Computational Techniques for Parametric Curves and Surfaces

Brian A. Barsky

Computer Science Division
Electrical Engineering and Computer Sciences Department
University of California
Berkeley, California 94720

Alain Fournier

Computer Systems Research Group
University of Toronto
Toronto, Ontario
M5S 1A1

RESUME

Les courbes et les surfaces paramétriques sont connues depuis longtemps, et leur utilisation pour représenter les objets en infographie (par opposition au design en CAO) est encore en pleine croissance. Il y a cependant quelquefois des hésitations á les utiliser, car il semble que le gain en puissance ne compense pas pour les difficultés de formulation et de calcul.

Le but de cet article et de rendre plus clair la signification et l'utilisation de ces objets, et de montrer qu'ils ont beaucoup en commun en dépit de l'apparente diversité de leurs formules.

Nous donnons les raisons d'ê tre, les propriétés et les références des courbes ou surfaces d'Hermite, de Coons, de Bézier, des "B-splines" et des "β-splines. Les méthodes de calcul et d'affichage communément utilisées dans les systémes graphiques sont discutées (calculs de points, transformations géométriques, algorithmes pour l'affichage).

Les exemples et les illustrations sont données pour les courbes et les surfaces cubiques.

ABSTRACT

Parametric curves and surfaces have been with us for a long time, and their use for object modeling in Computer Graphics (as opposed to designing in CAD applications) is still growing. There is sometimes, however, a reluctance to use them because it seems that the added power they give is more than offset by the complexity of their formulations and their computations.

The purpose of this paper is to make clearer their meanings and uses, and show how much they have in common behind the diversity of their formulations. The motivations, properties and references for the Hermite, Coons, Bézier, B-spline and β-spline curves or surfaces are given. The computation and display methods common in a standard graphics system are discussed (computations of points, geometric transformations, display algorithms).

The examples and illustrations are given for the curves and surfaces in their cubic form.

1. Introduction

Parametric curves and surfaces have been defined for a long time in mathematics, and used extensively in engineering and more recently in Computer Aided Geometric Design. In Computer Graphics outside of CAD, they have been used to model from simple objects with a few patches to 3-D animation models with several hundred patches. Numerous papers have been published on various algorithms to manipulate, compute and display them.

In spite of all this activity, they still look somewhat forbidding to most people in Computer Graphics. One proof of the dearth of new objects designed with Parametric curves and surfaces in Computer Graphics is the ubiquity of the famous teapot, made of 26 (or is it 28) patches, which appeared in a standard textbook [Newman79], twice on the cover of the CACM [Blinn76, Lane80b], and even in a computer animated film [Carpenter80].

We will review the main formulations, the properties, the computational methods and the display techniques associated with the most common types of parametric curves and surfaces. It is hardly necessary anymore to justify the choice of parametric representations. It allows multiple valued curves or surfaces and it gives independence from the coordinate system. Some of the drawbacks will be mentioned in the rest of this paper, and they have to do with the fact that the relationships between coordinates is only through the parameter(s). By the same token, we do not have to apologize for considering only polynomial formulations. Other functions (with the possible exception of trigonometric functions to represent circles, ellipsoids and spheres), are too costly to compute with little gain in power to justify the cost; and of course theoretically any curve can be approximated to any tolerance by polynomials (with some customary analytical caveats).

In addition to these important factors, other qualities are sought in the formulations for easier use in design and generally better users interface. Consider the simple definition of a parametric m^{th} degree polynomial:

$$Q(u)=a_0+a_1u+a_2u^2+\cdots+a_mu^m \quad (1.1)$$

Note that $Q(u)$ and the a_i's each have x and y (and possibly z) components. Assuming, without loss of generality if we consider a finite span, that the parameter u varies from 0 to 1, the definition of this polynomial requires $m+1$ coefficients

$$a_i, \quad i=0,1,\cdots m$$

Note: in the rest of this paper, in the interest of brevity, we will give a formula only for one component of the points in space if they are similar for all components.

