Computer Graphics and Geometry

Jim Emery

9/8/2007

Contents

Chapter 1

Simple Graphics

1.1 Draws and Moves

A curve can be approximated as a sequence of small line segments. Hence most line art can be represented as such sequences. The line segments can be accomplished with simple draws and moves. Smooth curves are drawn simply by joining pairs of points with small line segments. The curve will look smooth even when the segments are relatively large. For example a circle constructed with from 50 to 100 line segments will look quite smooth.

EGraphics (E as in Emery) is a device independent graphics system. It supports devices such as HPGL plotters, Postscript printers, the CGA, EGA, and VGA monitors, the Apollo workstations (GPR driver), X Windows, and Silicon Graphics workstations (GL driver). By using public domain programs such as **Printgl**, and **Ghostscript**, or by importing to programs like Microsoft Word, one can plot to most devices. The EGraphics file format consist of one letter or two letter commands, which are followed by parameters. The system supports: lines, arcs, Bezier curves, text, pen color, and shaded polygons (postscript). There are user subroutines, which may be called from C or FORTRAN and which write EGraphics files. The EGraphics commands are very simple and can be created directly from a program using simple print statements. Here is an EGraphics file for drawing a rectangle:

```
v-1 1 -1 1
w0. 10. 0. 10.
m1.0 1.0
d9 1
```

```
d9 9
d1 9
d1 1
```

The letter **v** stands for viewport. It specifies a region of a raster screen, or a region of some graphics device such as a plotter or a laser printer. The letter **w** stands for window. It specifies user space, sometimes called world space. These two definitions define a mapping from the world rectangle to the viewport rectangle. There is a second transformation embedded in the driver programs that maps to device coordinates. For example, if the device is a PC VGA monitor, the pixel values are numbered from (480,0) at the lower left corner of the screen, to (0,640) at the upper right. The viewport parameters (xmn,xmx,ymn,ymx) = (-1,1,-1,1) specify the maximum square screen region, or the maximum square plotter region. For example, if the viewport parameters were (0,1,0,1), then the world rectangle,$(xmn, xmx, ymn, ymx) = (0, 10, 0, 10)$, would be mapped to a pixel region on a PC screen with lower left corner (240,320), and upper right corner (0,640). On a plotting device the rectangle would be mapped to the upper right quadrant of the plotting region. The transformations from the user world space to the device space are handled by the device drivers such as **pltps.c**, **pltvga.c**, and **pltgpr.ftn**. The letter **m** means move, and The letter **d** means draw. Other such commands will be discussed later. On a PC, one can plot this file, which we assume is named p.gi, with the command:

```
pltvga p.gi
```

Other programs exist for plotting on other devices. For example to make a postscript file called p.ps, the command is

```
pltps p.gi p.ps
```

To make an HPGL plot, the command is

```
plthp p.gi p.hgl
```

To make a plot on Silicon Graphics:

```
pltgl p.gi
```

See the section **EGraphic** for more information.

1.2 Straight Lines

The equation of a straight line in the plane may be written as

$$q_1 x + q_2 y + q_3 = 0.$$

We may call q_1, q_2, q_3 the coordinates of the line. If we call $p_1 = x, p_2 = y, p_3 = 1$, the coordinates of a point that lies on the line, then the equation becomes

$$q_1 p_1 + q_2 p_2 + q_3 p_3 = 0.$$

We may write this as the inner product of the line coordinate vector with the point coordinate vector ,

$$q \cdot p = 0.$$

If we multiply q or p by a constant, nothing really changes, we have the same equation. Multiplying an equation by a number does not change the set of solutions of the equation. q and p are called the homogeneous coordinates of the line and the point respectively. If we consider p and q to be three dimensional vectors, then point p lies on line q if vector p is perpendicular to q, that is , if the inner product of p and q is zero. Suppose we are given two lines q_1 and q_2, and wish to find their intersection point p. Then vector p must be perpendicular to both q_1 and q_2. Hence p is equal to the cross product of q_1 with q_2,

$$p = q_1 \times q_2.$$

1.3 Circles And Arcs

A circular arc may be defined in many ways. An arc is defined by three noncollinear points. An arc may be defined by a center, a radius, and a start angle and an end angle. An arc may be defined by a start point, an end point, and a center (provided the center is equidistant from the start and end points). An arc may be defined by a start point p_s, a center c, and an arc angle θ_a. Let $p_s = (x_s, y_s)$ and $c = (x_c, y_c)$. Let R be a vector from the center to the start point. Then

$$R = (x_R, y_R),$$

where

$$x_R = x_1 - x_c$$

and

$$y_R = y_1 - y_c.$$

Then the circle radius is

$$r = \|R\| = \sqrt{x_R^2 + y_R^2}.$$

The start angle θ_s is the angle of R, which is called the argument of R. If function A is the **atan2** function of FORTRAN or C, then

$$\theta_s = A(y_R, x_R).$$

The end angle is

$$\theta_e = \theta_s + \theta_a.$$

The arc is traced out by the point

$$p = (x, y),$$

where

$$x = x_s + r\cos(\theta),$$

and

$$y = y_s + r\sin(\theta),$$

as angle θ varies from θ_s to θ_e. Suppose we want n uniformly spaced angles between θ_s and θ_e . These are given by

$$\theta_i = \frac{(i-1)(\theta_e - \theta_s)}{n-1} + \theta_s,$$

for $i = 1....n$.

The angles must be in radians. The conversion is

$$\theta_{rad} = \frac{\pi}{180}\theta_{deg}.$$

The arc is displayed by drawing straight lines between the points. For a complete circle, when we use $n = 100$ to $n = 200$ points, we obtain a smooth appearing curve. Since there are 2π radians in a circle, the corresponding number of points for a partial arc are

$$n = 100\frac{\theta_a}{2\pi},$$

and

$$n = 200\frac{\theta_a}{2\pi}.$$

Chapter 2

Geometry

2.1 Line Intersections

The equation of a line is of the form

$$ax + by + cz = 0$$

The coordinates of the line

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

are perpendicular to the coordinates of the point

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

If A and B are coordinate vectors of two lines then the cross product is the coordinate vector of the intersection point because it is perpendicular to both. In the same way if X and Y are coordinate vectors of two points then their cross product is the coordinate vector of the line containing both. This is an example of duality in projective space.

Example. Find the intersection point of the two lines with equations

$$x + y = 0,$$

and

$$2x + y = 1.$$

The cross product is

$$\begin{bmatrix} 2 \\ -5 \\ -1 \end{bmatrix}$$

So the affine coordinates of the intersection point are $(-2, 5)$.

Example. Find the line through the points

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

and

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

The cross product is

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

So the equation is

$$-x + 2y = 1.$$

Example. Find the line passing through $(1, 1)$ and making an angle of 30 degrees with the x-axis. The line passes through the point at infinity given by

$$\begin{bmatrix} \cos(30) \\ \sin(30) \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 \\ 1/2 \\ 0 \end{bmatrix},$$

and the point

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

Taking the cross product, we find the line to be

$$x - \sqrt{3}y = 1 - \sqrt{3}.$$

2.2 Transformations

2.2.1 Rotations

The matrix of a rotation is an orthogonal matrix. The column vectors of an orthogonal matrix are perpendicular, that is their dot products are zero. As a consequence, the transpose is the inverse. A proper rotation matrix has a determinant equal to one. The eigenvector corresponding to eigenvalue one is the axis of the rotation. The rotation is an isometry and preserves distance and hence the inner product. Thus

$$Rv \cdot Rv = v \cdot v.$$

Applying this to the unit coordinate vectors, one sees that the column vectors of R are orthogonal, and are unit vectors. Given a rotation matrix R one may find the rotation angle by computing the trace of the matrix. The trace is the sum of the elements on the main diagonal. In the case of rotation about the z axis, the matrix is

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So the trace is

$$1 + 2\cos(\theta).$$

The trace is invariant to a change of coordinate system, so the result is general.

Consider the inverse problem of finding the rotation matrix R corresponding to a rotation axis ℓ and a rotation angle θ. A three dimensional antisymmetric tensor may be represented as a cartesian vector, and so the tensor is called an axial vector. That is, if ℓ is a vector, then there is a matrix L so that for every vector v

$$Lv = \ell \times v.$$

Axial vectors appear in the concepts of angular velocity and angular momentum. Corresponding to the rotation axis ℓ, is the matrix (tensor)

$$L = \begin{bmatrix} 0 & \ell_3 & -\ell_2 \\ -\ell_3 & 0 & \ell_1 \\ \ell_2 & -\ell_1 & 0 \end{bmatrix}$$

Jay Fillmore computes

$$\exp(tL),$$

as an infinite series, and then shows that the rotation matrix is

$$R = I + \sin(\theta)L - (1 - \cos(\theta))L^2,$$

when the rotation axis ℓ is a unit vector. (See Fillmore Jay P., **A Note On Rotation Matrices**, IEEE Computer Graphics and Applications, February 1984, pp30-33.)

Example Let the rotation axis be the x axis, and let the unit vector along the rotation axis be $\ell = i$, where i is the unit coordinate vector. Then

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

and

$$L^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Then

$$\begin{aligned} R &= I + \sin(\theta)L - (1 - \cos(\theta))L^2 \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \end{aligned}$$

2.2.2 Transformations of Quadric Surfaces

A one to one linear transformation maps a quadric surface to a new quadric surface. Suppose

$$S = \{P : f(P, P) \leq 0\}$$

then

$$\begin{aligned} LS &= \{LP : f(P, P) \leq 0\} \\ &= \{Q : f(L^{-1}Q, L^{-1}Q) \leq 0\} \\ &= \{Q : (L^{-1}Q, AL^{-1}Q) \leq 0\} \\ &= \{Q : (Q, L^{-1T}AL^{-1}Q) <= 0\} \end{aligned}$$

$$= \{Q : f^*(Q, Q) \leq 0\},$$

where f^* is a bilinear form with matrix

$$L^{-1T} A L^{-1} = A'$$

The matrices for the scaling, translating and rotation transformations are given here.

(1) Scaling matrix:

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse of scaling matrix:

$$\begin{bmatrix} 1/a & 0 & 0 & 0 \\ 0 & 1/b & 0 & 0 \\ 0 & 0 & 1/c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(2) Affine rotation: Let $c = \cos(\theta)$ and $s = \sin(\theta)$. The sense of the rotations about the unit coordinate vectors is given by the right hand rule. With the thumb of the right hand extended in the direction of the vector, positive angle is the direction in which the fingers curl while grasping the vector.

Rotation about the x-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about the y-axis:

$$\begin{bmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about the z-axis:

$$\begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now if R is a rotation matrix, then

$$R^{-1}(\theta) = R(-\theta).$$

An orthogonal matrix R has the property $R^T = R^{-1}$, so the matrix of the transformed quadric surface is

$$A' = R(\theta)AR(-\theta).$$

(3)Affine translation: The matrix

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 0 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translates by

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

The inverse of T is

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 0 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The transformed matrix is

$$A' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix} A \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Example. Given a sphere with matrix

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

Then the scaling transformation matrix

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

Maps the sphere to an ellipse with matrix

$$\begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

The translation matrix T shifts it by 2

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

The matrix of the shifted conic becomes

$$(T^{-1})^T (S^{-1})^T A S^{-1} T^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/4 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 0 \end{bmatrix}.$$

The equation of the shifted ellipsoid is

$$x^2/4 + y^2 + z^2 + 1/2xw + 1/2wx = 0.$$

Suppose we rotate about the z-axis by angle 45 degrees. The rotation matrix is

$$R_z(45) = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The matrix of the quadric becomes

$$R_z(45)(T^{-1})^T(S^{-1})^T A S^{-1} T^{-1} R_z(-45).$$

This is

$$\begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{2}/8 & \sqrt{2}/8 & 0 & 1/2 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sqrt{2}/4 & \sqrt{2}/4 & 0 & 0 \end{bmatrix}.$$

The equation of the shifted rotated quadric is

$$(5/8)x^2 + (5/8)y^2 - (3/4)xy + z^2 + xw\sqrt{2}/2 + yw\sqrt{2}/2 = 0.$$

2.3 Vector Analysis

2.3.1 The Inner Product

We shall prove the law of cosines. Suppose we have three points

$$p_0 = (0,0), p_1 = (b,0), p_2 = (x,y) = (a\cos(\theta), a\sin(\theta)).$$

These points form a triangle with sides p_0p_2, p_0p_1, p_2p_1. These sides have lengths a, b, c. The angle between side p_0p_1 and side p_0p_2 is θ. We have

$$\begin{aligned} c^2 &= (x-b)^2 + y^2 \\ &= (x-b)^2 + a^2 - x^2 \\ &= x^2 - 2xb + b^2 + a^2 - x^2 \\ &= a^2 + b^2 - 2xb = a^2 + b^2 - 2ab\cos(\theta). \end{aligned}$$

Thus we have the law of cosines, namely the square of the side opposite an angle of a triangle, is equal to the sum of the squares of the adjacent sides, minus two times the product of the sides and the cosine of the angle. That is,

$$c^2 = a^2 + b^2 - 2ab\cos(\theta).$$

The inner product (dot product) of two vectors, A and B, is defined as

$$A \cdot B = a_1b_1 + a_2b_2 + a_3b_3.$$

Then the dot product of a vector with itself is the square of its length. That is,

$$A \cdot A = a_1a_1 + a_2a_2 + a_3a_3 = \|A\|^2.$$

Let

$$C = B - A.$$

Then

$$\begin{aligned} \|C\|^2 &= (B-A)\cdot(B-A) \\ &= B\cdot B - B\cdot A - A\cdot B + A\cdot A \\ &= \|B\|^2 - 2A\cdot B + \|A\|^2. \end{aligned}$$

From which it follows that

$$2A\cdot B = \|A\|^2 + \|B\|^2 - \|C\|^2.$$

But the right hand side is, by the law of cosines,

$$2\|A\|\|B\|\cos(\theta),$$

where θ is the angle between vectors A and B. Hence

$$A \cdot B = \|A\|\|B\|\cos(\theta).$$

Thus if the dot product is zero, then the cosine is zero, and so the angle between the vectors is plus or minus $\pi/2$, and the vectors are perpendicular.

2.3.2 The Vector Product

The vector product of two vectors A and B, (the cross product), is defined to be

$$A \times B = (a_2b_3 - a_3b_2)i + (a_3b_1 - a_1b_3)j + (a_1b_2 - a_2b_1)k,$$

where i, j, k are the unit coordinate vectors. This may be written as a determinant with i, j, k in the first row, the components of A in the second, and the components of B in the third row. When the rows of a determinant are interchanged, the sign of the determinant changes, hence

$$A \times B = -B \times A.$$

Then

$$A \times A = -A \times A.$$

But this can be true only if

$$A \times A = 0.$$

We have shown that the vector product of any two parallel vectors is zero.

Given three vectors A, B, C, we see that

$$A \cdot (B \times C),$$

is given as the determinant that has rows A, B, and C. By interchanging these rows twice, we see that

$$A \cdot (B \times C) = (A \times B) \cdot C.$$

That is, in the scalar triple product, the dot and the cross may be interchanged. Now using this result, we see that

$$(A \times B) \cdot B = A \cdot (B \times B) = A \cdot 0 = 0.$$

Then $A \times B$ is perpendicular to B. Similarly it is perpendicular to A. Therefore we have shown that the vector product of two vectors is perpendicular to each of them. This establishes the direction of the vector product, except possibly for sign. One may further establish the right hand rule. The direction of $A \times B$ is given by the right hand rule: Curl the fingers of your right hand from A to B, then $A \times B$ is in the direction of your thumb. One may verify directly that if V is a vector in the upper xy half plane that

$$i \times V$$

points in the positive z direction. This verifies the right hand rule in this case. One may also show the invariance of the cross product to a rigid motion, which establishes the right hand rule in general.

By direct computation one may verify that the vector triple product satisfies

$$A \times (B \times C) = B(A \cdot C) - C(A \cdot B).$$

This is the "Back Minus Cab Rule". We have established the direction of the cross product, now we shall find its magnitude. Let

$$C = A \times B.$$

Then

$$\begin{aligned} \|C\|^2 &= C \cdot C \\ &= (A \times B) \cdot C \\ &= A \cdot (B \times C) \\ &= A \cdot (B \times (A \times B)) \\ &= A \cdot (A(B \cdot B) - B(B \cdot A) \\ &= (A \cdot A)(B \cdot B) - (A \cdot B)^2 \\ &= \|A\|^2\|B\|^2(1 - \cos^2(\theta) = \|A\|^2\|B\|^2 \sin^2(\theta). \end{aligned}$$

Thus the magnitude of the cross product is the product of the lengths of the vectors, times the sine of the angle between them,

$$\|A \times B\| = \|A\|\|B\| \sin(\theta).$$

Example The equation of a plane. Let the plane have a unit normal vector N. Let $P = (x, y, z)$ be a point on the plane. Let d be the distance from the origin to the plane. Then d is equal to the length of P times the cosine of the angle between P and the normal N. Hence

$$d = P \cdot N.$$

Therefore the equation of the plane is

$$P \cdot N - d = xn_1 + yn_2 + zn_3 - d = 0.$$

Suppose we are given three points P_1, P_2, P_3 and we wish to find the equation of the plane passing through these points. The normal to the plane is perpendicular to each of $P_2 - P_1$ and $P_3 - P_1$. Therefore

$$N = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\|(P_2 - P_1) \times (P_3 - P_1)\|}$$

Also d is equal to the inner product of N with any one of the three points. For example

$$d = P_1 \cdot N.$$

Then the equation of the plane is

$$P \cdot N - P_1 \cdot N = xn_1 + yn_2 + zn_3 - d = 0.$$

2.4 Some Geometric Calculations

2.4.1 The Distance Between Two Lines in Space

Let the lines be given by the equations

$$L_1(t) = A_1 + B_1 t,$$

and

$$L_2(s) = A_2 + B_2 s.$$

Let $V(t,s)$ be a vector from L_2 to L_1. Let

$$V(t,s) = L_1(t) - L_2(s).$$

If V is to give the shortest distance between the lines, then it must be perpendicular to both B_1 and B_2. Hence

$$0 = B_1 \cdot V = (B_1 \cdot B_1)t - (B_1 \cdot B_2)s + B_1 \cdot (A_1 - A_2),$$

and

$$0 = B_2 \cdot V = (B_2 \cdot B_1)t - (B_2 \cdot B_2)s + B_2 \cdot (A_1 - A_2).$$

We will solve these equations for t and s. Let

$$p = \begin{bmatrix} (B_1 \cdot B_1) \\ -(B_1 \cdot B_2) \\ B_1 \cdot (A_1 - A_2) \end{bmatrix}$$

and

$$q = \begin{bmatrix} (B_2 \cdot B_1) \\ -(B_2 \cdot B_2) \\ B_2 \cdot (A_1 - A_2) \end{bmatrix}.$$

Let

$$u = \begin{bmatrix} t \\ s \\ 1 \end{bmatrix}.$$

Then the pair of equations defining t and s may be written as

$$p \cdot u = p_1 t + p_2 s + p_3 = 0,$$

and

$$q \cdot u = q_1 t + q_2 s + q_3 = 0.$$

Vector u is perpendicular to both vectors p and q, and u has its third coordinate equal to one. Let

$$r = p \times q.$$

Then r is perpendicular to both p and q. Then u is

$$u = r/r_3,$$

provided r_3 is not zero. In that case

$$t = r_1/r_3,$$

and

$$s = r_2/r_3.$$

If $r_3 = 0$, then the lines are parallel and there exists an infinite number of solutions t and s. In this case, because by assumption

$$p_1 = \|B_1\|^2$$

is not zero, we set $s = 1$ and compute

$$t = \frac{p_2 + p_3}{-p_1}.$$

2.4.2 The Bilinear Mapping And Its Inverse

Given a quadrilateral with vertices p_1, p_2, p_3, p_4, the bilinear mapping from the unit square to the quadrilateral is defined by

$$q_1(u) = (1-u)p_1 + p_2 u$$

$$q_2(u) = (1-u)p_4 + p_3 u$$

$$f(u,v) = (1-v)q_1(u) + q_2 v$$

$$= p_1 + (p_2 - p_1)u + (p_4 - p_1)v + (p_1 - p_2 - p_4 + p_3)uv.$$

Let us find the inverse of f. Let

$$a_1 = p_1$$

$$a_2 = p_2 - p_1$$

$$a_3 = p_4 - p_1$$

$$a_4 = p_1 - p_2 - p_4 + p_3$$

where

$$a_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

Then

$$a_1 + a_2 u + a_3 v + a_4 uv = f.$$

Then we have a vector equation

$$u(a_2 + a_4 v) = f - a_1 - a_3 v,$$

and two corresponding scalar equations

$$u(x_2 + x_4 v) = x - x_1 - x_3 v,$$

and

$$u(y_2 + y_4 v) = y - y_1 - y_3 v.$$

From these equations we get a quadratic equation for v

$$(x - x_1 - x_3 v)(y_2 + y_4 v) - (y - y_1 - y_3 v)(x_2 + x_4 v) = 0.$$

The quadratic is

$$av^2 + bv + c = 0,$$

where

$$a = x_3 y_4 + y_3 x_4,$$

$$b = -xy_4 + x_1 y_4 + y_1 x_4 - y x_4 - x_3 y_2 + y_3 x_2,$$

$$c = x y_2 - x_1 y_2 + y_1 x_2 - y x_2.$$

See the Maple program min20. From the geometry of the problem, there will be exactly one root v with $0 \leq v \leq 1$. Once we find the value of v, we may solve for u. Hence we have computed the inverse mapping f^{-1},

$$(u, v) = f^{-1}(x, y).$$

Now suppose we are given two quadrilaterals. Let f_1 be a bilinear function, mapping the unit square to the first quadrilateral. And let f_2 be a bilinear function, mapping the unit square to the second quadrilateral. Then $f_2 f_1^{-1}$ maps the first quadrilateral to the second.

