent curves that are not easily represented or are impossible to be represented by a function. Examples of this would be curves that have loops or places of infinite slopes. In this case we use a parameter, say t, and describe each variable x and y as functions of t, x = x(t) and y = y(t), over some range of t. If we are working in three dimensions then each of x, y, and z would be parametrized. The slopes of tangential lines for parametric curves are simple to compute. For example,

$$\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx} = \frac{dy}{dt} \Big/ \frac{dx}{dt},$$

i.e. the chain rule of differentiation becomes simply a quotient.

Splines are piece-wise defined polynomial functions of, usually, low degree. If you have data points (x_0, y_0), ... (x_n, y_n), x_i 's distinct, then the Natural Cubic Spline, S(x), will interpolate this data (i.e., $S(x_i) = y_i$) and is given by cubic polynomials defined on each of the intervals $[x_i, x_{i+1}]$, i = 0,..., n-1. To determine the coefficients of this spline, one must solve an (n-1)x(n-1) tridiagonal linear system of equations. This spline, though historically significant, is not very useful in computer graphics. A unique spline does not exist if there is a repeated x value and if a data point needs to be changed then the entire spline will need to be recomputed.

To overcome these problems splines that do not necessarily interpolate the given data are used. The most often used splines are the Bézier (BEZI70 and BEZI74) and uniform B-splines (CURR47). Splines can be defined for any degree though the cubic splines are usually sufficiently robust to handle the smoothness requirements in graphics.

The uniform cubic B-spline, b, stretches over five data points, called knots. The distances between the knots are all taken as 1 and the point 0 is put in the middle of these five knots to achieve symmetry. Formally this is given by:

$$b(u) = \begin{cases} 0 & if\ u \le -2 \\ \frac{(2+u)^3}{6} & if\ -2 \le u \le -1 \\ \frac{(2+u)^3}{6} - \frac{2(1+u)^3}{3} & if\ -1 \le u \le 0 \\ -\frac{2(1-u)^3}{3} + \frac{(2-u)^3}{6} & if\ 0 \le u \le 1 \\ \frac{(2-u)^3}{6} & if\ 1 \le u \le 2 \\ 0 & if\ 2 \le u \end{cases}$$

If we need to draw a curve based on knots $P_0 = (x_0, y_0, z_0)$, ... , $P_n = (x_n, y_n, z_n)$, then one way to construct the approximating curve is to form the linear combination

$$C(u) = \sum_{i=0}^{n} p_i b_i(u)$$

where p_i stands for x_i, y_i, or z_i. The most convenient way to draw the B-spline curve is to use the following formula. This form expresses the value of C in any subinterval $[P_i, P_{i+1}]$ with i not 0 nor n-1.

$$C(u) = \frac{1}{6}\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix}$$

As the parameter u changes from 0 to 1 the formula generates the curve from P_i to P_{i+1}.

The other most commonly used cubic spline is the Bézier spline. The main difference between the two is that the Bézier curve is globally, not locally, controlled by the points P_i. Changing one of the control points affects the entire Bézier curve, but its strongest affect in is the neighborhood of the point. The matrix formulation of the Bézier curve is given below:

$$C(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 3 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

To generate spline surfaces, one simply takes the Cartesian product of two splines. For example,

we define a B-spline surface b(u,v) = b(u)b(v). The matrix formulation of this surface is given by

$$S(u,v) = \frac{1}{36} UMPM^T V^T$$

where

$$U = (u^3, u^2, u, 1),\ V = (v^3, v^2, v, 1) \text{ and}$$

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \text{ and } P = \begin{bmatrix} p_{i-1,j-1} & p_{i-1,j} & p_{i-1,j+1} & p_{i-1,j+2} \\ p_{i,j-1} & p_{i,j} & p_{i,j+1} & p_{i,j+2} \\ p_{i+1,j-1} & p_{i+1,j} & p_{i+1,j+1} & p_{i+1,j+2} \\ p_{i+2,j-1} & p_{i+2,j} & p_{i+2,j+1} & p_{i+2,j+2} \end{bmatrix}$$

A similar formulation for Bézier surfaces exists.

There are, of course, other splines and generalizations that can be used to model curves or surfaces, e.g., Hermite, non-uniform B-splines, and Beta splines. The interested reader is referred to BART87.

Section 5: Other Neat Stuff

We have only begun to scratch the surface of the different types of mathematics used in computer graphics, and we have omitted many of the details of the concepts presented in this paper. For example, most of the calculus concepts involved in the illumination section were omitted. However, it should be fairly obvious that geometry, linear algebra and calculus play fundamental roles in the representation and manipulation of images. Other aspects of computer graphics utilize often more complicated mathematics. For example, antialiasing uses digital signal processing and hence Fourier transforms, and physically based modeling requires the solution of partial differential equations. To truly understand fractals, complex arithmetic and a little analysis will go a long way. In order to fully appreciate and understand the finer aspects of computer graphics one must at least appreciate, if not understand, the finer aspects of mathematics, as you would not have one without the other.

References:

BART87 Bartels, R. H., J. C. Beatty, B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Inc. 1987.

BEZI70 Bézier, P., *Emploi des Machines á Commande Numérique*, Masson et Cie, Paris, 1970. Translated by Forrest, A. R., and A. F. Pankhurst as Bézier, P., *Numerical Control -- Mathematics and Applications*, Wiley, London, 1972.

BEZI74 Bézier, P., "Mathematical and Practical Possibilities of UNISURF," in Barnhill, R. E., and R. F. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, New York, 1974.

BRES65 Bresenham, J. E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal*, 4(1), 1965, 25-30.

BRES77 Bresenham, J. E., "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *Communications of the ACM*, 20(2), February 1977, 100-106.

CURR47 Curry, H. B., and I. J., Schoenberg, "On Spline Distributions and Their Limits: the Polya Distribution Functions," *Bull. American Mathematical Society*, 53, Abstract 380t (1947), 109.

FOLE90 Foley, J. D., A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Edition, Addison-Wesley Publishing Company, Inc., 1990.

GOUR71 Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. on Computers*, C-20(6), June 1971, 623-629.

HART67 Hartshorne, R., *Foundations of Projective Geometry*, Benjamin/Cummings Publishing Company, 1967.

NISH85 Nishita, T., I. Okamura, and E. Nakamae, "Shading Models for Point and Linear Sources," *ACM TOG*, 4(2), April 1985, 124-146.

PHON75 Phong, B.-T., "Illumination for Computer Generated Images," *Communications of the ACM*, 18(6), June 1975, 311-317.

PITT67 Pitteway, M. L. V., "Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter," *Computer J.*, 10(3), November 1967, 282-289.

POKO89 Pokorny, C., C. F. Gerald, *Computer Graphics: The Principles Behind the Art and Science*, Franklin, Beedle & Associates, 1989.

VANA84 Van Aken, J. R., "An Efficient Ellipse-Drawing Algorithm," CG&A, 4(9), September 1984, 24-35.

VERB84 Verbeck, C. P., and D. P. Greenberg, "A Comprehensive Light-Source Description for Computer Graphics," *CG & A*, 4(7), July 1984, 66-75.

WARN83 Warn, D. R., "Lighting controls for Synthetic Images," *SIGGRAPH 83*, 13-21.

WYLI64 Wylie, C. R. Jr., *Foundations of Geometry*, McGraw-Hill Book Company, 1964.