The first choice to make is about the degree (highest power of u with a non-zero coefficient) or the order (number of coefficient, or degree+1) of the polynomial to use. Cubic are generally used in Computer Graphics and other applications because they represent a good compromise between power and complexity. They have the power to allow nonplanar space curves and inflexion points, and curvature continuity between two curves defined by different polynomials, which lower degree polynomial do not have. They are not too complex in the sense that their second derivative is a linear function of u, and the number of operations necessary to evaluate them is not too high. Cubic polynomial will provide most of the examples here, but in most cases extension to other degrees is straightforward.

What coefficients should be chosen for a curve? From equation (1.1), the following equations are readily verified:

$$Q(0)=a_0$$
$$Q(1)=a_0+a_1+a_2+a_3$$
$$Q^{(1)}(0)=a_1$$
$$Q^{(1)}(1)=a_1+2a_2+3a_3$$

Solving for the a_i,

$$a_0=Q(0)$$
$$a_1=Q^{(1)}(0)$$
$$a_2=3(Q(1)-Q(0))+2Q^{(1)}(0)-Q^{(1)}(1) \quad (1.2)$$
$$a_3=2(Q(0)-Q(1))+Q^{(1)}(0)+Q^{(1)}(1)$$

Unfortunately, there is little intuition associated with this set of coefficients. What is needed is a formulation which has a stronger geometric interpretation. It is well known that the set of all polynomial of order M (of degree M-1 or less) is a vector space of dimension M. In other words, any set of four linearly independent polynomials of order M can form a basis for this space.

Most people, especially in Computer Graphics, are familiar with the concepts of vector space, dimensionality, basis and change of coordinate system, mainly as it relates to our two or three dimensional Euclidean space. In Computer Graphics, or dealing with interactive techniques in general, the choice of a basis is not in term of power and generality, but of convenience. Any linearly independent set of vectors has all the generality and power needed, since it can be used to represent any point with the same number of coefficients. So the application and user convenience should dictate which one to chose. For example, it is better, when using three mutually orthogonal vectors as a basis in 3-space, to have one of them pointing away from the center of the planet on which the user stands (technically known as "up"). As another example, if the user picks points in space with a gun (or more pacifically a telescope), then polar coordinates allowing to use azimuth, elevation and distance are more "natural" than Cartesian coordinates.

A similar situation exists, perhaps less obviously, in chosing the right formulation for the parametric curves and surfaces. Taking as an example the family of parametric curves of third degree, the usual power basis (u^0,u^1,u^2,u^3) gives coefficients which, as we have seen from formula 1.2 are not very geometrically informative. The decisive step was taken by Bézier, when

he rearranged the coefficients of the cubic polynomial to use a different basis (the *Bernstein polynomials*) which allowed the coefficients to have a clear, appealing geometric interpretation (see subsection 2.2 for more details).

To show that there is nothing mysterious about it, consider as an exercise the following goals: we want to design a basis for third-degree curves such that the user specifies the two end points, a middle point (middle in parametric space, but anywhere in geometric space), and the tangent vector at this point (the vector defined by the mid-point for origin and another point for extremity). We need to determine the 4×4 matrix M to compute the curve:

$$x(u)=[u^3 u^2 u^1 1]\,[M]\begin{bmatrix} P_0 \\ P_{\frac{1}{2}} \\ T_{\frac{1}{2}} \\ P_1 \end{bmatrix} \qquad (1.3)$$

From the defining vertices (P_0, $P_{\frac{1}{2}}$, $T_{\frac{1}{2}}$, P_1).