2.4.3 The Intersection of Three Nonparallel Planes

Given three planes with linearly independent normal vectors, n_1, n_2, n_3, then the unique intersection point p

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$

is given by

$$p = \frac{r_1(n_2 \times n_3) + r_2(n_3 \times n_1) + r_3(n_1 \times n_2)}{n_1 \cdot n_2 \times n_3},$$

where the equations of the planes are

$$n_1 \cdot p = r_1,$$

$$n_2 \cdot p = r_2,$$

and

$$n_3 \cdot p = r_3.$$

To prove this we shall use a reciprocal basis. Given a basis $v_1, v_2, v_3, ..., v_n$,of a vector space (inner product space), a set of vectors $u_1, ..., u_n$, that satisfy the property that

$$u_i \cdot v_j = \delta_{i,j},$$

are called a reciprocal basis of the vector space. Using this property, it is easy to show that the vectors $u_1, ...u_n$, are in fact linearly independent. From the properties of the dot and cross product, one sees that the vectors

$$u_1 = \frac{(n_2 \times n_3)}{n_1 \cdot n_2 \times n_3},$$

$$u_2 = \frac{(n_3 \times n_1)}{n_1 \cdot n_2 \times n_3},$$

$$u_3 = \frac{(n_1 \times n_2)}{n_1 \cdot n_2 \times n_3}.$$

are reciprocal to n_1, n_2, n_3. So there exists numbers a_1, a_2, a_3 so that

$$p = a_1 u_1 + a_2 u_2 + a_3 u_3.$$

Now dotting with n_i, we find that

$$a_i = n_i \cdot p = r_i.$$

This completes the proof.

Note that if p_i is any point on plane i, then $r_i = n_i \cdot p_i$,. So that defining the planes by a point on the plane and a normal vector, gives the information needed for the computation.

2.5 Projective Geometry

2.5.1 Projective Space

Consider the equation of a line

$$L_1 x + L_2 y + L_3 = 0.$$

Introduce a third coordinate w, and define

$$p_1 = xw, p_2 = yw, p_3 = w.$$

Multiplying the equation by w, and using the dot product, it becomes

$$L \cdot p = 0.$$

The components of p are called homogeneous coordinates. They are determined up to a scalar multiple. If $p_3 = w$ is not zero, we may divide by w and thus make the third coordinate equal 1. That is frequently we will have $w = 1$. Let us find the line L that passes through the points e and s. Since the points must lie on the line ,we have two equations that must be satisfied

$$L \cdot e = 0,$$

and

$$L \cdot s = 0.$$

A solution to these equations is $L = e \times s$. This is true because the cross product of two vectors is perpendicular to each vector. Consider the equations of two parallel lines

$$a_1 x + a_2 y + a_3 = 0$$

and

$$a_1 x + a_2 y + b_3 = 0,$$

where a_3 and b_3 are not equal. Let us write these equations with homogeneous coordinate vector x.

$$a_1 x_1 + a_2 x_2 + a_3 x_3 = 0$$

$$a_1 x_1 + a_2 x_2 + b_3 x_3 = 0.$$

These lines intersect at infinity. Subtracting one from the other ,we have $(a_3 - b_3)x_3 = 0$, which implies $x3 = 0$. This indicates that when the third

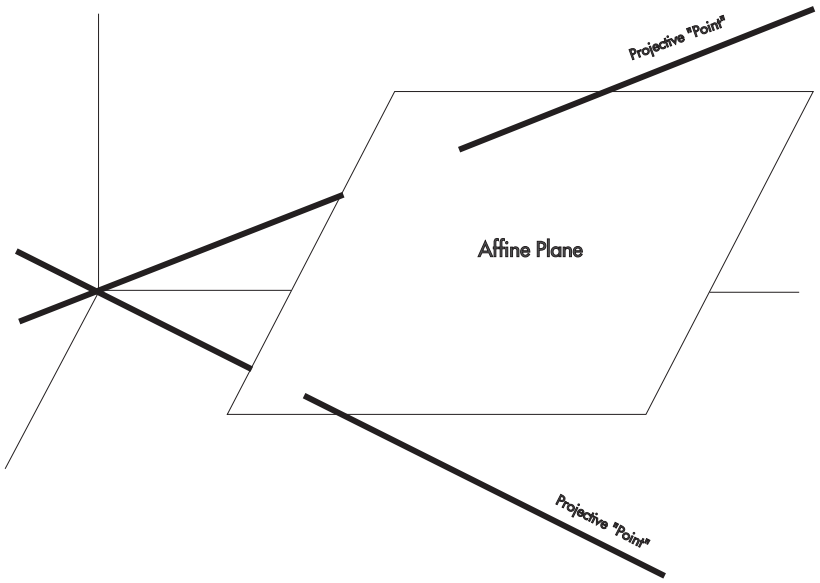

Figure 2.1: The Affine Plane and Projective Space.

coordinate of a point is 0, the point is at infinity. Let us consider again the line $L = e \times s$,which passes through e and s. If we subtract s from e, we obtain a point on L at infinity. This follows because

$$L \cdot (e - s) = e \times s \cdot (e - s) = 0 - 0 = 0.$$

So the point is on L. We assume e and s have the same third coordinate, so that $e - s$ is at infinity. The vector $e - s$ has the direction of L.

We have given an algebraic motivation for projective geometry. We will now give a geometric motivation, and then a definition of projective space.

Suppose we have a cone with vertex at the origin. Let there be a plane $z = 1$. The intersection of the cone with the plane is a curve, which depending on the orientation of the cone will be an ellipse, a parabola, or a hyperbola. We may identify the lines on the cone with points on the plane. The plane is called the affine plane. The parabola and hyperbola have points at infinity, an ellipse or a circle, which is a special case of an ellipse, does not have infinite points. In the figure showing projective points and the affine plane, think of the two dark lines passing through the origin as lying on the cone. They pierce the affine plane as shown. When either of these lines move so as to become parallel to the plane, the corresponding intersection point will

go to infinity. This means that points at infinity on the plane correspond to lines that are parallel to the affine plane. A parabola results when only one line on the cone does not intersect the affine plane $z = 1$. This occurs when the cone is resting on the $z = 0$ plane. If the $z = 0$ plane intersects the cone rather than being tangent as in the case of the parabola, then there will be two intersection lines, that is two points at infinity. This is the case of the hyperbola. If no lines on the cone are in the $z = 0$ plane then we get the case of an ellipse. We can represent all of the points on a conic section by lines through the origin, even the infinite ones. Two dimensional projective space is a set of lines through the origin. Lines through the origin are really 1-dimensional subspaces, that is they are the collection of all the vectors that point in the direction of such a line. Homogeneous coordinates are the coordinates of any vector in the direction of the line. Multiplying them by a scaler gives the coordinates of another vector in the direction of the line. We have used the $z = 1$ plane as the affine plane and this is convenient, but using any other plane that does not pass through origin could be used. What we have called points at infinity, depend on which plane we select. So points at infinity are not really identifiable in projective space itself.

We can use these techniques to do certain simple calculations in an efficient manner. For example suppose we want the perpendicular bisector of the segment joining two points e and s. The perpendicular bisector of the segment es is

$$M = rot(e - s) \times (e + s)/2,$$

where rot rotates a vector by a positive 90 degrees. We have constructed M as the line through the point at infinity $e - s$, and through the point, which is the midpoint of the segment es. The midpoint is $(e + s)/2$.

For a second example, suppose again we have two points e and s. They define a line L. Suppose we have a point d not on L. Let N be the line parallel to L and through d. Then

$$N = d \times (e - s).$$

Higher dimensional projective spaces is defined in the same way as the two dimensional projective space treated here. An n-dimensional projective space is the set of all one- dimensional subspaces (lines through the origin) in some $n + 1$ dimensional vector space. We have used three dimensional real Euclidean space as the vector space, in which to define our two-dimensional

projective space. So a point in two dimensional real projective space is the set all 3d vectors having a fixed direction.

Rational polynomials come about as affine representations of homogeneous polynomials. The canonical conic is

$$r = (t_2, t, 1) = (x, y, z),$$

This is a parabola:

$$x = t_2 = y_2.$$

A general conic is then

$$r' = P(r) = (a_1 t_2 + a_2 t + a_3, b_1 t_2 + b_2 t + b_3, c_1 t_2 + c_2 t + c_3),$$

where P is a projective transformation. The transformations we need are special projective transformations. We need only a rotation and a scaling. Suppose we have a conic section curve. It is the intersection of some cone and the affine plane. Rotate the cone so that its axis is parallel to the z-axis. The new intersection curve is a circle. Scale the x and y coordinates so that the circle is the unit circle (radius = 1). We have shown that any conic section can be mapped to the unit circle by a projective transformation. This can be reversed and a unit circle mapped to any conic. Finally in two such steps any conic can be mapped to any other conic. In the process finite points can be mapped to infinite points and vice versa. A projective transformation is any nonsingular linear transformation in the underlying vector space considered to operate on the projective points (the one-dimensional subspaces). A special projective transformation that takes finite points to finite points and infinite points to infinite points is called an affine transformation. All points that are finite are related. Affine is derived from words meaning "related to". Thus the affine plane is the set of related or finite points. A projective transformation being essentially a linear transformation can be represented by a matrix. The mapping of a homogeneous coordinate vector can be accomplished by matrix multiplication. Mathematicians tend to use column vectors for points with transformation of points corresponding to multiplication on the left. This is done because it corresponds to the traditional way of writing functions. Computer scientists being somewhat ignorant prefer row vectors (they take up less space on the printed page). Row vectors are established in computer graphics representation of homogeneous coordinates. The programs here listed use column vectors.

2.6 Differential Geometry

Differential Geometry is the study of the variation of the shape of curves and surfaces. It deals with concepts such as curvature, tangent, normal, arclength, and surface shape.

2.7 Algebraic Geometry

Algebraic Geometry is the study of geometric properties of algebraic and polynomial equations in two or more variables. It is useful for problems involving curve and surface intersections, and the conversion of parametric representations into algebraic representations.

2.8 Splines

Splines curves and surfaces are constructed from pieces of polynomials, usually in such a manner as to give smooth curves and surfaces. The name spline derives from the name of the thin piece of wood that was used as a drawing instrument to produce curved lines through sets of points. A deflected thin beam, whose small deflection is due to point loads, is a spline curve. The cubic spline function is a piecewise 3rd degree polynomial curve. The polynomial pieces are joined at the knot points in such a manner as to make the resulting function continuous, to have continuous first derivative (slope), and to have continuous second derivative (This implies continuous curvature). In general the third derivative is discontinuous. There is a simple algorithm for constructing a cubic spline that passes through a given set of interpolation points.

There is a class of functions that serves well as a basis for a vector space of splines (the concept of splines is generalized to include all piecewise polynomials, whether the pieces are smoothly joined or not). These are the B-splines (Basic splines). Every element of a spline space may thus be represented as a linear combination of B-splines. This is in the same way that every periodic function may be represented as a Fourier series (under certain conditions), that is, as a linear combination (perhaps infinite) of Sines and Cosines.

The theory of B-splines is extensive and rather complex. However, B-splines are easy to compute from a recursive formula. We shall give the formal definition of B-splines, which requires an understanding of divided

differences, but we shall not go into great detail. Let T_x be the truncated power function of order k centered at x.

$$T_x(t) = (x-t)_+^{k-1}.$$

Let $\tau = \{t_i\}$ be a sequence of nondecreasing points on the real line. The value of the ith B-Spline of order k at x is

$$B_{i,k,\tau}(x) = (-1)^k (t_{i+k} - t_i) G_{\{t_i,..t_{i+k}\}} T_x$$

when $t_{i+k} > t_i$. If $t_{i+k} = t_i$, then $B_{i,k,\tau}(x)$ is zero. G is the divided difference operator.

Properties: (1) Each B-Spline has support in a finite interval, (that is, it is nonzero in only a finite interval). (2) B-Splines form a partition of unity, (they add up to 1 at each point of the domain) (3) Each B-Spline is a piecewise polynomial of order k, (4) They and their derivatives can be computed recursively, (5) Given a space $P_{k,\xi,\nu}$ of piecewise polynomials, there exists a knot sequence τ so that the B-Splines are a basis of the space. (6) If the knot set consists of distinct points, then each B-Spline is a linear combination of shifted truncated power functions of the form $T_x^k(t_i)$, and thus is a spline in the traditional sense.

Let us consider an example. Let us construct the B-Spline of order 1 on the knots t_0 and t_1. From the definition

$$B_{0,1}(x) = T_x(t_0) - T_x(t_1),$$

when the knots are distinct. Otherwise it is zero. $B_{0,1}(x)$ is a square pulse. It takes a positive constant value for $t_0 \leq x < t_1$ and is zero elsewhere. It is continuous from the right, but not from the left. The generalized divided difference is defined by right derivatives when knots coincide. It follows that the B-Splines defined in this way are infinitely differentiable from the right. However, sometimes B-Splines are defined so that they are infinitely differentiable from the left.

Let us now look at B-Splines of order 2. Let $\tau = \{0, 1, 2\}$. From the definition we find

$$B_{0,2}(x) = T_x^2(2) - 2T_x^2(1) + T_x^2(0).$$

$$B_{0,2}(x) = 0, x < 0$$

$$B_{0,2}(x) = x, 0 \leq x < 1$$
$$B_{0,2}(x) = 2 - x, 1 \leq x < 2$$
$$B_{0,2}(x) = 0, 2 \leq x.$$

This is a triangular hat function with support in the interval $[0, 2]$. When knots coincide there is a loss of continuity. For example let $\tau = \{0, 0, 1\}$. Then

$$B_{0,2}(x) = G_{\{0,0,1\}} T_x^2(0)$$
$$= (T_x^2(1) - T_x^2(0)) - \frac{dT_x^2(0)}{dt}$$
$$= T_x^2(1) - T_x^2(0) + 1.$$

Thus

$$B_{0,2}(x) = 0, x < 0$$
$$B_{0,2}(x) = 1 - x, 0 \leq x < 1$$
$$B_{0,2}(x) = 0, 1 \leq x.$$

One order of continuity has been lost at 0.

Higher order B-splines can be calculated by recursion.

$$B_{i,k} = \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1} + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1},$$

where a term is zero if its denominator is zero. This method of computation is known as the Cox-DeBoor algorithm.

To show how simple computation is we give the following FORTRAN program for calculating a kth order B-spline.

```
c+ bspln   b-spline function  (recursive definition)
      function bspln(k,x,t)
c  parameters:
c      k-order of bspline
c      x-point
c      t-knot vector
c recursively calculated from lower order b-splines
c note: not all compilers support recursion
```

```
      dimension t(*)
      zero=0.
      b=zero
      if(t(1+k).gt.t(1))then
       if(k.eq.1)then
         if((x.ge.t(1)).and.(x.lt.t(2)))then
               b=1.
         endif
       else
        c=t(k)-t(1)
        if(c.gt.zero)then
            b=(x-t(1))*bspln(k-1,x,t(1))/c
        endif
        c=t(1+k)-t(2)
        if(c.gt.zero)then
         b=b+(t(1+k)-x)*bspln(k-1,x,t(2))/c
        endif
       endif
      endif
      bspln=b
      return
      end
```

A b-spline curve is a linear combination of b-splines. Thus if a set of np control points (px_i, py_i, pz_i) is given then one simply multiplies the ith vector by the ith B-spline and sums to get a point on the curve. The following subroutine does this summation.

```
c+ bsc3    point on b-spline space curve (recursive definition)
      subroutine bsc3(k,s,t,px,py,pz,np,x,y,z)
c parameters:
c        k-order of the curve, k >= 1
c        s-parameter value
c        t-knot vector of length np+k;
c        t(1)<=t(2)<=......<=t(np+k)
c    px,py,pz-coordinates of the polygon vertices
c       np-number of vertices
c      x,y,z-returned coordinates of point on curve
```

```
c  the curve is
c c(s) = p(1)*bspln(k,s,t(1))+...+p(t(np))*bspln(k,s,t(np))
c note: not all compilers support recursion
      dimension t(*),px(*),py(*)
      x=0.
      y=0.
      z=0.
      do 10 i=1,np
      b=bspln(k,s,t(i))
      x=x+b*px(i)
      y=y+b*py(i)
      z=z+b*pz(i)
10    continue
      return
      end
```

Multiple knots result in a discontinuity at the knot. A double knot, for a cubic B-spline (4th order B-spline), causes the second derivative to be discontinuous, a triple knot causes the first derivative to be discontinuous, and a quintuple knot causes a discontinuous value at the knot, that is a jump or step in the value. It is in this way that B-splines can represent all piecewise polynomials regardless of their knot discontinuities.

A Wilson-Fowler spline is a rather specialized spline calculated with an iterative algorithm. It is a parametric curve, each coordinate function (i.e. x(t) or y(t)) being a piecewise cubic. It has continuous tangent and curvature and in some sense is smooth. It has some interest, partially historical. The details of this spline are not explained here. There is often confusion about types of splines. Let us take the Wilson-Fowler spline as an example. Consider the questions: (1)Is a B-spline a Wilson-Fowler spline? and conversely, (2)Is a Wilson-Fowler spline a B-spline? Let me try to explain the answer with an analogy. Consider ordinary two dimensional euclidean vectors, i.e. directed arrows in the plane. It is common to write such vectors in terms of the basis vectors **i** and **j**, which are the unit vectors in the coordinate directions. Call the vectors that may be written this way ij's. Now suppose we calculate a class of two dimensional force vectors that occur in some kind of mechanical device. Is each force vector an ij vector? Yes, because every two dimensional vector is an ij vector, that is, **i** and **j** are a basis of two dimensional vectors. Is every **ij** vector a force vector? Not necessarily, because,

forces in the device might only occur in certain directions.

The Wilson-Fowler spline is like a force vector, and B-splines are like **ij** vectors. A Wilson-Fowler spline is not an arbitrary piecewise polynomial curve. It is a special piecewise polynomial curve. It has a B-spline representation just as every force vector has an ij representation, but since it is special, not every B-spline curve is a Wilson-Fowler spline, just as not every ij vector is a force vector. The well known parametric cubic splines that are parameterized by chord length, have geometric continuity, but again they are not necessarily Wilson-Fowler splines. Such parametric cubic splines may be found by solving a set of tridiagonal linear equations. To show explicitly that the parametric cubic spline and the WF-spline are different, The coordinate functions of a WF-spline and a parametric cubic spline may be compared directly and the difference computed. However, the maximum differences for each coordinate function may occur at different parameter values. Still one may use the pythagorean theorem to find a bound on the absolute difference.

Again the important thing to remember is that the B-splines are a basis of the space of piecewise polynomial. That means that if we have a curve that is made up of pieces, each of which is a polynomial, then we can write it in B-spline form. In particular this is true of the Wilson-Fowler spline. It can be written in B-spline form. Program WFBS.FTN does such a conversion.

Let us now discuss the spline curve (or surface) known as the NonUniform Rational B-Spline, which is often called a NURBS (or incorrectly a NURB). There was a time when most B-splines were based on uniformly spaced knots. Such cubic splines are real splines, meaning that they are smooth and have continuous second derivatives. This is because they have no coincident knots. This also means that such splines are not a general basis of piecewise polynomial curves. They do not allow discontinuities. So nonuniform means that multiple knots are allowed and that distances between parameter knot values can vary. This naming is unfortunate because the nonuniform class is the more general one. Rational means that the coordinate functions, which make up the pieces of the curve are rational functions. These are functions $x/w, y/w$,and z/w where x, y, z, and w are polynomials. The class may be regarded as simply a four dimensional B-spline space, that is, a homogeneous or projective space. This allows curves that are not polynomials, such as conic sections, to be represented exactly. We shall not go into the details of projective space, but will remark that the ideas are not terribly difficult, and that rational B-splines are not any more difficult than polynomial B-splines.

Frequently the pieces of a NURBS will have a denominator equal to one,

so that the NURBS is equivalent to a polynomial B-spline. In the future, one should refer to a NURBS simply as a B-spline.

To really understand splines, one must dig into the mathematical theory and compute with them. Understanding in science and mathematics only comes through hard work: one must do the problems, not just passively listen to the lecture. Lectures are only for motivation and direction.

2.9 Perspective Projection

Let Q be a point in real projective 3 Space (RP3). Q has component vector

$$Q = (Q_1, Q_2, Q_3, Q_4) = (x, y, z, w),$$

where the components are the homogeneous coordinates of the point. Consider a plane with coordinate vector A. The equation of the plane is given as an inner product,

$$(A, Q) = A_1Q_1 + A_2Q_2 + A_3Q_3 + A_4Q_4 = 0.$$

Let P be a projection point not in the plane. Let $L(Q)$ be the point of intersection of the plane with the line

$$Q + tP.$$

Then

$$(A, Q + tP) = 0.$$

Solving for t, we find

$$L(Q) = Q - \frac{(A, Q)}{(A, P)}P.$$

L is a linear projection operator, i.e. $L * L = L$. Using a rigid motion transformation, we may transform the plane to the xy plane, and the projection point to the positive z axis. Hence we consider the special case where the projection plane has equation

$$z = 0,$$

and the projection point P is on the positive z axis. Then

$$A = (0, 0, 1, 0)^T,$$

and

$$P = (0, 0, p_3, p_4)^T.$$

In this special case of projection into the xy plane,we write

$$L = L_{xy}.$$

Then

$$(x', y', w')^T = L_{xy}Q = (x, y, z, w)^T - \frac{z}{p_3}(0, 0, p_3, p_4)^T.$$

$$= (x, y, 0, w - z\frac{p_4}{p_3})^T.$$

$$L_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -p_4/p_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix},$$

where d is the distance from the projection plane to the projection point. Given a projection point P and a center point C, let T_1 be an affine transformation translating C to the origin. Let P have elevation angle θ, and azimuth angle ϕ. The azimuth angle gives the direction in the xy plane measured from the positive x axis. The azimuth is the rotation angle, about the z axis, that the xz plane would have to be rotated, so that it would include the projection direction. For example, the azimuth of the positive x axis is zero, and the azimuth of the positive y axis is $\pi/2$. The elevation is the angle from the xy plane to the projection direction. We rotate the image of the projection point into the yz plane with a rotation T_2 about the z-axis by angle $-\phi - \pi/2$. We rotate the image of the projection point up to the z-axis, with a rotation T_3 about the positive x axis, by an angle $-(\pi/2 - \theta)$. The complete perspective transformation matrix is

$$L = L_{xy}T_3T_2T_1,$$

where

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_2 = \begin{bmatrix} c(\phi + \pi/2) & s(\phi + \pi/2) & 0 & 0 \\ -s(\phi + \pi/2) & c(\phi + \pi/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -s(\phi) & c(\phi) & 0 & 0 \\ -c(\phi) & -s(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\theta - \pi/2) & -s(\theta - \pi/2) & 0 \\ 0 & s(\theta - \pi/2) & c(\theta - \pi/2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & s(\theta) & c(\theta & 0 \\ 0 & -c(\theta) & s(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Example. Isometric projection. The picture plane has coordinate vector

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

The projection point is at infinity,

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

Hence $d = \infty$, and so

$$L_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The azimuth angle is $\phi = \pi/4$, and

$$\sin(\phi) = \frac{\sqrt{2}}{2},$$

$$\cos(\phi) = \frac{\sqrt{2}}{2}.$$

The tangent of the elevation angle is

$$\tan(\theta) = \frac{1}{\sqrt{2}}.$$

Then

$$\sin(\theta) = \frac{\sqrt{3}}{3}$$

$$\cos(\theta) = \frac{\sqrt{6}}{3}$$

T_1 is the identity, because the center of projection is already at the origin.

$$T_1 = I.$$

$$T_2 = \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{3} & \frac{\sqrt{6}}{3} & 0 \\ 0 & -\frac{\sqrt{6}}{3} & \frac{\sqrt{3}}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Then

$$T_3T_2T_1 = \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ -\frac{\sqrt{6}}{6} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & 0 \\ \frac{\sqrt{12}}{6} & \frac{\sqrt{12}}{6} & \frac{\sqrt{3}}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then

$$L = L_{xy}T_3T_2T_1 = \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ -\frac{\sqrt{6}}{6} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A projective transformation matrix is defined only up to a scalar multiple. Hence we can factor out $\sqrt{6}/6$, to get

$$L = \begin{bmatrix} -\sqrt{3} & \sqrt{3} & 0 & 0 \\ -1 & -1 & 2 & 0 \\ 0 & 0 & 0 & \sqrt{6} \end{bmatrix}.$$

The images of the unit points are,

$$L\begin{bmatrix}1\\0\\0\\1\end{bmatrix}=\begin{bmatrix}-\sqrt{3}\\-1\\0\\\sqrt{6}\end{bmatrix},$$

$$L\begin{bmatrix}0\\1\\0\\1\end{bmatrix}=\begin{bmatrix}\sqrt{3}\\-1\\0\\\sqrt{6}\end{bmatrix},$$

and

$$L\begin{bmatrix}0\\0\\1\\1\end{bmatrix}=\begin{bmatrix}0\\2\\0\\\sqrt{6}\end{bmatrix}.$$

The image of the coordinate axes are as shown in the Figure. The lengths are foreshortened by the factor $1/\sqrt{6}$. Conventional isometric drawing, true lengths are laid off along the coordinate axes, so that the matrix is

$$\begin{bmatrix}-\sqrt{3}/2 & \sqrt{3}/2 & 0 & 0\\-1/2 & -1/2 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}.$$

In perspective drawing, the horizon is defined as the intersection of the xy-plane with the plane at infinity

$$w=0.$$

A horizon point can not be reached in 3-space, but we may "see" its projection. A line at infinity can project to a finite line in the picture plane. Similarly the point at infinity where two parallel lines meet, may project into a finite point in 2-dimensional space. Such a point is called a vanishing point. For example, the point

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

is the point at infinity, where parallel 45 degree lines in the plane meet at infinity. The vanishing point is $L(Q)$ in the picture plane. The horizon in the picture plane is determined by two vanishing points, corresponding to two sets of parallel lines in the xy-plane. When the projection point P is at infinity, as in isometric projection, parallel lines in 3-space are mapped to parallel lines in 2-space so there are no vanishing points.

If the line through P and Q is parallel to the picture plane, then the projection of Q will be a point at infinity in the picture plane.

The following information is needed to construct the perspective transformation: (1)The center of the projection plane

$$(C_x, C_y, C_z)^T.$$

. (2)The direction of the projection point from the center, which is specified by the azimuth angle ϕ, and the elevation angle θ. (3)The reciprocal of the distance from the projection point to the picture plane

$$\frac{1}{d},$$

which is zero for a projection point at infinity.

Note that the usual projection from an infinite point on the z axis is specified by elevation $\theta = \pi/2$, azimuth $\phi = -\pi/2$, and reciprocal distance $1/d = 0$. This gives L_{xy}.

Here is an implementation of the calculation in C:

```
/*c+ ppmat   matrix of a perspective projection.
input:
 cntr       homogeneous coordinates of center.
 elev,azimu (degrees) coordinates of eye point on a sphere with
             center cntr. Suppose the cntr is the coordinate origin,
             then the eye point is:
             location     elev      azimu
              x-axis        0          0
              y-axis        0         90
              z-axis       90        any angle
 deye        distance of eye point from cntr. To get
             orthogonal projection make deye very large.
output:
 a           3 by 4 matrix of the projection from projective
```

```
             3-space to projective 2-space.
 prj         eye projection point (output), may have use for
             visibility testing.
  If p is a vector containing the homogeneous coordinates
  of a 3-dimensional point, then a*p = q is a vector
  containing the homogeneous coordinates of the 2-dimensional
  projection.  The affine coordinates of the projected point are:
  x=q[0]/q[2], y = q[1]/q[2]
  for isometric projection set elev=35.264, azimu=45, deye large.
*/
void ppmat(cntr,elev,azimu,deye,a,prj)
double *cntr,elev,azimu,deye,*a,*prj;
{
 extern void trnslt();
 extern void rxaxis();
 extern void matml();
 int i,j;
 double b[4],t[16],r[16],u[4];
 double e[16],p[12],pi,ang1,ang2,cs1;
 pi = 3.141592653589;
 ang1 = elev*pi/180.0;
 ang2 = azimu*pi/180.0;
 *u = cos(ang1)*cos(ang2);
 *(u+1) = cos(ang1)*sin(ang2);
 *(u+2) = sin(ang1);
 *(u+3) = 1.0;
 for(i=0; i<3; i++){
  *(prj+i) = *(cntr+i)+deye**(u+i);
  *(b+i) = -(*(cntr+i));
 }
 *(b+3) = 1.0;
 trnslt(b,t);
 /*compute the rotation matrix r taking vector*/
 /*u to the positive x-axis.*/
 rxaxis(u,r);
 matml(r,t,e,4,4,4);
 for(i=0; i<3; i++){
  for(j=0; j<4; j++){
```

```
   *(p+i+j*3) = 0.0;
  }
 }
 *(p+3) = *(p+7) = *(p+11) = 1.0;
 *(p+2) = -(1.0/deye);
 matml(p,e,a,3,4,4);
}
```

See the FORTRAN library file **mathlibd.ftn** for a Fortran version of ppmat. An alternate method of doing a perspective projection is given in the section **Z-Buffer Triangle Rendering**.

Chapter 3

A Device Independent Graphics System

3.1 EGraphics Commands

EGraphics is a simple file system for representing device independent graphics. The file contains one graphics command per line. The first letter (or letters) on a line defines the action. The rest of the line defines parameters for the action, e.g. d1.0 2.0 means draw to 1.0 2.0. Commands supported:

```
v (viewport)
w (window)
m (move)
d (draw)
a (arc)
b (bezier cubic)
s (symbol)
t (text)
g (text angle)
p (transmit raw postscript command)
z (text size),
#tc (specify center of text)
Sxmin xmax ymin ymax (size of the full viewport in millimeters)
```

The *w* command is the window command (w should occur in the first column):

```
w xmn xmx ymn ymx
```

For example

```
w -10 20 -5 5
```

defines a window around user data with range

$$-10 \leq x \leq 20$$

$$-5 \leq y \leq 5.$$

The v command is the viewport command:

```
v xmn xmx ymn ymx
```

It defines a viewport on the graphics device. It maps the window defined by the w command onto a subregion of the device display. This may be a crt screen, a paper plot, or a laser printer page. The full display of the device is specified as

```
v -1 1 -1 1.
```

The command

```
v 0 1 0 1
```

would map the w window to the upper right quadrant of the display.

The m command is the move command:

```
m x y
```

moves to a point with coordinates (x, y).

The d command is the draw command:

```
d x y
```

draws a line from the current position to the point with coordinates (x, y).

The S command is a plot size command.

It sets up a plot size in millimeters for a variable size device, such as a plotter that supports HPGL. It is ignored when it is not appropriate for a given device. For example, to plot on a region 8.5 inches wide by 11 inches high, the command is

```
S 0 215.9 0 279.4
```

The command for plotting on a **Calcomp** thermal imaging plotter, using **plthpgl.c**, in a region defined in inches by

$$-10 \leq x \leq 10,$$

$$-10 \leq y \leq 10,$$

is

```
S -254 254 -254 254
```

The *a* command draws an arc

```
a centerx centery radius angle1 angle2 NumberPoints
```

The angles are in radians, so to draw a complete circle with center $(1, 1)$, and radius 2, and having 100 points, the command is

```
a 1 1 2 0 6.28 100
```

The *b* command draws a bezier curve segment

```
b x1 y1 x2 y2 x3 y3 x4 y4
```

The *s* command draws a symbol

```
s x y SymbolNumber SymbolSize
```

The symbol size is relative to the viewport window. The viewport window has height

$$2 = (1 - (-1)),$$

so if the character height is to be 3 per cent of the screen height, then the height would be

$$2(.03) = .06.$$

The command *c* changes the color. The command

```
c 5
```

changes to color number 5. The postscript driver produces a postscript **rgb** command, for each color change. But it is preceded by a percentage sign and so is a postscript comment. One can edit the file and remove the percentage sign, when the file is to be sent to a color postscript printer.

The *t* command draws text.

```
t x y nchar TextString
```

For example the command

```
t2.612903 8.35 11 Sample Plot
```

draws the eleven character string "Sample Plot", starting at location

$$x = 2.612903, y = 8.35.$$

There is a sequence of commands to draw centered text:

```
#tc x y 1 (remark: the 1 turns on the centered state and stores center x y)
t 0 0 25 This is centered at (x,y) (Remark: the 0 0 is ignored)
#tc x y 0 (remark: the 0 resets the state to original, x y is ignored)
```

The command p transmits a raw postscript command. For example these commands fill a polygon defined by a preceeding move and draws:

```
p gsave
p 0.55 setgray
p fill
p grestore
p stroke
```

The command z changes the text size (relative to viewport window)

```
z .05
```

The command g changes the text angle (relative to the viewport window) The command

```
g 90
```

rotates the text 90 degrees.

3.2 Files for HPGL Plots On A Calcomp Plotter

The 36 inch Calcomp 52436 DrawingMaster Plus thermal imaging plotter, which is named **rutledge**, plots with x running down the paper roll, and with y running across the plotter roll from left to right.

The gi command, which defines the device region for plotting on the **Calcomp** plotter, using **plthp.c**, in a region defined in inches by

$$-10 \leq x \leq 10,$$

$$-10 \leq y \leq 10,$$

is

```
S -254 254 -254 254
```

Here the default plotting origin is at the center of the roll. We will show how to move the plot to the left edge of the plotter.

The default origin is located at the center of the paper roll, at about 18 inches from the left end. The best strategy for nondistorted plotting of a drawing is to use a square viewport, a square window, and a square device window. That is, the parameters of v, the parameters of w, and the parameters of S, should each define a square. Further, for full scale plotting, the parameters of w and S should be equivalent, that is they should define the same size window, taking into account that the parameters in S are in millimeters and those of w might be in inches. Suppose for example that we want a full scale plot of a drawing given in inches with

$$-20 \leq x \leq 25, -10 \leq y \leq 5$$

So that the length in the x direction is 45 inches and the length in the y direction is 20 inches. First we must define a square containing the drawing. The size of the square will be equal to the largest dimension here 45. Thus a possible square is

$$-20 \leq x \leq 25, -10 \leq y \leq 35$$

This is defined by

```
w-20 25 -10 35
```

We need to define a square region on the plotter of the same size. If this region is defined by

```
Sx1 x2 y1 y2
```

Then $(-20, -10)$ maps to (x_1, y_1). At the left side of the plotter, the y coordinate equals about $-(36/2)25.4 = -457.2$ millimeters. To be safe, let us take the left edge of the plotter to be at $y_1 = -450$ millimeters. Then the right boundary of the device plotting region is equal to this number plus the size of the window in millimeters.

$$y_2 = -450 + (45) * (25.4) = 693$$

We may take

$$x_1 = 0$$

and

$$x_2 = x_1 + (45)(25.4) = 1143.$$

Thus our device window definition is

```
S0 1143 -450 693
```

Notice that this device window extends from the left edge of the plotter drum to several inches beyond the right edge. This is not a problem because we are not drawing near the right edge. We have defined a square window that extends beyond the plotter. By making all of the windows square, the characters and symbols will be in the right proportion. And since most plot devices are approximately square, the drawing will be proportional on all hpgl plotting devices that are square. If a full scale drawing is greater in the y direction than the 36 inch size of the drum, then the drawing must be rotated to fit on the plotter paper.

A drawing can be reduced in scale by multiplying the S parameters by a scaling factor, for example multiplying by 1/2 will make a half scale drawing.

Here are examples of two gi files that are centered on the plotter rather than being moved to the left side of the plotter. The first file **longpole.gi** produces a quarterscale drawing after being converted to a hpgl file with program **plthp**.

```
%1016 millimeters = 40 inches
%S0 1016 0 1016
% 1/4 size 40 inches -> 10 inches
S0 254 0 254
v-1 1 -1 1
%windows should be square
```

```
w0 40 0 40
% character height = .03(10 inches) = .3 inches
z .03
m0 0
d40 0
d40 2
d0 2
d0 0
t1 3 48 A very very very long pole, 40 Inches (1/4 size)
```

Note that the percent character is being used to denote a comment. The second file **longpole2.gi** produces a fullscale drawing after being converted to a hpgl file with program **plthp**.

```
% cat longpole2.gi
%1016 millimeters = 40 inches
%To make plot start at left edge, subtract 18*25.4=457.2 from ymn ymx
S0 1016 0 1016
% 1/4 size 40 inches -> 10 inches
%S0 254 0 254
v-1 1 -1 1
%windows should be square
w0 40 0 40
% character height = .03(40 inches) =1.2 inches
z .03
m0 0
d40 0
d40 2
d0 2
d0 0
t1 3 49 A very very very long pole, 40 Inches (full size)
```

It is conceivable that for an extremely large plot, say for the full scale plot of a telephone pole, that the integer parameters in the HPGL IP command (this defines the window size in HP plotter units) could cause an integer overflow (there are 40 HP plotter units per millimeter). This remains to be tested.

3.3 EGraphics Examples

This example is an annotated plot of a line, an arc, a text string, a symbol, and a Bezier curve:

```
rem file: test2.gi
rem define viewport as whole screen
v-1 1 -1 1
rem define world window xmin=-7 xmax=7 ymin=-6 ymax=8
w-7 7 -6 8
rem change to color 3
c3
rem move to  x=0 y=0
m0 0
rem draw to x=5 y=3
d5 3
c4
rem draw circular arc: center x=1 y=5 angle1=0 angle2=2pi points=100
a1 1 5 0 6.28 100
c5
rem draw bezier curve:
rem control points x1=-1 y1=1 x2=2 y2=2 x3=3 y3=-1 x4=4 y4=0
b-1 1 2 2 3 -1 4 0
c6
rem set text size to 3 percent of screen
z .03
rem set text angle to 45 degrees
g 45
rem draw test string at x=-1 y=-2 of length 9 characters
t-1 -2 9 Draw Text
c7
rem draw symbol number 3 at x=-1 y=0 of size 2 percent
s-1 0 3 .02
```

This example draws a simple x-y plot. It was produced with program **pc.ftn**, which makes a plot from sets of x-y data.

```
v-.75 1 -.75 .75
w1 6 1 8.7
```

```
m1 2
d2 4
d3 1
d4 8
d5 7
d6 3
c9
s2 4 2 .1E-1
s3 1 3 .1E-1
s4 8 4 .1E-1
s5 7 5 .1E-1
v-1 1.2 -1 1
w.2857143 6.571429 -.2833333 9.983333
z.4E-1
c5
g0
z.35E-1
m1 1
d6 1
m1 1
d1 .9101667
t1 .640667 1 1
m2.25 1
d2.25 .9101667
t2.25 .640667 4 2.25
m3.5 1
d3.5 .9101667
t3.5 .640667 3 3.5
m4.75 1
d4.75 .9101667
t4.75 .640667 4 4.75
m6 1
d6 .9101667
t6 .640667 1 6
#tc    3.500000      0.4166667       1.000000
t3.17 .2813339 6 X-Axes
#tc    3.500000      0.4166667      0.0000000
g90
```

```
z.35E-1
m1 1
d1 8
m1 1
d.95 1
t.9000001 1 1 1
m1 2.75
d.95 2.75
t.9000001 2.75 4 2.75
m1 4.5
d.95 4.5
t.9000001 4.5 3 4.5
m1 6.25
d.95 6.25
t.9000001 6.25 4 6.25
m1 8
d.95 8
t.9000001 8 1 8
#tc   0.6666666       4.500000       1.000000
t.7000003 3.9071 6 Y-Axes
#tc   0.6666666       4.500000      0.0000000
c2
z.5E-1
g0
#tc    3.500000       8.350000       1.000000
t2.612903 8.35 11 Sample Plot
#tc    3.500000       8.350000      0.0000000
```

This example plots a cube. It demonstrates the use of raw postscript commands. The postscript driver will produce a postscript file that makes a shaded image. The file was produced by program **plato.ftn**.

```
w-3.131 3.108 -3.175 3.064
v-1 1 -1 1
m.1284 -0.823
d.8039 -0.4973
d.8071 .5867
d.129 .2693
```

```
d.1284 -0.823
pgsave
p  0.55 setgray
pfill
pgrestore
pstroke
m-0.8058 -0.5858
d.1284 -0.823
d.129 .2693
d-0.809 .5004
d-0.8058 -0.5858
pgsave
p  0.76 setgray
pfill
pgrestore
pstroke
m.129 .2693
d.8071 .5867
d-0.127 .8138
d-0.809 .5004
d.129 .2693
pgsave
p  0.34 setgray
pfill
pgrestore
pstroke
```

3.4 Merging Plot Files, Adding Window Commands

When merging two or more plot files, we usually need to define a new window that contains all of the plotting region of all of the files. A program, called **pltmerge.c**, will perform this redefinition. It searches the plot files to find a containing window. It writes to the screen the minimum containing window information, and produces a new merged plot file. It replaces all window definitions by a single window definition and it makes this a uniform window,

which means that the x dimension of the plot region equals the y dimension. By default the window is also expanded by 5 percent. This can be overridden by using -w as the last parameter.

If we had three plot files **p1.gi**, **p2.gi**, and **p3.gi**, we could combine into a single file **p.gi**, by typing the command

```
pltmerge p1.gi p2.gi p3.gi p.gi
```

pltmerge.c may also be used to add a window definition to a file that does not have one. For example consider the file **p1.gi**

```
c3
m0 0
d5 3
c4
a1 1 5 0 6.28 100
c5
b-1 1 2 2 3 -1 4 0
c6
z .03
g 45
t-1 -2 13 pltfiles test
c7
s-6 0 3 .02
```

This file has no window or viewport command. The file contains a line, a circle, a bezier curve, a text string, and a symbol. We run the program with the command

```
pltmerge p1.gi p.gi
```

We get screen output

```
 containing window -6 6 -4 6
```

which gives the smallest window that contains the plot data. The new file is

```
v-1 1 -1 1
w-6.3 6.3 -5.3 7.3
c3
m0 0
```

```
d5 3
c4
a1 1 5 0 6.28 100
c5
b-1 1 2 2 3 -1 4 0
c6
z .03
g 45
t-1 -2 13 pltfiles test
c7
s-6 0 3 .02
```

Refer to the figure to view the graphical elements of this plot. The new plot file, **p.gi**, now has a window and viewport definition. Notice that the window differs from the minimum containing window. It is uniform, and has been expanded by five percent. If one actually wanted the minimum containing window, it could be pasted into the file from the screen output. A listing of **pltmerge.c** is given in the next section.

3.5 Listing of pltmerge.c

```
/*pltmerge.c merge plot files into a single plot file \n");*/
/* 12/21/94 */
#include <stdio.h>
#include<ctype.h>
#include <math.h>

main (argc,argv)int argc;char *argv[];{
 extern void words();
 extern void stov();
 extern double dmn();
 extern double dmx();
 FILE *f[30],*out;
 double xmn = 1.e30;
 double xmx = -1.e30;
 double ymn = 1.e30;
 double ymx = -1.e30;
 double d[20];
 double s,cx,cy;
 int ws[10],wl[10],i,j,k,n,uw;
 int nfiles;
 char c1,c2;
 char a[200],*b,*c;
 if(argc < 3){
  printf(" merge plot files into a single plot file. \n");
  printf(" find enclosing window. \n");
```

```
 printf(" if missing, add window and viewport statements.\n");
 printf(" default: uniformly scaled window expanded by 5 percent. \n");
 printf(" get minimum enclosing window by giving as last parameter: -w \n");
 printf(" pltmerge file1.gi file2.gi ... fileout.gi [-w] \n");
 exit(1);
}
c1=argv[argc-1][0];
c2=argv[argc-1][1];
if((c1 == '-')&&(c2 == 'w')){
 nfiles = argc-3;
 uw = 0;
}
else{
 nfiles=argc-2;
 uw=1;
}
for(i=0; i < nfiles ; i++){
 f[i]=fopen(argv[i+1],"r");
 if(f[i] == NULL){
  printf(" I can not find one of the input files\n");
  exit(0);
 }
}
out = fopen(argv[nfiles + 1],"w");
for(i=0;i < nfiles ;i++){
 while(fgets(a,200,f[i]) != NULL){
  for(j=0; j < 200; j++){
   k = j;
   if(a[j] != ' ') break;
  }
  c=a+k;
  if((j=strlen(c)) > 2){
   /* delete newline from end of string */
   *(c+j-1)='\0';
   if(*c =='/'){
    break;
   }
   b=c+1;
   if(*c=='w'){
    words(b,&n,ws,wl);
    stov(b,ws,wl,4,d);
    xmn = dmn(xmn,*d);
    xmx = dmx(xmx,*(d+1));
    ymn = dmn(ymn,*(d+2));
    ymx = dmx(ymx,*(d+3));
   }
   if((*c=='d')||(*c=='m')||(*c=='s')||(*c=='t')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,2,d);
    xmn = dmn(xmn,*d);
    xmx = dmx(xmx,*d);
    ymn = dmn(ymn,*(d+1));
    ymx = dmx(ymx,*(d+1));
   }
   if((*c=='a')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,3,d);
```

```
    xmn = dmn(xmn,*d - *(d+2));
    xmx = dmx(xmx,*d + *(d+2));
    ymn = dmn(ymn,*(d+1) - *(d+2));
    ymx = dmx(ymx,*(d+1) + *(d+2));
   }
   if((*c=='b')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,8,d);
    xmn = dmn(xmn,*d);
    xmn = dmn(xmn,*(d+2));
    xmn = dmn(xmn,*(d+4));
    xmn = dmn(xmn,*(d+6));

    xmx = dmx(xmx,*d);
    xmx = dmx(xmx,*(d+2));
    xmx = dmx(xmx,*(d+4));
    xmx = dmx(xmx,*(d+6));

    ymn = dmn(ymn,*(d+1));
    ymn = dmn(ymn,*(d+3));
    ymn = dmn(ymn,*(d+5));
    ymn = dmn(ymn,*(d+7));

    ymx = dmx(ymx,*(d+1));
    ymx = dmx(ymx,*(d+3));
    ymx = dmx(ymx,*(d+5));
    ymx = dmx(ymx,*(d+7));
   }
  }
 }
}
for(i=0; i < nfiles ; i++){
 rewind(f[i]);
}
if(uw){
 cx=(xmn+xmx)/2.;
 cy=(ymn+ymx)/2.;
 s=1.05*dmx(xmx-xmn,ymx-ymn);
 xmn=cx-s/2.;
 xmx=cx+s/2.;
 ymn=cy-s/2.;
 ymx=cy+s/2.;
}
fprintf(out,"v-1 1 -1 1\n");
fprintf(out,"w%g %g %g %g\n",xmn,xmx,ymn,ymx);
for(i=0;i < nfiles ;i++){
 while(fgets(a,200,f[i]) != NULL){
  for(j=0; j < 200; j++){
   k = j;
   if(a[j] != ' ') break;
  }
  c=a+k;
  if((j=strlen(c)) > 2){
   /* delete newline from end of string */
   *(c+j-1)='\0';
   if(*c =='/'){
    break;
```

```
    }
    b=c+1;
    if((*c != 'w') && (*c != 'v') ){
     fprintf(out,"%s\n",c);
    }
   }
  }
 }
}
/*c+ words  get words in string */
void words(s,n,ws,wl) int *n,*ws,*wl;char *s;
/*
     s-string
     n-number of words found in string
       (words are groups of characters bordered by spaces)
    ws[i]-word start position in s of ith word
    wl[i]-word length of ith word
*/
{
extern int lenstr();
static int i,j,k,c,nc;
/*
     c=character type, nc=next character type
     c=0 if space c=1 otherwise
*/
     /*printf(" string into words is %s \n",s);*/
    k = strlen(s);
     /*printf(" length= %d \n",k);*/
    j = *n = c = 0;
    for(i=0; i<=k;i++) {
        if(i == k) nc = 0;
        else  {
            if(s[i]==' ') nc = 0; else  nc = 1;
              }
        if(c < nc) {*(ws+*n)=i;*n = *n+1;}
        if(c > nc)*(wl+ *n-1)=i - *(ws+ *n-1);
        c = nc;
    }
}
/*c+ stov string to vector (used with routine words)*/
void stov(b,ws,wl,n,d)int n;double *d;int *ws,*wl;char *b;
/*
  call function words to compute ws and n
     b-string
     n-number of words in string
       (words are groups of characters bordered by spaces)
    ws[i]-word start position in s of ith word
    wl[i]-word length of ith word
    d-vector returned
*/
{
char s[25];
static int m;
    for(m=0; m<n; m++)
   {
copys(s,b,ws[m],wl[m]);
*(d+m) = atof(s);
```

```
    }
}
/*c+ copys string copy*/
copys(to,from,pos,num)char *to,*from;int pos,num;
{
char b[256];
strcpy(b,from);
b[pos+num]='\0';
strcpy(to,b+pos);
}
/*c+ dmn minimum of two doubles */
double dmn(a,b)double a,b;{
 if( a < b){
  return(a);
 }
 else{
  return(b);
 }
}
/*c+ dmx maximum of two doubles */
double dmx(a,b)double a,b;{
 if( a < b){
  return(b);
 }
 else{
  return(a);
 }
}
```

3.6 Adding Axes and Titles To A Plot

The program **pltax.c** adds axes and titles to a plot. As an example, suppose we have a file called **p1.gi**, that contains the graphics commands of the example given in the section on **pltfiles.c**. Then the command

```
pltax p1.gi p.gi X-Axis Y-Axis 'Pltax Example'
```

produces a plot file **p.gi**. The plot is shown in the figure. A multiword title must appear between a pair of Double quotes (single quotes in UNIX) so that the command interpreter does not take the individual words to be separate parameters. For example:

```
pltax p1.gi p.gi X-Axis Y-Axis "Pltax Example"
```

3.7 Listing of pltax.c

```
/*pltax.c create axes, and title labels for a gi plot.*/
```

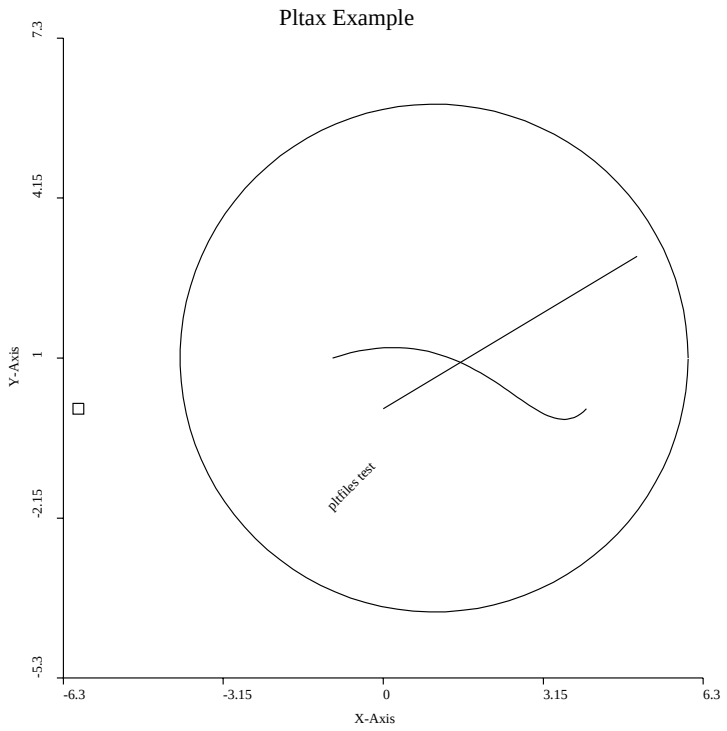

Figure 3.1: Program **pltax.c** adds axes and titles.

```
/* 12/22/94 */
#include <stdio.h>
#include<ctype.h>
#include <math.h>

main (argc,argv)int argc;char *argv[];{
 extern void words();
 extern void stov();
 extern double dmn();
 extern double dmx();
 extern char *rmlb();
 FILE *f[30],*out,*in;
 double xmn = 1.e30;
 double xmx = -1.e30;
 double ymn = 1.e30;
 double ymx = -1.e30;
 double d[20];
 double s,cx,cy,sf;
 double xmnn,xmxn,ymnn,ymxn;
 double x,y,dx,dy;
 double tl,tc;
 char t[100];
 char *tt;
 int ws[10],wl[10],i,j,k,n;
 int nfiles,ntic,sl;
 static char a[200],*b,*c;
 if(argc < 3){
  printf(" create axes, and title labels for a gi plot \n");
  printf(" pltax filein.gi fileout.gi x-title y-title plot-title\n");
  exit(1);
 }
 out = fopen(argv[2],"w");
 in=fopen(argv[1],"r");
 if(in == NULL){
  printf(" I can not find the input file\n");
  exit(0);
 }
  while(fgets(a,200,in) != NULL){
   for(j=0; j < 200; j++){
    k = j;
    if(a[j] != ' ') break;
   }
   c=a+k;
   if((j=strlen(c)) > 2){
    /* delete newline from end of string */
    *(c+j-1)='\0';
    if(*c =='/'){
     break;
    }
    b=c+1;
    if(*c=='w'){
     words(b,&n,ws,wl);
     stov(b,ws,wl,4,d);
     xmn = dmn(xmn,*d);
     xmx = dmx(xmx,*(d+1));
     ymn = dmn(ymn,*(d+2));
     ymx = dmx(ymx,*(d+3));
```

```
   }
   if((*c=='d')||(*c=='m')||(*c=='s')||(*c=='t')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,2,d);
    xmn = dmn(xmn,*d);
    xmx = dmx(xmx,*d);
    ymn = dmn(ymn,*(d+1));
    ymx = dmx(ymx,*(d+1));
   }
   if((*c=='a')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,3,d);
    xmn = dmn(xmn,*d - *(d+2));
    xmx = dmx(xmx,*d + *(d+2));
    ymn = dmn(ymn,*(d+1) - *(d+2));
    ymx = dmx(ymx,*(d+1) + *(d+2));
   }
   if((*c=='b')){
    words(b,&n,ws,wl);
    stov(b,ws,wl,8,d);
    xmn = dmn(xmn,*d);
    xmn = dmn(xmn,*(d+2));
    xmn = dmn(xmn,*(d+4));
    xmn = dmn(xmn,*(d+6));

    xmx = dmx(xmx,*d);
    xmx = dmx(xmx,*(d+2));
    xmx = dmx(xmx,*(d+4));
    xmx = dmx(xmx,*(d+6));

    ymn = dmn(ymn,*(d+1));
    ymn = dmn(ymn,*(d+3));
    ymn = dmn(ymn,*(d+5));
    ymn = dmn(ymn,*(d+7));

    ymx = dmx(ymx,*(d+1));
    ymx = dmx(ymx,*(d+3));
    ymx = dmx(ymx,*(d+5));
    ymx = dmx(ymx,*(d+7));
   }
  }
 }
printf(" containing window %g %g %g %g \n",xmn,xmx,ymn,ymx);
 rewind(in);
cx=(xmn+xmx)/2.;
cy=(ymn+ymx)/2.;
dx=.15*(xmx-xmn);
dy=.15*(ymx-ymn);
sf=1.5;
xmxn=xmx+sf*dx+.6*dx;
xmnn=xmn-sf*dx+.6*dx;
ymxn=ymx+sf*dy+.6*dy;
ymnn=ymn-sf*dy+.6*dy;
fprintf(out,"v-1 1 -1 1\n");
fprintf(out,"w%g %g %g %g\n",xmnn,xmxn,ymnn,ymxn);
 while(fgets(a,200,in) != NULL){
  for(j=0; j < 200; j++){
```

```
  k = j;
  if(a[j] != ' ') break;
 }
 c=a+k;
 if((j=strlen(c)) > 2){
  /* delete newline from end of string */
  *(c+j-1)='\0';
  if(*c =='/'){
   break;
  }
  b=c+1;
  if((*c != 'w') && (*c != 'v') ){
   fprintf(out,"%s\n",c);
  }
 }
}
fprintf(out,"g 0\n");
fprintf(out,"c 5\n");
fprintf(out,"z .05\n");
if(argc > 5){
 sl=strlen(argv[5]);
 tl=sl*(.05/2.)*(xmx-xmn);
 tc=(xmn+xmx-tl)/2.;
 fprintf(out,"t %g %g %d %s\n",tc,ymx + dy/5.,sl,argv[5]);
}
fprintf(out,"z .03\n");
fprintf(out,"m%g %g\n",xmn,ymn);
fprintf(out,"d%g %g\n",xmx,ymn);
ntic=5;
for(i=0;i < ntic; i++){
 x = i*(xmx-xmn)/(ntic-1) +xmn;
 fprintf(out,"m%g %g\n",x,ymn);
 fprintf(out,"d%g %g\n",x,ymn - dy/10.);
 sprintf(t,"%6.4g",x);
 tt=rmlb(t);
 sl=strlen(tt);
 fprintf(out,"t%g %g %d %s\n",x,ymn - dy/3.,sl,tt);
}
if(argc > 3){
 sl=strlen(argv[3]);
 tl=sl*(.03/2.)*(xmx-xmn);
 tc=(xmn+xmx-tl)/2.;
 fprintf(out,"t %g %g %d %s\n",tc,ymn - 7.*dy/10.,sl,argv[3]);
}
fprintf(out,"m%g %g\n",xmn,ymn);
fprintf(out,"d%g %g\n",xmn,ymx);
fprintf(out,"g 90\n");
for(i=0;i < ntic; i++){
 y = i*(ymx-ymn)/(ntic-1) +ymn;
 fprintf(out,"m%g %g\n",xmn,y);
 fprintf(out,"d%g %g\n",xmn - dx/10.,y);
 sprintf(t,"%6.4g",y);
 tt=rmlb(t);
 sl=strlen(tt);
 fprintf(out,"t%g %g %d %s\n",xmn - dx/3.,y,sl,tt);
}
if(argc > 4){
```

```
   sl=strlen(argv[4]);
   tl=sl*(.03/2.)*(ymx-ymn);
   tc=(ymn+ymx-tl)/2.;
   fprintf(out,"t %g %g %d %s\n",xmn - 7.*dx/10.,tc,sl,argv[4]);
  }
}
/*c+ words  get words in string */
void words(s,n,ws,wl) int *n,*ws,*wl;char *s;
/*
     s-string
     n-number of words found in string
       (words are groups of characters bordered by spaces)
    ws[i]-word start position in s of ith word
    wl[i]-word length of ith word
*/
{
extern int lenstr();
static int i,j,k,c,nc;
/*
     c=character type, nc=next character type
     c=0 if space c=1 otherwise
*/
     /*printf(" string into words is %s \n",s);*/
    k = strlen(s);
     /*printf(" length= %d \n",k);*/
    j = *n = c = 0;
    for(i=0; i<=k;i++) {
        if(i == k) nc = 0;
        else  {
            if(s[i]==' ') nc = 0; else  nc = 1;
              }
        if(c < nc) {*(ws+*n)=i;*n = *n+1;}
        if(c > nc)*(wl+ *n-1)=i - *(ws+ *n-1);
        c = nc;
    }
}
/*c+ stov string to vector (used with routine words)*/
void stov(b,ws,wl,n,d)int n;double *d;int *ws,*wl;char *b;
/*
  call function words to compute ws and n
     b-string
     n-number of words in string
       (words are groups of characters bordered by spaces)
    ws[i]-word start position in s of ith word
    wl[i]-word length of ith word
    d-vector returned
*/
{
char s[25];
static int m;
    for(m=0; m<n; m++)
   {
copys(s,b,ws[m],wl[m]);
*(d+m) = atof(s);
    }
}
/*c+ copys string copy*/
```

```
copys(to,from,pos,num)char *to,*from;int pos,num;
{
char b[256];
strcpy(b,from);
b[pos+num]='\0';
strcpy(to,b+pos);
}
/*c+ dmn minimum of two doubles */
double dmn(a,b)double a,b;{
 if( a < b){
  return(a);
 }
 else{
  return(b);
 }
}
/*c+ dmx maximum of two doubles */
double dmx(a,b)double a,b;{
 if( a < b){
  return(b);
 }
 else{
  return(a);
 }
}
/*c+ rmlb remove leading blanks from string*/
char *rmlb(a)char *a;{
 int n,j,k;
 n=strlen(a);
 for(j=0; j < n; j++){
  k = j;
  if(a[j] != ' ') break;
 }
 return(a+k);
}
```

Chapter 4

Postscript and More Graphics

Postscript is a programing language for describing graphic images.

4.1 Postscript Viewers

Two postscript viewers are **Ghostscript** and **xpsview**. Ghostscript runs on unix systems and on DOS. On Dos one types gs filename.ps. For help gs -h. To print on a laserjet plus do:

gs -sDEVICE=ljetplus filename.ps

Ghostscript can also direct output to bitmap files, for example

gs -sDEVICE=gif8 -sOutputFile=filename.gif filename.ps

Bitmap gif files can be viewed with the program **cshow**.

There are also ways that Ghostscript can create the **UNIX** public domain bitmap format **ppm**, which can be converted to many other formats using various public domain programs. For information on this see the Ghostscript doc files.

Ghostscript is an interactive interpreter. One can use it as a calculator:

```
gs>2 2 add
gs>stack (displays stack)
```

The number 4 is now on the stack.

xpsview is an X Windows application that runs on Silicon Graphics workstations. Xpsview requires a work station that is capable of doing display postscript. **Xpsview** does not like the full postscript header

```
%!PS Adobe blah blah
```

So instead use:

```
%!PS
```

The program **FreedomOfThePress**, is a postscript interpreter that prints to various nonpostscript printers. It also makes **PCX** graphics files that can be viewed with the graphics program **paintbrush**.

4.2 Shaded Facets

Polyhedra can be displayed with or without darkened borders. The following postscript program displays a cube without dark borders around each polygon.

```
%!PS
%%BoundingBox 70 250 550 730
%% Creator: Emery pltps.ftn program
%%EndComments
72 300 div 72 300 div scale
275 1100 translate
 3 setlinewidth
newpath
1045 754 moveto
1261 858 lineto
1262 1206 lineto
1045 1104 lineto
1045 754 lineto
  0.55 setgray
fill
stroke
745 830 moveto
1045 754 lineto
1045 1104 lineto
744 1178 lineto
745 830 lineto
  0.76 setgray
fill
stroke
```

```
1045 1104 moveto
1262 1206 lineto
963 1279 lineto
744 1178 lineto
1045 1104 lineto
  0.34 setgray
fill
stroke
showpage
```

A program giving dark borders is

```
%!
%%BoundingBox 70 250 550 730
%% Creator: Emery pltps.ftn program
%%EndComments
72 300 div 72 300 div scale
275 1100 translate
 0 setlinewidth
newpath
1045 754 moveto
1261 858 lineto
1262 1206 lineto
1045 1104 lineto
1045 754 lineto
gsave
  0.55 setgray
fill
grestore
stroke
745 830 moveto
1045 754 lineto
1045 1104 lineto
744 1178 lineto
745 830 lineto
gsave
  0.76 setgray
fill
grestore
```

```
stroke
1045 1104 moveto
1262 1206 lineto
963 1279 lineto
744 1178 lineto
1045 1104 lineto
gsave
  0.34 setgray
fill
grestore
stroke
stroke
showpage
```

4.3 Arithmetic

Here is a program that does multiplication, defines a string, converts the number to a string, and prints the rotated string after a coordinate system rotation by 45 degrees.

```
%!PS
/c findfont 60 scalefont setfont
300 300 moveto
1234 5678 mul
/s 20 string def
45 rotate
s cvs
show
showpage
```

4.4 Shading Pixels

If we think of pixels as rectangles to be filled, then we may produce a raster image by moving to a pixel location, drawing a rectangle, and filling it with gray. See programs **q90.ftn** and **bm2ps.c**.

```
% cat shadepix.ps
```

```
%!PS
/dx 30 def
/dy 30 def
/s
{ gsave setgray moveto dx 0 rlineto
0 dy rlineto dx neg 0 rlineto closepath
fill stroke grestore} def
0 100   0 s
30 100 .05 s
60 100 .1 s
90 100 .15 s
120 100 .2 s
150 100 .25 s
180 100 .3 s
210 100 .35 s
240 100 .4 s
270 100 .45 s
300 100 .5 s
330 100 .55 s
360 100 .6 s
390 100 .65 s
420 100 .7 s
450 100 .75 s
480 100 .8 s
510 100 .85 s
540 100 .9 s
570 100 .95 s
600 100 1. s
showpage
```

4.5 The Mathematical Word Processing System TeX

TeX is a system created by the famous Stanford computer scientist (mathematician) Don Knuth. It is in the public domain and versions of it may be obtained over the internet. It is the standard language for scientific word

processing and is required for submission to many scientific and mathematical journals. There are also many commercial versions. The most well known for PC's is PCTeX, and PCTeX for windows. The program Scientific Word is a GUI implementation of TeX. The equation editor in the Lotus word processor **AmiPro** is based on TeX. The extended version of the Microsoft Word equation editor, which is called Mathtype, will generate TeX output. TeX is documented in a series of books by Don Knuth called **Mathematical Typesetting**. This document is written in TeX.

4.6 Importing Postscript Figures Into TeX

The **psfig** program may be used to place postscript figures into a TEXfile. The following directory produced by the tex tar tape contains psfig.

```
TeX/contributions/TeXgraphics/psfig18.tar.Z, 227989 bytes, 446 blocks
\end{figure}
\begin{verbatim}
\input{psfig}     (This line at top of file)
\begin{figure}    (These lines located where figure is to appear)
\psfig{figure=filename.ps,height=3.in}
\caption{This is the figure caption.}
\end{figure}
```

The postscript bounding box comment is used to size the figure. The bounding box can be measured from the postscript output of the figure.

4.7 Putting ASCII Text Into Postscript Form With ascii2ps.c

```
%  ascii2ps
 ascii text file to postscript file
 ascii2ps inputfile outputfile fontsize fontnumber [-p,-f]
 fontsize= (points) 6,8,10,....(default 10)
 fontnumber= (1)Times-Roman (2) Helvetica
 fontnumber= (3)Courier (4) Helvetica-Bold (default 1)
 option -p for page numbers (default no page numbers)
 the option -f causes a '.f', starting in the first
```

```
column, to be interpreted as a form feed
```

4.8 HPGL

HPGL is the Hewlett Packard graphics language for pen plotters. Many laser printers will now also accept this language. An HPGL picture may be imported into Microsoft Word. The image is imported by selecting insert—picture in Microsoft word. The file tri.hgl was created from a .gi file using program plthp.c. Here is the file:

```
IN;SP1;
IP0,0,7112,7112;SC0,7112,0,7112;
PA1185 1185 PD;
PA5927 1185 PD;
PA3556 5927 PD;
PA1185 1185 PD;
PU;SP0;
```

Some of the commands are: PA 5927 1185, which means plot absolute, that is, move the pen in the down position to coordinates 5927 1185. PU means pen up. Consult an HPGL manual for more information.

The public domain program printgl may be used to plot hpgl pictures on other devices. For example:

To plot on a laserjet printer:

```
printgl filename.hgl /FL
```

To plot on the VGA screen:

```
printgl filename.hgl /FV
```

For help information type printgl with no arguments.

4.9 GL

GL is the graphics language for Silicon Graphics workstations. Here is a simple GL program:

```
#include <gl/gl.h>
main(){
float v[2];
prefsize(900,900);
winopen("Simple");
ortho2(-2.,2.,-2.,2.);
color(WHITE);
clear();
color(BLACK);
bgnline();
v[0]=-1.;
v[1]=-1.;
v2f(v);
v[0]=1.;
v[1]=1.;
v2f(v);
endline();
sleep(10);
gexit();
return 0;
}
```

prefsize() sets the size of the window in pixels. winopen() opens a window and gives it a name. ortho2() sets up a clipping region mapped to the window. color() sets the drawing color. clear() clears the screen to the drawing color. v2f() maps a 2d floating point vertex. There are similar commands for 3d graphics.

GL is best at graphics dynamics and animation. GL processes 3d polygons using the ZBuffer technique. On the powerfull SGI workstations, transforming, and rendering the polygons is done in hardware, and is very fast. These machines can process many of thousands of triangles between screen paints. The double buffering technique is used for animation, that is, one graphics buffer is being displayed, while a second is being built, then they are swapped.

OpenGL is a computer independent graphics system that is based on SGI GL. OpenGL is different from GL. It runs on many computers. OpenGL is to be the graphics system for Windows NT.

4.10 X Windows

X Windows is a computer independent graphics and windowing system created at MIT. It is in the public domain, and is the basic graphics system used on most **UNIX** workstations.

4.11 Window Mapping

Window and viewport transformations require transformations between rectangles, which are aligned with the coordinate axes. Suppose we have a rectangle defined by a corner point (x_1, y_1) and a diagonally opposite point (x_2, y_2). Suppose we want to map this rectangle to a new rectangle (x_1', y_1') and (x_2', y_2').

We define an x scale factor

$$s_x = \frac{x_2' - x_1'}{x_2 - x_1},$$

and we define a y scale factor

$$s_y = \frac{y_2' - y_1'}{y_2 - y_1}.$$

Then we define the x mapping by

$$f_x(x) = s_x(x - x_1) + x_1'.$$

and the y mapping by

$$f_y(y) = s_y(y - y_1) + y_1'.$$

Clearly this is a linear mapping and

$$f_x(x_1) = x_1',$$

$$f_x(x_2) = x_2',$$

$$f_y(y_1) = y_1',$$

and

$$f_y(y_2) = y_2'.$$

4.12 Contour Plotting

Contour plotting may be done using the program **cntr.ftn**, which produces sequences of small line segments where level planes intersect an approximate triangulated representation of the function surface. Program **l2crvs.ftn** can convert a random set of line segments into connected curves.

4.13 Function Plotting and Graphing

Function plotting can be done with program **pltax.c** which adds labels, titles and axes to a .gi file. We have talked about **pltax.c** previously.

4.14 Geometric Modeling

Geometric modeling is a large area of current research. It concerns the representation of real objects with computer models. Many techniques exist for this representation, including the representation of the mathematical boundary of the objects with mathematical surfaces and patches, and the representation of objects as 3d points sets which are given as Boolean combinations (i.e. using the union, the intersection, and the complement operators of set theory) of primitive sets such as spheres, cubes, cylinders and so on. Mathematical concepts of geometric modeling include the concepts of connectedness and deformable shape (topology), as well as geometrical concepts. Algebraic geometry, projective geometry, differential geometry, geometric design, and computational geometry, all have application to geometric modeling. Some commercial geometric modeling systems include: ACIS, ProEngineer, Catia, IDEAS, and Unigraphics Parasolids.

Chapter 5

Bezier Methods

5.1 The Binomial Theorem and the Bernstein Polynomials

The binomial theorem is

$$(a+b)^n = \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) a^{n-k} b^k.$$

Therefore

$$1 = ((1-x)+x)^n = \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) (1-x)^{n-k} x^k.$$

The kth summand is called the kth Bernstein polynomials of order n. It is written as

$$b_k^n(x).$$

The kth Bernstein polynomial of order n is

$$b_k^n(x) = \left(\begin{array}{c} n \\ k \end{array} \right) (1-x)^{n-k} x^k.$$

It is convenient to define

$$b_k^n(x)$$

to be 0 if $k < 0$ or $k > n$. We can show that a Bernstein polynomial may be computed as a linear combination of lower order Bernstein polynomials. Thus

$$b_i^n(x) = (1-x) b_i^{n-1} + x b_{i-1}^{n-1}$$

Indeed,

$$(1-x)\left(\begin{array}{c} n-1 \\ i \end{array}\right)(1-x)^{n-1-i}x^i + x\left(\begin{array}{c} n-1 \\ i-1 \end{array}\right)(1-x)^{n-1-(i-1)}x^{i-1}$$

$$= \frac{(n-1)!}{(n-i)!i!}(n-i+i)(1-x)^{n-i}x^i = b_i^n(x).$$

Also we see that $b_0^0 = 1$. Given any $n+1$ points in a vector space $P_0, P_1, ..., P_n$, the sum

$$B(x) = \sum_{i=0}^{n} P_i b_i^n(x),$$

is a curve in the space, where the parameter x takes values in the unit interval. This is called a Bezier curve, and the points are called control points. The Bernstein functions form a partition of unity. That is the sum of the Bernstein functions add to one. This is true because

$$1 = ((1-x)+x)^n = \sum_{i=0}^{n}\left(\begin{array}{c} n \\ i \end{array}\right)(1-x)^{n-i}x^i.$$

A set is convex if for every pair of points in the set, the line segment joining the pair of points is also in the set. That is given p_1 and p_2, then for

$$\lambda_1 + \lambda_2 = 1,$$

$$\lambda_1 p_1 + \lambda_2 p_2$$

is in the set. The convex hull of a set is the smallest convex set containing the original set. This is the set of all linear combinations of points of the set for which the coefficients add to one. We see then that the Bezier curve lies in the convex hull of the control points. The shape of the Bezier curve resembles the shape of the control points.

The Bezier curve of degree three is very popular. It has the form

$$b_0^n(t)p_0 + b_1^n(t)p_1 + b_2^n(t)p_2 + b_3^n(t)p_3.$$

Here is a FORTRAN subroutine for computing a cubic Bezier curve:

```
c+ bez3  bezier plane cubic curve
      subroutine bez3(t,px,py,x,y)
c input:
```

```
c  t     variable in the interval [0,1]
c  px,py coordinates of the four control points
c output:
c  x,y   returned point on the bezier curve
      implicit real*8(a-h,o-z)
      dimension px(*),py(*)
      b0=1-t
      x=0.
      y=0.
      b=b0*b0*b0
      x=x+b*px(1)
      y=y+b*py(1)
      b=3.*b0*b0*t
      x=x+b*px(2)
      y=y+b*py(2)
      b=3.*b0*t*t
      x=x+b*px(3)
      y=y+b*py(3)
      b=t*t*t
      x=x+b*px(4)
      y=y+b*py(4)
      return
      end
```

5.2 Transformation Between a Bezier Bases and A Power Basis

Suppose

$$p(x) = \sum_{k=0}^{n} a_k x^k = \sum_{k=0}^{n} c_k b_k^n(x).$$

Proposition. Let

$$A_{ij} = \frac{\begin{pmatrix} i \\ j \end{pmatrix}}{\begin{pmatrix} n \\ j \end{pmatrix}}$$

for $i \geq j$ and zero otherwise. Then A is the change of basis matrix from the power representation to the Bernstein representation. That is

$$c = Aa.$$

Proof.

$$x^i = x^i((1-x)+x)^{n-i} = x^i \sum_{k=0}^{n-i} \binom{n-i}{k} (1-x)^{n-i-k} x^k$$

$$= \sum_{k=0}^{n-i} \binom{n-i}{k} (1-x)^{n-i-k} x^{k+i}$$

$$= \sum_{k=i}^{n} \binom{n-i}{k-i} (1-x)^{n-k} x^k$$

$$= \sum_{k=i}^{n} \frac{\binom{k}{i}}{\binom{n}{i}} \binom{n}{k} (1-x)^{n-k} x^k$$

$$= \sum_{k=i}^{n} \frac{\binom{k}{i}}{\binom{n}{i}} b_k^n(x)$$

$$= \sum_{k=i}^{n} A_{ki} b_k^n(x) = \sum_{k=0}^{n} A_{ki} b_k^n(x).$$

Then we have

$$\sum_{j=0}^{n} c_j b_j^n = \sum_{i=0}^{n} a_i x^i$$

$$= \sum_{i=0}^{n} a_i \sum_{j=0}^{n} A_{ji} b_j^n$$

$$= \sum_{j=0}^{n} \sum_{i=0}^{n} A_{ji} a_i b_j^n.$$

This gives

$$c_j = \sum_{i=0}^{n} A_{ji} a_i.$$

This completes the proof.

By expanding b_i^n we have

$$b_i^k = \sum_{k=i}^{n} (-1)^{k-i} \left(\begin{array}{c} n \\ k \end{array} \right) \left(\begin{array}{c} k \\ i \end{array} \right) x^k.$$

Then letting B be the inverse of A, we have

$$B_{ki} = (-1)^{k-i} \left(\begin{array}{c} n \\ k \end{array} \right) \left(\begin{array}{c} k \\ i \end{array} \right),$$

for $k \geq i$ and zero otherwise.

For $n = 3$ we have

$$b_0^3 = (1-x)^3$$
$$b_1^3 = 3(1-x)^2 x$$
$$b_2^3 = 3(1-x)^1 x^2$$
$$b_3^3 = x^3.$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & 0 & 0 \\ 1 & 2/3 & 1/3 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Thus

$$1 = b_0^3(x) + b_1^3(x) + b_2^3(x) + b_3^3(x)$$
$$x = 1/3 b_1^3(x) + 2/3 b_2^3(x) + b_3^3(x)$$
$$x^2 = 1/3 b_2^3(x) + b_3^3(x)$$
$$x^3 = b_3^3(x)$$

For the inverse B, we have

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}.$$

Then for example

$$b_2^3(x) = 3x^2 + -3x^3.$$

5.3 The Derivative of a Bernstein Polynomial

We shall show that the derivative of the Bernstein polynomial b_k^n is given as a linear combination of two lower order Bernstein polynomials. We shall show that

$$\frac{db_k^n}{dx} = n(b_{k-1}^{n-1} - b_k^{n-1}).$$

In fact we have

$$\frac{db_k^n}{dx} = \left(\begin{array}{c} n \\ k \end{array} \right) [(n-k)(1-x)^{(n-1)-k}x^k + k(1-x)^{(n-1)-(k-1)}x^{k-1}]$$

$$= \left(\begin{array}{c} n \\ k \end{array} \right) \left[kb_{k-1}^{n-1} / \left(\begin{array}{c} n-1 \\ k-1 \end{array} \right) - (n-k)b_k^{n-1} / \left(\begin{array}{c} n-1 \\ k \end{array} \right) \right].$$

Using

$$\left(\begin{array}{c} m \\ j \end{array} \right) = \frac{m!}{j!(m-j)!},$$

and canceling terms, we get the result.

5.4 The Derivative of a Bezier Curve

We have shown that

$$\frac{db_k^n}{dx} = n(b_{k-1}^{n-1} - b_k^{n-1}).$$

Therefore if

$$p = p_0 b_0^n + p_1 b_1^n + ... + p_n b_n^n,$$

then

$$p' = n[p_0(-b_0^{n-1}) + p_1(b_0^{n-1} - b_1^{n-1}) + p_2(b_1^{n-1} - b_2^{n-1}) + ... + p_n(b_{n-1}^{n-1})]$$

$$= n(p_1 - p_0)b_0^{n-1} + n(p_2 - p_1)b_1^{n-1} + ... + n(p_n - p_{n-1})b_{n-1}^{n-1}.$$

The derivative of a Bezier curve can be computed by combining the control points to get a new Bezier curve of lower order. This can be continued to compute higher order derivatives.

5.5 The Representation of a Line Segment

Let a line segment have end points P_0 and P_3. The line segment is

$$(1-x)P_0 + xP_3 =$$

$$[(b_0^3 + b_1^3 + b_2^3 + b_3^3) - (\frac{b_1^3}{3} + \frac{2b_2^3}{3} + b_3^3)]P_0 + [\frac{b_1^3}{3} + \frac{2b_2^3}{3} + b_3^3]P_3 =$$

$$[b_0^3 + \frac{2b_1^3}{3} + \frac{b_2^3}{3}]P_0 + [\frac{b_1^3}{3} + \frac{2b_2^3}{3} + b_3^3]P_3 =$$

$$b_0^3 P_0 + b_1^3 P_1 + b_2^3 P_2 + b_3^3 P_3,$$

where

$$P_1 = \frac{2}{3}P_0 + \frac{1}{3}P_3 = P_0 + \frac{1}{3}(P_3 - P_0),$$

and

$$P_2 = \frac{1}{3}P_0 + \frac{2}{3}P_3 = P_0 + \frac{2}{3}(P_3 - P_0).$$

5.6 The Projective Bezier Curve

A Bezier curve in projective space is defined by

$$c(t) = \sum_{k=0}^{n} P_k b_k^n,$$

for $0 \leq t \leq 1$. Each homogeneous control point P_k is a four dimensional vector. The Euclidean coordinates (Affine coordinates) are given by

$$x(t) = c_1(t)/c_4(t)$$

$$y(t) = c_2(t)/c_4(t)$$

$$z(t) = c_3(t)/c_4(t).$$

The curve defined by these three coordinates is called a rational curve because the coordinates are rational functions of the parameter t.

5.7 An Example: The Circular Arc

Consider a circle of radius 1 and center at $(1, 0)$. It has the equation

$$(x-1)^2 + y^2 = 1.$$

The equation of the line through the origin with slope t has equation

$$y = tx.$$

We will use t as a parameter for our unit circle. Given a line with slope t, let the intersection point (x, y) of the line and the circle correspond to t. Solving these two equations simultaneously we find that

$$x = \frac{2}{1+t^2},$$

and

$$y = \frac{2t}{1+t^2}.$$

Moving the center to the origin we have

$$x = \frac{2}{1+t^2} - 1 = \frac{1-t^2}{1+t^2},$$

and

$$y = \frac{2t}{1+t^2}.$$

Thus as t varies from minus infinity to plus infinity we get every point of the circle except the point $(-1, 0)$. We have essentially a rational parameterization of the unit circle. Unfortunately the parametric interval is not finite and the parameterization is not uniform. So in practice we shall use only a portion of this parameterization.

Suppose we want a rational parametric arc of angle 2θ, where θ is less than π . We may use our parameterization of the unit circle. From our original circle centered at $(1, 0)$ we see that if θ is the angle from the center to the point (x, y), then the slope of the intersecting straight line is

$$t = \tan(\theta/2).$$

So let

$$\alpha = \tan(\theta/2),$$

$$t_1 = -\alpha,$$

and

$$t_2 = \alpha.$$

Then as t varies between t_1 and t_2 we get a circular arc of angle 2θ. We shall represent this arc as a Bezier curve. To this end, we change our parameterization from $[t_1, t_2]$ to $[0, 1]$. Let

$$s = \frac{t - t_1}{t_2 - t_1} = \frac{t + \alpha}{2\alpha}.$$

Solving this for t and substituting in the rational representation for the unit circle we get a rational quadratic function of s.

$$x = \frac{-4\alpha^2 s^2 + 4\alpha^2 s + (1 - \alpha^2)}{4\alpha^2 s^2 - 4\alpha^2 s + (1 + \alpha^2)},$$

$$y = \frac{4\alpha s - 2\alpha}{4\alpha^2 s^2 - 4\alpha^2 s + (1 + \alpha^2)}.$$

Then

$$\begin{bmatrix} p_{0x} \\ p_{1x} \\ p_{2x} \\ p_{3x} \end{bmatrix} = A \begin{bmatrix} 1 - \alpha^2 \\ 4\alpha^2 \\ -4\alpha^2 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} p_{0y} \\ p_{1y} \\ p_{2y} \\ p_{3y} \end{bmatrix} = A \begin{bmatrix} -2\alpha \\ 4\alpha \\ 0 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} p_{0z} \\ p_{1z} \\ p_{2z} \\ p_{3z} \end{bmatrix} = A \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} p_{0w} \\ p_{1w} \\ p_{2w} \\ p_{3w} \end{bmatrix} = A \begin{bmatrix} 1 + \alpha^2 \\ -\alpha^2 \\ 4\alpha^2 \\ 0 \end{bmatrix},$$

The curve c is given by

$$c(s) = p_0 b_0(s) + p_1 b_1(s) + p_2 b_2(s) + p_3 b_3(s).$$

Where

$$p_k = \begin{bmatrix} p_{kx} \\ p_{ky} \\ p_{kz} \\ p_{kw} \end{bmatrix}.$$

If T is an affine transformation then

$$Tc(s) = Tp_0 b_0(s) + Tp_1 b_1(s) + Tp_2 b_2(s) + Tp_3 b_3(s).$$

It follows that the curve C may be transformed by transforming the control points. A general affine transformation has matrix

$$T = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & q_x \\ -\sin(\theta) & \cos(\theta) & 0 & q_y \\ 0 & 0 & 0 & q_z \\ 0 & 0 & 0 & 1/s \end{bmatrix},$$

where θ is a rotation angle about z, s is a scaling, and q is a translation vector. We have converted to bezier coefficients by using our conversion matrix A to convert from the power basis to the Bernstein bases and we then get a set of Bezier control points: p_0, p_1, p_2, and p_3. In the case that our arc has angle π , $\alpha = -1$.

5.8 The Bernstein Polynomial On Interval [a,b]

Let

$$t = \frac{s-a}{b-a}.$$

Then

$$B_i^n(t) = B_i^n((s-a)/(b-a) = \begin{pmatrix} n \\ i \end{pmatrix} \frac{(s-a)^i (b-s)^{n-i}}{(b-a)^n}.$$

We define the ith Bernstein polynomial of degree n, on the interval $[a, b]$, to be

$$B_{i,a,b}^n(s) = \begin{pmatrix} n \\ i \end{pmatrix} \frac{(s-a)^i (b-s)^{n-i}}{(b-a)^n}.$$

5.9 A Bezier Curve as a Special B-Spline

The ith B-spline bases function $N_{i,k,\tau}$, of order k (and degree $n = k-1$), on knot set τ, is recursively defined by the Cox-DeBoor algorithm.

The algorithm is expressed as the equation.

$$N_{i,k,\tau}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} N_{i,k-1,\tau}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} N_{i+1,k-1,\tau}(x).$$

A term is understood to be zero if its denominator is zero. $N_{0,1}(x)$, is a square pulse. It takes a positive constant value for $t_0 \leq x < t_1$ and is zero elsewhere. It is continuous from the right, but not from the left.

Let τ_n be the special knot set $t_0 = t_1 = .. = t_n = a$, and $t_{n+1} = t_{n+2} = ...t_{2n+1} = b$. Thus in the cubic case

$$\tau_3 = \{a, a, a, a, b, b, b, b\}.$$

We claim that the ith B-spline of degree n is the ith Bernstein polynomial. We take the order of the B-spline to be $k = n + 1$. That is the order is the degree plus one. We shall prove that

$$N_{i,k,\tau_n} = B^n_{i,a,b} \chi_{[a,b)}.$$

The characteristic function of a set takes value one on any point in the set, and value zero on any point outside the set. The characteristic function $\chi_{[a,b)}$ of the set $[a, b)$ is equal to 1, if t is in $[a, b)$, and is equal to 0 otherwise.

To prove that the equation is correct, we use induction on the order k. So assume that the equation is true for order $k-1$. We start the induction with the fact that $N_{0,1} = 1 = B^0_{0,a,b}$ on the knot set $\{a, b\}$. So the equation is true for order 1. Using the B-spline recursion formula, and the assumed truth of the equation for order $k-1$, we have

$$\begin{aligned} N_{i,k,\tau_n}(x) &= x N_{i,k-1,\tau_n}(x) + (1-x) N_{i+1,k-1,\tau_n}(x) \\ &= x N_{i-1,k-1,\tau_{n-1}}(x) + (1-x) N_{i,k-1,\tau_{n-1}}(x) \\ &= x B^{n-1}_{i-1,a,b}(x) + (1-x) B^{n-1}_{i,a,b}(x) = B^n_{i,a,b}(x). \end{aligned}$$

Thus the equation is true for order k, if it is true for order $k-1$. This completes the inductive proof.

We shall now establish a result that allows a continuous composite Bezier curve to be written in a simple form, with a simple knot set, as a B-spline

curve. We shall illustrate the property with a specific knot set. Consider the knot set

$$\tau = \{t_0, t_1, ..., t_{10}\} = \{a, a, a, a, b, b, b, c, c, c, c\}.$$

Then

$$\begin{aligned} N_{3,4,\tau} &= \frac{t - t_3}{t_6 - t_3} N_{3,3,\tau} + \frac{t_7 - t}{t_7 - t_4} N_{4,3,\tau} \\ &= \frac{t - a}{b - a} N_{3,3,\tau} + \frac{c - t}{c - b} N_{4,3,\tau} \\ &= \frac{t - a}{b - a} B^2_{3,a,b} \chi_{[a,b]} + \frac{c - t}{c - b} B^2_{4,a,b} \chi_{[b,c]} \\ &= \frac{(t - a)^3}{(b - a)^3} \chi_{[a,b)} + \frac{(c - t)^3}{(c - b)^3} \chi_{[b,c]} \\ &= B^3_{3,a,b} \chi_{[a,b)} + B^3_{4,a,b} \chi_{[b,c]}. \end{aligned}$$

(this needs some checking and editing). A continuous composite Bezier curve with control points $P_0, ..., P_6$, in the interval $[a, c]$ is given by

$$c(t) = (P_0 B^3_{0,a,b} + P_1 B^3_{1,a,b} + P_2 B^3_{2,a,b} + P_3 B^3_{3,a,b}) \chi_{[a,b)}$$

$$+ (P_3 B^3_{0,b,c} + P_4 B^3_{1,b,c} + P_5 B^3_{2,b,c} + P_6 B^3_{3,b,c}) \chi_{[b,c)}.$$

From what we have deduced above, this is equivalent to

$$c(t) = P_0 N_{0,4,\tau} + P_1 N_{1,4,\tau} + ... + P_6 N_{6,4,\tau}.$$

The result can be extended to any continuous composite Bezier curve of degree n.

5.10 Finding a Cubic Bernstein Interpolant

Let $f(t)$ be a function defined on the interval $[a, b]$. We wish to find the coefficients of a cubic interpolant to f, on equally spaced points in $[a, b]$, in the Bezier basis. We shall first specialize and find the coefficients for a cubic polynomial defined on $[0, 1]$. Consider

$$p(x) = \sum_{j=0}^{3} a_j B^3_j(x),$$

where $0 \leq x \leq 1$. Write

$$p_i = p(i/3) = \sum_{j=0}^{3} a_j B_j^3(i/3), i = 0, 1, 2, 3.$$

Solving this system, we find

$$a_0 = p_0$$
$$a_1 = -\frac{5}{6}p_0 + 3p_1 - \frac{3}{2}p_2 + \frac{1}{3}p_3,$$
$$a_2 = \frac{1}{3}p_0 - \frac{3}{2}p_1 + 3p_2 - \frac{5}{6}p_3,$$
$$a_3 = p_3.$$

Let

$$x = \frac{t-a}{b-a}.$$

If $x_i = \frac{i}{3}$ then

$$t_i = a + \frac{b-a}{3}i.$$

Define

$$p_i = p(x_i) = f(t_i).$$

Then by the uniqueness of polynomial interpolation

$$q(t) = p(x(t)),$$

Is the desired cubic interpolant to f. To calculate $q(t)$, first find the corresponding x, and then evaluate $p(x)$.

We could also establish these facts by making use of the equation

$$B_j^3(x(t)) = \begin{pmatrix} 3 \\ j \end{pmatrix} \frac{(b-t)^{3-j}(t-a)^j}{(b-a)^3}.$$

Note that when j of these local interpolants are chained together to give a Bezier piecewise polynomial parameterized on the interval [0,j], equally spaced points in this parameterization, do not necessarily correspond to equally spaced points on a chord length parameterization. This is because the chords are not necessarily of uniform length.

5.11 File Structure

We shall describe a data structure for representing Bezier curves. Let the first row contain a single integers k_1, k_2, k_3. k_1 is the degree. $k_2 = 0$ if each segment ending control point is the same as the next starting control point, and thus appears only once in the file. If $k_2 = 1$ then such control points do not agree in general, which allows for discontinuity of the curve. If $k_3 = 0$ the curve is not rational. It is rational if $k_3 = 1$. Thus the file takes the form

$$k_1 k_2 k_3$$

$$x_1 y_1 z_1 w_1$$

$$x_2 y_2 z_2 w_2$$

$$x_3 y_3 z_3 w_3$$

$$x_4 y_4 z_4 w_4$$

$$x_5 y_5 z_5 w_5$$

.....

$$x_m y_m z_m w_m$$

The number of columns may vary from 1 for functions, to 4 for rational curves in 3-space . The first $k_1 + 1$ rows contain the Bezier control points for the first Bezier curve, the next $n + 1$ rows are the control points for the next Bezier curve (n rows if $k_2 = 0$), and so on. If there are k Bezier curve segments joined together, then they may be parameterized on the interval $[0, k]$ in the obvious way. Note also that the segment chord length is given by the length of the affine vector from the first Bezier control point to the last control point of the segment. So the curve can also be parameterized by chord length.

5.12 The de Casteljau Algorithm

Define $P_{n_0 \ldots n_k}(t)$ to be the point on the Bezier curve, which has control points $P_{n_0}, ..., P_{n_k}$, at parameter t.
Proposition.

$$P_{n_0 \ldots n_k}(t) = (1 - t) P_{n_0 \ldots n_{k-1}}(t) + t P_{n_1 \ldots n_k}(t)$$

proof.

$$(1-t)P_{n_0\dots n_{k-1}}(t) + tP_{n_1\dots n_k}(t) =$$

$$(1-t)\sum_{i=0}^{k-1} B_i^{k-1}P_{n_i} + t\sum_{i=0}^{k-1} B_i^{k-1}P_{n_{i+1}} =$$

$$(1-t)B_0^{k-1}P_{n_0} + (1-t)\sum_{i=1}^{k-1} B_i^{k-1}P_{n_i} + t\sum_{i=1}^{k-1} B_{i-1}^{k-1}P_{n_i} + B_{k-1}^{k-1}P_{n_k} =$$

$$B_0^k P_{n_0} + (1-t)\sum_{i=1}^{k-1}((1-t)B_i^{k-1} + tB_{i-1}^{k-1})P_{n_{i+1}} + B_k^k P_{n_k} =$$

$$\sum_{i=0}^{k} B_i^k P_{n_k} = P_{n_0\dots n_k}(t).$$

This completes the proof.

Define

$$P_j^k = P_{j\dots j+k}.$$

Then the recursion formula becomes

$$P_j^k = (1-t)P_j^{k-1} + tP_{j+1}^{k-1}.$$

We may display this in a table as

$$\begin{array}{cccc} P_0 & & & \\ P_1 & P_0^1 & & \\ P_2 & P_1^1 & P_0^2 & \\ P_3 & P_2^1 & P_1^2 & P_0^3 \\ \dots & \dots & \dots & \dots \end{array}$$

5.13 Subdivision

Proposition. Let

$$0 < a < 1$$

then for $t \in [0,1]$

$$P_{0\ldots k}(at) = Q_{0\ldots k}(t),$$

where

$$Q_0 = P_0(a) = P_0,$$
$$Q_1 = P_{01}(a),$$
$$Q_2 = P_{012}(a),$$
$$\ldots\ldots\ldots$$
$$Q_k = P_{012..k}(a).$$

Proof. Let $k = 1$.

$$Q_{01}(t) = (1-t)Q_0 + tQ_1 =$$
$$(1-t)P_0 + tP_{01}(a) =$$
$$(1-t)P_0 + t((1-a)P_0 + aP_1) =$$
$$(1-ta)P_0 + taP_1 =$$
$$P_{01}(at).$$

Proposition. Let

$$0 < a < 1$$

then for $t \in [0,1]$

$$P_{0\ldots k}(a(1-t)+t) = R_{0\ldots k}(t),$$

where

$$R_0 = P_{0123\ldots k}(a).$$

$$\ldots\ldots\ldots$$
$$R_{k-2} = P_{(k-2)(k-1)k}(a),$$
$$R_{k-1} = P_{(k-1)k}(a),$$
$$R_k = P_k(a) = P_0,$$

Proof.

5.14 The WF-curve as a Bezier curve

The Wilson-Fowler curve (WF-curve) is an interpolating curve. Suppose it passes through the interpolation points

$$P_1, P_2, ..., P_n.$$

Let u_i be a unit vector in the direction of the ith line segment joining the points, that is in the direction of $P_{i+1} - P_i$. Let v_i be the unit vector rotated a positive 90 degrees from u_i. These two vectors form a local coordinate system centered at P_i.

Let ℓ_i be the chord length of the ith segment and let s_i be the accumulated chord length to the ith point. The value $c(s)$ of the Wilson-Fowler curve at a point on the ith segment, whose parameter s satisfies $s_i \leq s \leq s_{i+1}$, is given by

$$c(s) = P_i + (s - s_i)u_i + f_i(s - s_i)v_i,$$

where

$$f_i(x) = \frac{t_i^a x(x - \ell_i)^2 + t_i^b x^2(x - \ell_i)}{\ell_i^2}.$$

Note that

$$\frac{df_i(0)}{dx} = t_i^a,$$

and

$$\frac{df_i(\ell_i)}{dx} = t_i^b.$$

Consider

$$f_i(x) = \frac{t_i^a x(x - \ell_i)^2 + t_i^b x^2(x - \ell_i)}{\ell_i^2}.$$

We shall find the Bezier control points for this curve and then translate and rotate to get the control points for the WF segment. For simplicity of notation we will suppress the i subscript with the understanding that we are dealing with the ith segment of the WF-curve.

Let $w = x/\ell$. Then $f(x) = g(w)$ where

$$g(w) = \ell(t_a + t_b)w^3 - (2t_a + t_b)w^2 + t_a w.$$

Multiplying the column vector of coefficients

$$\ell \begin{bmatrix} 0 \\ t_a \\ -(2t_a + t_b) \\ t_a + t_b \end{bmatrix},$$

by the change of basis matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & 0 & 0 \\ 1 & 2/3 & 1/3 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

we find bernstein coefficient vector

$$\begin{bmatrix} 0 \\ \ell t_a/3 \\ -\ell t_b/3 \\ 0 \end{bmatrix}.$$

These are the y components of the control points. The x components of the control points are

$$\begin{bmatrix} 0 \\ \ell/3 \\ 2\ell/3 \\ \ell \end{bmatrix}.$$

This follows because

$$w = 1/3b_1^3(w) + 2/3b_2^3(w) + b_3^3(w).$$

So the Bezier control points for this curve are

$$Q_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$Q_1 = \begin{bmatrix} \ell/3 \\ \ell t_a/3 \end{bmatrix},$$

$$Q_2 = \begin{bmatrix} 2\ell/3 \\ -\ell t_b/3 \end{bmatrix},$$

and

$$Q_3 = \begin{bmatrix} \ell \\ 0 \end{bmatrix}.$$

Now we shall rotate and translate the curve and thus rotate and translate the control points, so that we obtain the ith WF curve segment. Define

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = P_{i+1} - P_i.$$

Then

$$\ell = \sqrt{d_x^2 + d_y^2}.$$

The proper rotation matrix is

$$\begin{bmatrix} d_x/\ell & -d_y/\ell \\ d_y/\ell & d_x/\ell \end{bmatrix}.$$

After applying the rotation matrix we must translate by P_i. Thus the control points for the ith WF segment are

$$Q_0 = P_i,$$

$$Q_1 = P_i + \begin{bmatrix} d_x/3 - d_y t^a/3 \\ d_y/3 + d_x t^a/3 \end{bmatrix},$$

$$Q_2 = P_i + \begin{bmatrix} 2d_x/3 + d_y t^b/3 \\ 2d_y/3 - d_x t^b/3 \end{bmatrix},$$

and

$$Q_3 = P_i + \begin{bmatrix} d_x \\ d_y \end{bmatrix} = P_{i+1}.$$

Therefore finally we have obtained a Bezier representation of the WF-curve. On the ith WF segment we have

$$c(s) = Q_0 b_0^3(w) + Q_1 b_1^3(w) + Q_2 b_2^3(w) + Q_3 b_3^3(w),$$

where

$$0 \le w = \frac{s - s_i}{\ell_i} \le 1.$$

5.15 Barycentric Coordinates

Given n points in a space $p_1, p_2, ..., p_n$, an $n-1$ dimensional simplex is the set of points

$$\{p : p = \lambda_1 p_1 + \lambda_2 p_2 + ... + \lambda_n p_n\},$$

where each λ_i is between 0 and 1, and they sum to 1. Such a set of points is called an $n-1$ dimensional simplex. The set of λ_i for a point p are called the barycentric coordinates of the point, because if the coordinates are considered mass points at the vertices p_i , then p is the center of mass.

Consider the one dimensional case defined by points p_1 and p_2. Assume $\lambda_1 + \lambda_2 = 1$. Then

$$p = \lambda_1 p_1 + \lambda_2 p_2$$

is a point on the line through the points p_1 and p_2. And if $0 < \lambda_1 < 1$ and $0 < \lambda_2 < 1$ then p is between p_1 and p_2. To prove this we write

$$\begin{aligned} p &= \lambda_1 p_1 + \lambda_2 p_2 \\ &= (1 - \lambda_2) p_1 + \lambda_2 p_2 \\ &= p_1 + \lambda_2((p_2 - p_1). \end{aligned}$$

So P is the sum of vectors p_1 and a multiple of $p_2 - p_1$, and so lies on the line through p_1 and p_2. If $0 < \lambda_2 < 1$ then clearly p is between p_1 and p_2.

Now consider the case of three points p_1, p_2, p_3. If $\lambda_1 + \lambda_2 + \lambda_3 = 1$, then the set of points

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_2 p_2$$

are the points of the plane through p_1, p_2, p_3, (assuming these points are not collinear). Further, if each of the λ_i are between 0 and 1, then we get an interior point of the 2-dimensional simplex (triangle). We may prove this by writing

$$\begin{aligned} p &= \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3 \\ &= (1 - \lambda_2 - \lambda_3) p_1 + \lambda_2 p_2 + \lambda_3 p_3 \\ &= p_1 + \lambda_2((p_2 - p_1) + \lambda_3((p_3 - p_1). \end{aligned}$$

So p is in the plane spanned by the two vectors $p_2 - p_1, p_3 - p_1$ from the origin p_1.

Now suppose $0 < \lambda_i < 1$ for $i = 1, 2, 3$. Then let

$$q = \lambda_1 p_1 + \lambda_1 p_2$$

and

$$r = \frac{q}{\lambda_1 + \lambda_2} = \frac{\lambda_1}{\lambda_1 + \lambda_2} p_1 + \frac{\lambda_2}{\lambda_1 + \lambda_2} p_2.$$

We have

$$\frac{\lambda_1}{\lambda_1 + \lambda_2} + \frac{\lambda_2}{\lambda_1 + \lambda_2} = 1.$$

So r is between p_1 and p_2, that is, on edge $p_1 p_2$. Then

$$p = (\lambda_1 + \lambda_2) r + \lambda_3 p_3.$$

So p is between r and p_3, thus interior to the 2-simplex (triangle).

Proposition. Barycentric coordinates are unique.

proof. Suppose

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3 = \lambda_1' p_1 + \lambda_2' p_2 + \lambda_3' p_3$$

Then

$$p = p_1 + \lambda_2 (p_2 - p_1) + \lambda_3 (p_3 - p_1)$$

and

$$p = p_1 + \lambda_2' (p_2 - p_1) + \lambda_3' (p_3 - p_1).$$

So

$$0 = (\lambda_2 - \lambda_2')(p_2 - p_1) + (\lambda_3 - \lambda_3')(p_3 - p_1).$$

If the coefficients are not zero, then $(p_2 - p_1)$ and $(p_3 - p_1)$ are linearly dependent, and p_1, p_2, p_3 are collinear. Hence the unprimed and primed coordinates are equal.

Now suppose a triangle p_1, p_2, p_3 is projected to the xy plane with a transformation T. T is linear so if

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$$

then

$$Tp = \lambda_1 T p_1 + \lambda_2 T p_2 + \lambda_3 T p_3.$$

By uniqueness p and Tp have the same barycentric coordinates. Hence the barycentric coordinates can be computed on the projected image, provided the projected points are still collinear. Then using these coordinates with the original points $p_1 p_2, p_3$ we can find a point in the triangle, in the original simplex.

This can be used to to order triangles back to front in the z direction for graphics drawing.

Suppose we are given an n-simplex with vertices $v_0, v_1, v_2, ..., v_n$. The barycentric coordinates of a point p sum to one. If the coordinates satisfy

$$0 < \lambda_i < 1,$$

then the point is an interior point of the simplex. If any coordinate is negative, then the point is exterior to the simplex. If

$$0 \leq \lambda_i \leq 1,$$

then the point is in the interior or on the boundary of the simplex. In the case

$$0 \leq \lambda_i \leq 1,$$

when a coordinate $\lambda_j = 0$, the point is on the boundary of the simplex opposite the vertex p_j.

To find the barycentric coordinates we may select an arbitrary vertex, say p_n, and solve the linear system

$$\sum_{i=0}^{n-1} \lambda_i (p_i - p_n) = p - p_n,$$

for $\lambda_0, ..., \lambda_{n-1}$. Since the barycentric coordinates sum to 1, this also determines λ_n.

Let us apply this to the problem of determining that a point is in a triangle of the plane. Suppose we are given the triangle vertices

$$p_1 = (1, 2),$$

$$p_2 = (1, 3),$$

$$p_3 = (2, 3).$$

and wish to determine if $p = (1.5, 2.6)$ is in the triangle. Our linear system is

$$\begin{bmatrix} (1-2) & (1-2) \\ (2-3) & (3-3) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} (1.5-2) \\ (2.6-3) \end{bmatrix}.$$

The solution is

$$\lambda_1 = .1, \lambda_2 = .4$$

Then we compute $\lambda_3 = .5$. Therefore, because all coordinates are between 0 and 1, the point is in the triangle.

The general computation to determine an interior point, requires 11 additions or subtractions, 6 multiplications, 2 divisions, and 3 comparisons. The computation may be done as follows.

Let $a_{11} = x_1 - x_3$,$a_{21} = y_1 - y_3$,$a_{12} = x_2 - x_3$,$a_{22} = y_2 - y_3$, and $b_1 = x - x_3$,$b_2 = y - y_3$. Then letting D be the determinant

$$D = a_{11}a_{22} - a_{21}a_{12},$$

we have

$$\lambda_1 = \frac{b_1 a_{22} - b_2 a_{12}}{D},$$

$$\lambda_2 = \frac{a_{11} b_2 - a_{21} b_1}{D},$$

and

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

See the C procedure **bary2** (There is also a Fortran version).

```
//c+ bary2 barycentric coordinates of a point in the plane
int bary2(double* p,double* p1,double* p2,double* p3,double* lambda){
 double d,a11,a12,a21,a22,b1,b2;
 int i;
 a11=p1[0]-p3[0];
 a21=p1[1]-p3[1];
 a12=p2[0]-p3[0];
 a22=p2[1]-p3[1];
 b1=p[0]-p3[0];
 b2=p[1]-p3[1];
 d=a11*a22-a21*a12;
 if(d == 0.){
  return(0);
 }
 lambda[0]=(b1*a22 - b2*a12)/d;
 lambda[1]=(a11*b2-a21*b1)/d;
 lambda[2]=1.-lambda[0] - lambda[1];
 for(i=0;i<3;i++){
  if((lambda[i] <= - EPSILON) || (lambda[i] >= 1.+ EPSILON)){
   return(0);
  }
 }
 return(1);
}
```

Let us now consider the problem of computing the barycentric coordinates of a point P with respect to a triangle in space with vertices $P_1, P_2 P_3$. Let

$$A_1 = P_1 - P_3,$$

$$A_2 = P_2 - P_3,$$
$$A = P - P_3.$$

We shall set up a system of orthogonal vectors U_1, U_2, U_3 Let

$$B_3 = A_1 \times A_2,$$

$$B_2 = B_3 \times A_1.$$

Let

$$U_1 = \frac{A_1}{\|A_1\|},$$
$$U_2 = \frac{B_2}{\|B_2\|},$$
$$U_3 = \frac{B_3}{\|B_3\|} = U_1 \times U_2.$$

Using the "Back Minus Cab" rule, we have

$$B_2 = -(A_1 \cdot A_2)A_1 + (A_1 \cdot A_1)A_2.$$

Let P' be the projection of P to the plane of the triangle (which is P if P is already in that plane). Then we have

$$P' = (A \cdot U_1)U_1 + (A \cdot U_2)U_2$$

$$P_1 = (A_1 \cdot U_1)U_1 + (A_1 \cdot U_2)U_2$$
$$P_2 = (A_2 \cdot U_1)U_1 + (A_2 \cdot U_2)U_2$$

Then we have the two dimensional case given above where the points P, P_1, P_2, P_3 have coordinates with respect to U_1, U_2

$$x = A \cdot U_1$$

$$y = A \cdot U_2$$

$$x_1 = A_1 \cdot U_1$$

$$y_1 = A_1 \cdot U_2$$

$$x_2 = A_2 \cdot U_1$$

$$y_2 = A_2 \cdot U_2$$

and

$$x_3 = 0$$

$$y_3 = 0$$

The Fortran subroutine **baryt** implements this calculation (there is also C version.)

```
c+ baryt barycentric coordinates of a point relative to a triangle in space
      subroutine baryt(p,p1,p2,p3,b)
c input:
c p        3d point
c p1,p2,p3 3d points of triangle
c output:
c b        barycentric coordinates of the projection
c          of p to the plane of the triangle.
c          projection(p)=b(1)*p1+b(2)*p2+b(3)*p3
c          b(1)+b(2)+b(3) = 1
c          the projection is outside the triangle if
c          and only if some coordinate is negative
c Reference: Computer Graphics and Geometry, Jim Emery graphic.tex
      implicit real*8(a-h,o-z)
      dimension p(*),p1(*),p2(*),p3(*),b(*)
      dimension a(3),a1(3),a2(3),b1(3),b2(3),u1(3),u2(3)
      do i=1,3
       a1(i)=p1(i)-p3(i)
       a2(i)=p2(i)-p3(i)
       a(i)=p(i)-p3(i)
      enddo
      c1=-dotpr(a1,a2)
      c2=dotpr(a1,a1)
      do i=1,3
       b2(i)=c1*a1(i)+c2*a2(i)
      enddo
      c3=dotpr(b2,b2)
      do i=1,3
       u1(i)=a1(i)/sqrt(c2)
       u2(i)=b2(i)/sqrt(c3)
      enddo
      x=dotpr(u1,a)
      y=dotpr(u2,a)
      x1=dotpr(u1,a1)
      y1=dotpr(u2,a1)
      x2=dotpr(u1,a2)
      y2=dotpr(u2,a2)
      d=x1*y2-y1*x2
      b(1)=(x*y2-y*x2)/d
      b(2)=(y*x1-x*y1)/d
```

```
b(3)=1.-b(1)-b(2)
return
end
```

Chapter 6

Graphics Rendering

6.1 Introduction

Graphics rendering is the process of creating surface images by computing pixel color intensities. The two principal methods are polygon shading and ray tracing.

6.2 Z-Buffer Triangle Rendering

Let p_1, p_2, p_3 be the vertices of a triangle. We shall project the triangle from a point q, which is located on the positive Z axis, to the xy plane. Let z_q be the distance from the projection point q to the xy plane. Let point p have coordinates (x, y, z). Then the projected coordinates are

$$x'_p = \frac{z_q x}{z_q - z}$$

$$y'_p = \frac{z_q y}{z_q - z}$$

Let the window in the xy plane be defined by $(x_{min}, x_{max}, y_{min}, y_{max})$. Let n_y be the number of pixels in the y direction and n_x the number of pixels in the x direction. Let i and j be the pixel coordinates. Then

$$x' = x_{min} + ih,$$

$$y' = y_{min} + jk.$$

The first step of the algorithm is to project the vertices of the triangle to the pixel plane. Next we find a box enclosing the projection. Let the pixel coordinates in the box be

$$i_1 \leq i \leq i_2$$

$$j_1 \leq j \leq j_2.$$

Then for each (i, j) in the box, we compute the corresponding coordinates (x', y'). Then we compute the barycentric coordinates with respect to the projected triangle p'_1, p'_2, p'_3. Let these coordinates be $(\lambda_1, \lambda_2, \lambda_3)$. If the barycentric coordinates are between 0 and 1, then the point is in the triangle.

We have

$$p' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \lambda_1 p'_1 + \lambda_2 p'_2 + \lambda_3 p'_3$$

Then also

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$$

The distance from the point p to the xy projection plane is z. If the value of z is less than the value stored in the z-buffer, at coordinate (i, j), then a shade value is calculated. z becomes the new value in the z-buffer. Let N_1, N_2, N_3 be normals at the triangle vertices. The triangle normal for Gouraud shading is computed as

$$N = \lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3.$$

The shade value is then

$$s = N \cdot (q - p).$$

Shade values are stored also in an n_x by n_y array, and become the image.

A general projection point and plane may be transformed to the standard point and plane with an affine transformation. Then the triangle P_1, P_2, P_3, and its normal, must be transformed by this affine transformation before the steps described above are applied. In the following section there is a listing of an implementation of this ZBuffer technique.

6.3 A ZBuffer Program

The program listed here, **t2ps.c**, creates a postscript image of a set of triangles. The triangle file is a set of vertices, each sequence of three vertices defines a triangle. The pixel resolution may be changed by altering the variables ni and nj, which are the number of rows and columns of pixels respectively. One selects a viewing direction by specifying an azimuth angle, and an elevation angle. Azimuth and elevation are the angles defining the view direction (degrees). The azimuth angle is in the xy plane, and is measured) from the x-axis to the projection of the view direction. The elevation angle is measured from the xy plane to the view direction. A zoom factor greater than one causes the picture frame to zoom into the object. Default values of azimuth, elevation, and zoom, are respectively 0., 0., and 1. A sequence of rotations is applied to the triangles do that the viewing direction becomes the Z-axis, so that the ZBuffer algorithm may be applied. See program **trnsf.c** for various matrix routines in the C language. Study the listing for further documentation.

```
/*programname t2ps.c zbuffer, triangles to postscript 6/22/95*/
/*Creator: Jim Emery*/
#include <stdio.h>
#include<ctype.h>
#include <math.h>
#define EPSILON .00001
#define PI 3.14159265358979
unsigned char pix[250000];
short zb[250000];
main (argc,argv)
int argc;
char *argv[];
{
 extern void itoa();
 extern void napoly();
 extern double max3double();
 FILE *in,*out;
 int c,d0,d1,n,k,nf;
 int width,height,nsecs;
 int snum,secstart,secend,wrt;
 char prefix[50],fname[50];
 int ni,nj,maxpixvalue;
 int maxint;
 double x1,y1,z1;
 double x2,y2,z2;
 double x3,y3,z3;
 double xang,zang,zoom;
 double azimuth,elevation;
 double tt[16],r1[16],r2[16],b[4];
 double x[3],y[3],z[3];
 double q[3],p1[4],p2[4],p3[4],pp1[2],pp2[2],pp3[2];
```

```
double normal[3],shade;
double area,zq;
int ishade,i,j,j1,j2,j3,i1,i2,i3;
double lambda[3];
double xmn,xmx,ymn,ymx,zmn,zmx;
double d;
char s1[200];
double xx,yy,zz,xc,yc,zc,dx,dy,dz,sc;
double a[10];
int nr,nt,l;
int imn,imx,jmn,jmx,inside;
ni=500;
nj=500;
maxpixvalue=240;
maxint=32767;
printf(" maxint = %d \n",maxint);
pix[0]=255;
c=pix[0];
printf(" c= %d\n",c);
if(argc < 3){
 printf(" Triangles to postscript file\n");
 printf(" Triangles are defined by sequences of three vertices.\n");
 printf(" For example, these three vertices define a triangle:\n");
 printf(" 0. 0. 0.\n");
 printf(" 1. 0. 0.\n");
 printf(" 1. 1. 1.\n");
 printf(" A triangle file is a sequence of such vertices.\n");
 printf(" Azimuth and elevation are the angles defining the view direction (degrees).\n");
 printf(" The azimuth angle is in the xy plane, and is measured\n");
 printf(" from the x-axis to the projection of the view direction.\n");
 printf(" The elevation angle is measured from the xy plane to the view direction.\n");
 printf(" A zoom factor greater than one causes the picture frame to zoom into the object.\n");
 printf(" Default values of azimuth, elevation, and zoom, are respectively 0., 0., and 1.\n");
 printf(" t2ps triangle_file postscript_file [azimuth elevation zoom]\n");
 exit(1);
}
azimuth=0.;
elevation=0.;
zoom=1.;
if(argc == 4){
 azimuth = atof(argv[3]);
}
if(argc == 5){
 azimuth = atof(argv[3]);
 elevation = atof(argv[4]);
}
if(argc == 6){
 azimuth = atof(argv[3]);
 elevation = atof(argv[4]);
 zoom = atof(argv[5]);
}
printf(" elevation = %g \n",elevation);
printf(" azimuth = %g \n",azimuth);
xang=-(90. -elevation)*PI/180.;
zang=-(azimuth+90.)*PI/180.;
printf(" xang= %g \n",xang);
printf(" zang= %g \n",zang);
```

```
rotz(zang,r1);
rotx(xang,r2);
matrixm(r2,r1,4,4,4,tt);
printf(" matrix= \n");
matrixp(tt,4,4);
in=fopen(argv[1],"r");
if(in==NULL){
 printf(" Can not find the triangle file.");
 exit(0);
}
out=fopen(argv[2],"w");
for(i=0;i<ni;i++){
 for(j=0;j<nj;j++){
  pix[i*nj+j]=255;
  zb[i*nj+j]=0;
 }
}
xmn=1.e30;
xmx=-xmn;
ymn=xmn;
ymx=xmx;
zmn=xmn;
zmx=xmx;
n=0;
while(fgets(s1,200,in) != NULL){
 l=strlen(s1);
 n=n+1;
 nr=reads(s1,a);
 xx=a[0];
 yy=a[1];
 zz=a[2];
 if(xx < xmn)xmn=xx;
 if(xx > xmx)xmx=xx;
 if(yy < ymn)ymn=yy;
 if(yy > ymx)ymx=yy;
 if(zz < zmn)zmn=zz;
 if(zz > zmx)zmx=zz;
}
printf("bounding box: xmn xmx ymn ymx zmn zmx\n");
printf(" %g %g %g %g %g %g\n",xmn,xmx,ymn,ymx,zmn,zmx);
nt=n/3;
printf(" number of triangles = %d \n",nt);
xc=(xmx + xmn)/2.;
yc=(ymx + ymn)/2.;
zc=(zmx + zmn)/2.;
dx=xmx-xmn;
dy=ymx-ymn;
dz=zmx-zmn;
sc=sqrt(dx*dx+dy*dy+dz*dz)/zoom;
/*sc=max3double(dx,dy,dz);*/
printf(" scale = %g \n",sc);
close(in);
in=fopen(argv[1],"r");
/* rewind(in); */
n=0;
xmn=-.5;
xmx=.5;
```

```
ymn=-.5;
ymx=.5;
zmn=-.5;
zmx=.5;
zq=1000.;
while(fgets(s1,200,in) != NULL){
 nr=reads(s1,a);
 b[0]=(a[0]-xc)/sc;
 b[1]=(a[1]-yc)/sc;
 b[2]=(a[2]-zc)/sc;
 b[3]=1.;
 n++;
 if(n == 1){
  trnsf4(tt,b,p1);
 }
 if(n == 2){
  trnsf4(tt,b,p2);
 }
 if(n == 3){
  n = 0;
  trnsf4(tt,b,p3);

  /* compute normals */
  x[0]=p1[0];
  x[1]=p2[0];
  x[2]=p3[0];

  y[0]=p1[1];
  y[1]=p2[1];
  y[2]=p3[1];

  z[0]=p1[2];
  z[1]=p2[2];
  z[2]=p3[2];
  napoly(x,y,z,3,&area,normal);
  shade=normal[2];
  ishade=shade*maxpixvalue;

  /* projection */
  pp1[0]=zq*p1[0]/(zq-p1[2]);
  pp1[1]=zq*p1[1]/(zq-p1[2]);

  pp2[0]=zq*p2[0]/(zq-p2[2]);
  pp2[1]=zq*p2[1]/(zq-p2[2]);

  pp3[0]=zq*p3[0]/(zq-p3[2]);
  pp3[1]=zq*p3[1]/(zq-p3[2]);
  /* find pixel coordinates */
  /* 0 <= j <= nj, 0 <= i <= ni */

  j1 = .5 + (pp1[0]-xmn)*nj/(xmx-xmn);
  i1 = .5 + ni - (pp1[1]-ymn)*ni/(ymx-ymn);

  j2 = .5 +(pp2[0]-xmn)*nj/(xmx-xmn);
  i2 = .5 +ni - (pp2[1]-ymn)*ni/(ymx-ymn);
  j3 = .5 +(pp3[0]-xmn)*nj/(xmx-xmn);
  i3 = .5 +ni - (pp3[1]-ymn)*ni/(ymx-ymn);
```

```
  imn=min3int(i1,i2,i3);
  if(imn < 0 )imn=0;
  imx=max3int(i1,i2,i3);
  if(imx > (nj-1) )imx=(nj-1);
  jmn=min3int(j1,j2,j3);
  if(jmn < 0 )jmn=0;
  jmx=max3int(j1,j2,j3);
  if(jmx > (ni-1) )jmx= (ni-1);
  for(i=imn;i <= imx;i++){
   q[1] = (ni-i)*(ymx-ymn)/ni + ymn;
   for(j=jmn;j <= jmx;j++){
   q[0] = j*(xmx-xmn)/nj + xmn;
    inside=bary2(q,pp1,pp2,pp3,lambda);
    if(inside){
     zz = lambda[0]*p1[2] + lambda[1]*p2[2] + lambda[2]*p3[2];
     k=((zz-zmn)/(zmx-zmn))*maxint;
     if(k > zb[i*nj+j]){
      zb[i*nj+j]= k ;
      pix[i*nj+j]= ishade;
     }
    }
   }
  }
 }
}
/*end of while*/

width = nj;
height= ni;
fprintf(out,"%%!PS\n");
fprintf(out,"%%%%Creator: Jim Emery\n");
fprintf(out,"%%%%Title: Program t2ps.c Triangles to Postscript\n");
fprintf(out,"%%%%CreationDate: \n");
fprintf(out,"%%%%BoundingBox:126 216 486 576\n");
fprintf(out,"%%%%EndComments\n");
fprintf(out,"40 216 translate\n");
fprintf(out,"500.0 500.0 scale\n");
fprintf(out,"/picstr %d string def\n",width);
fprintf(out,"%d %d 8 [%d 0 0 %d 0 %d]",width,height,width,-height,height);
fprintf(out," {currentfile picstr readhexstring pop} image\n");
k=0;
n=0;
while(k < width*height){
 c = pix[k];
 k++;
 /*  c= 255 - c; */
 /* convert to hex */
 d1=c/16;
 d0=c-16*d1;
 if(d1 < 10){
  d1 = d1 + 48;
 }
 else{
  d1 = d1 + 55;
 }
 putc(d1,out);
 if(d0 < 10){
```

```
   d0 = d0 + 48;
  }
  else{
   d0 = d0 + 55;
   }
  putc(d0,out);
  n++;
  if(n == width){
   fprintf(out,"\n");
   n=0;
  }
 }
 fprintf(out,"showpage\n");
 fclose(in);
 fclose(out);
 return(0);
}
/*c+ copys string copy*/
void copys(to,from,pos,num)char *to,*from;int pos,num;
{
char b[256];
strcpy(b,from);
b[pos+num]='\0';
strcpy(to,b+pos);
}
/*c+ itoa integer to string*/
void itoa(n,s)int n;char *s;
{
 char b[256];
 int i,j,k;
 sprintf(b,"%d\0",n);
 k=strlen(b);
 j=0;
 for(i=0;i <= k;i++){
  if(b[i] != ' '){
   s[j] = b[i];
   j = j + 1;
  }
 }
}
/*c+ napoly  unit normal and area of polyhedral face*/
void napoly(x,y,z,n,a,vn)
double x[],y[],z[],*a,vn[];
int n;
{
 extern void crsspr();
 double r1[3],r2[3],dv[3],zero;
 int i,k,j;
 zero = 0.0;
 for(i=0; i<3; i++) vn[i] = 0.0;
 for(k=1;k <= n;k++){
  r1[0] = x[k-1];
  r1[1] = y[k-1];
  r1[2] = z[k-1];
  j = k+1;
  if(j > n) j = 1;
  r2[0] = x[j-1];
```

```
  r2[1] = y[j-1];
  r2[2] = z[j-1];
  crsspr(r1,r2,dv);
  for(i=0; i<3; i++) vn[i] += dv[i];
 }
 *a = 0.0;
 for(i=0; i<3; i++) *a += vn[i]*vn[i];
 if(*a > zero) {
  *a = sqrt(*a);
  for(i=0; i<3; i++){
   vn[i] /=  *a;
  }
 }
 *a /= 2.0;
}
/*
c+ dotpr    scalar product.
*/
double dotpr(a,b)
double a[],b[];
{
 double s;
 int i;

 s = 0.0;
 for(i=0; i<3; i++){

  s = s + a[i]*b[i];
 }
 return s;
}
/*
c+ crsspr  vector cross product.
*/
void crsspr(a,b,c)
double a[],b[],c[];
{
 /*  c=product of a and b */
 c[0] = a[1]*b[2]-a[2]*b[1];
 c[1] = a[2]*b[0]-a[0]*b[2];
 c[2] = a[0]*b[1]-a[1]*b[0];
}
/*c++ max3int maximum of three integers*/
int max3int(i,j,k)int i,j,k;
{
 if(i < j){
  if(j < k){
   return(k);
  }
  else{
   return(j);
  }
 }
 else{
  if(i < k){
   return(k);
  }
```

```
  else{
   return(i);
  }
 }
}
/*c++ min3int minimum of three integers*/
int min3int(i,j,k)int i,j,k;
{
 if(i > j){
  if(j > k){
   return(k);
  }
  else{
   return(j);
  }
 }
 else{
  if(i > k){
   return(k);
  }
  else{
   return(i);
  }
 }
}
/*c++ max3double maximum of three numbers*/
double max3double(i,j,k)double i,j,k;
{
 if(i < j){
  if(j < k){
   return(k);
  }
  else{
   return(j);
  }
 }
 else{
  if(i < k){
   return(k);
  }
  else{
   return(i);
  }
 }
}
/*c++ bary2 barycentric coordinates of a point in the plane*/
int bary2(p,p1,p2,p3,lambda)double *p,*p1,*p2,*p3,*lambda;
{
 double d,a11,a12,a21,a22,b1,b2;
 int i;
/* printf("p= %g %g\n",p[0],p[1]);*/
 a11=p1[0]-p3[0];
 a21=p1[1]-p3[1];
 a12=p2[0]-p3[0];
 a22=p2[1]-p3[1];
 b1=p[0]-p3[0];
 b2=p[1]-p3[1];
```

```
 d=a11*a22-a21*a12;
 if(d == 0.){
  return(0);
 }
 lambda[0]=(b1*a22 - b2*a12)/d;
 lambda[1]=(a11*b2-a21*b1)/d;
 lambda[2]=1.-lambda[0] - lambda[1];
 for(i=0;i<3;i++){
  if((lambda[i] <= - EPSILON) || (lambda[i] >= 1.+ EPSILON)){
   return(0);
  }
 }
 return(1);
}
/*c+ reads get numbers from string*/
int reads(s,a)
char *s;
double *a;
{
 /*
 returned value is:
 n is number of values found
 s-string
 a-array of numbers read
 numbers delimited by blanks
 */
 extern double atof();
 char c[25],d[2];
 int i,l,code,nr;
 strcpy(c,"");
 nr=0;
 l=strlen(s);
 if(s[l-1] == '\n')l=l-1;
 if(l > 0){
  d[0]=' ';
  for(i=0;i<l;i++){
   d[0]=s[i];
   if(d[0] != ' ')strncat(c,d,1);
   if ((d[0]==' ') || (i==l-1)){
    if(strlen(c) != 0){
     nr=nr+1;
     a[nr-1]=atof(c);
     strcpy(c,"");
    }
   }
  }
 }
 return(nr);
}
/*
c+ index position of substring in string
*/
index(s1,sub) char *s1,*sub;
{
/*
 s1=string
 sub=substring
```

```
 value= -1, if sub is not a substring of s1.
         n, if the first occurance of sub in
           s1 starts at character n where
           0 is the first character of s1.
*/
  int i,j,i1,i2,k,p;
  i2=strlen(sub);i1=strlen(s1)-i2;
  if((i2==0) || (i1<0))return(-1);
  i=0;
  do
   {
    p=1;
    for(j=0;(j<i2)&&p;j=j+1)p=(s1[i+j]==sub[j]);
    k=i;
   }while((i++<i1)&&(!p));
  if(!p)return( -1); else return(k);
}
/*c+ trnsf4 four dimensional linear transformation.*/
int trnsf4(t,x,y)double *y,*t,*x;
{
 /* linear transformation of x into y by the 4 by 4 matrix t*/
 /* in homogeneous coordinates. y = tx*/
 int i,j;
 for(i=0; i<4; i++){
  *(y+i) = 0.0;
  for(j=0; j<4; j++){
   *(y+i) += (*(t+i+j*4)**(x+j));
  }
 }
 return(0);
}
/*c+ rotx rotation about y axis.*/
int rotx(ang,r)double ang,*r;
{
 /* r is the 4 by 4 matrix of rotation about the unit x*/
 /* vector by ang radians.  the direction of rotation*/
 /* is determined by the right hand rule:  with the thumb*/
 /* of the right hand pointing in the direction of the*/
 /* vector the fingers determine the positive direction*/
 /* of rotation. The vectors are column vectors.*/
 /*fortran addressing: r_{i,j} = r[i+j*4-1] */
 int i,j;
 double c,s;
 c = cos(ang);
 s = sin(ang);
 for(i=0; i<4; i++){
  for(j=0; j<4; j++){
   *(r+i+j*4) = 0.0;
  }
 }
 *r = 1.0;
 *(r+5) = c;
 *(r+9) = -s;
 *(r+6) = s;
 *(r+10) = c;
 *(r+15) = 1.0;
 return(0);
```

```
}
/*c+ rotz rotation about z axis.*/
int rotz(ang,r)double ang,*r;
{
 /* r is the 4 by 4 matrix of rotation about the unit z*/
 /* vector by ang radians.  the direction of rotation*/
 /* is determined by the right hand rule:  with the thumb*/
 /* of the right hand pointing in the direction of the*/
 /* vector the fingers determine the positive direction*/
 /* of rotation. The vectors are column vectors.*/
 /*fortran addressing: r_{i,j} = r[i+j*4-1] */
 int i,j;
 double c,s;
 c = cos(ang);
 s = sin(ang);
 for(i=0; i<4; i++){
  for(j=0; j<4; j++){
   *(r+i+j*4) = 0.0;
  }
 }
 *r = c;
 *(r+4) = -s;
 *(r+1) = s;
 *(r+5) = c;
 *(r+10) = *(r+15) = 1.0;
 return(0);
}
/*c+ matrixm the product of matrices.*/
int matrixm(a,b,m,n,l,c)int m,n,l;double *a,*b,*c;
{
 /* c=a*b, */
 /* a: m rows by n columns*/
 /* b: n rows by l columns*/
 /* c: m rows by l columns*/
 /*fortran addressing: a_{i,j} = a[i+j*m-1] */
 int i,j,k;
 for(i=0; i<m; i++){
  for(j=0; j<l; j++){
   *(c+i+j*m) = 0.0;
   for(k=0; k<n; k++){
    *(c+i+j*m) += *(a+i+k*m) * (*(b+k+j*n));
   }
  }
 }
 return(0);
}
/*c+ matrixp print a matrix*/
int matrixp(a,m,n)int m,n;double *a;
{
 /*input:  a, m by n matrix*/
 int i,j;
 for(i=0; i<m; i++){
  for(j=0; j<n; j++){
   printf("%11.5g ",a[i+j*m]);
  }
  printf("\n");
 }
```

```
 return(0);
}
```

Chapter 7

Using Graphics Programs

7.1 Using CorelDraw

Drawing should consist of overlayed objects that together form the picture. Greek letters can be found in the symbol font. To put letters and symbols on a diagram, create a character and then duplicate it, and move it to position. A postscript file can be created in two ways. The drawing can be exported to an eps file, and then curves extracted using program **ps2psbz.ftn** However, text curves all seem to be overlaid at the lower left corner. To prevent this, select each text string, and select convert to curves in the arrange menu. The characters will not be filled when ps2psbz converts the file. **ps2psbz.ftn** also produces a p.gi file.

To create curves from text by hand, trace the boundary with line segments. Use node edit rollout to convert each line segment to a curve (convert to curves), select each node and choose smooth to make tangent directions agree on neighboring bezier curves. Alternately this process might be done better in bezier mode.

The second method of creating a postscript file is to select the windows printer **Apple LaserWriter II NT on FILE:**. That is, open **Main**, **Control Panel**, **Printers**, and then print from CorelDraw. The drawing will be written to

```
{\bf d:\ps\p.ps}
```

This windows postcript file will print directly without the **ps2psbz.ftn** conversion. It seems to be a better file than that exported from CorelDraw.

7.2 Printing Graphics

Graphics can be printed on network printers. In windows select **Print Manager**, **Options** , and **Network Connections**. We get the window **NetWare Printer Connections**. Add to the ports list the queues

```
LPT2: \\\DA00_TEK220I_Q
and
LPT3: \\\A01_IIIsi_Q
```

The first printer is a Tektronix Phaser 220I Color Postsript printer located at MC45. The second printer is a Hewlett-Packard Laserjet IIIsi located at MC45. It will print in postscipt mode, or Laerjet mode. This latter printer is called **HORGON** on the **Silicon Graphics Network**. Select one of these printers and select the capture button. Then select permanent to make the selection permanent.

This can be don in DOS also:

```
capture L=2 Q=DA00_TEK220I_Q
```

To see the printer queue status use Novell command **pconsole**.

Also at this location is a big HP PaintJet plotter.

HPGL plots may be imported into Microsoft word, and printed. Alsp .gi files can be converted to HPGL using **plthp.c**. And .gi files can be converted to postscript files using **pltps**.

7.3 Using Autosketch

Drawings created with **Autosketch** can be saved as **DXF** files, and then converted to .gi files using program **dxf.c**.