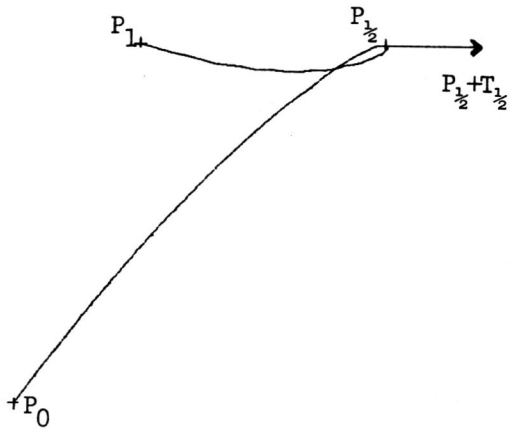

Figure 1.1 Curve and control vertices.

Note that we could rearrange the order of the vertices, and therefore the rows at our convenience.
From the constraints:

$$x(0)=P_0; \quad C(1)=P_1; \quad C(\tfrac{1}{2})=P_{\frac{1}{2}}$$

$$x^{(1)}\ (1/2)=[\ (1/2)^3\ (1/2)^2\ (1/2)^1\ 1]\,M\begin{bmatrix} P_0 \\ P_{\frac{1}{2}} \\ T_{\frac{1}{2}} \\ P_1 \end{bmatrix}$$

we obtain a system of 16 equations with sixteen unknowns, which when solved give the matrix:

$$M = \begin{bmatrix} -4 & 0 & -4 & 4 \\ 8 & -4 & 6 & -4 \\ -5 & 4 & -2 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

To prove that it constitutes a basis, we just have to verify that the determinant is not zero. The useful properties of this formulation, for which it was designed, is that it interpolates the three given points, with explicit control over the tangent at the middle. The user can then control it in a visually obvious way. In other words, the four coefficients of the cubic polynomial have been replaced by something that **mean** something to the user. Figure 1.2 shows how the curve reacts to a change in tangent vector without modifying the other vertices.

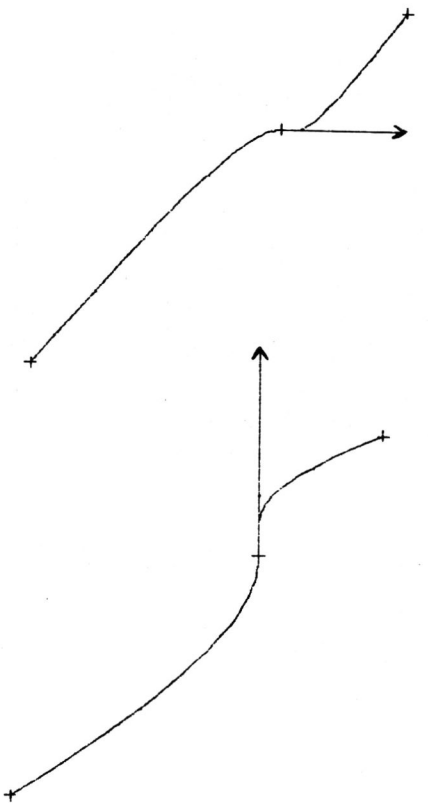

Figure 1.2 Curves with same control vertices but for $T_{\frac{1}{2}}$

Other properties can be useful in applications. They could almost all be described in term of the *no surprise* principle. The usual list is:

-*Convex hull property*. A curve is said to have the convex hull property if it is entirely within the convex hull of the control vertices.
-*Variation diminishing property*. Informally, this means that the curve is "smoother" than the polyline defined by the control vertices.
-*Local control*.The change induced by a change to a control vertex has only a local effect.
-*Continuity*.If, as is mostly the case, a shape is

modelled piecewise from several parametric curves, it is desirable to have the value and first few derivatives equal where the pieces meet. The notation is $C^{[0]}$ for zero order continuity (the curves meet) and in general $C^{[n]}$ for n^{th}-order continuity, that is equality of the n^{th} derivative.

It will be seen in the following sections that these properties are also relevant to the display algorithms. Without proof, we will state that the formulation given does not have the convex hull property, has the variation diminishing property in some sense, and provides global control locally (the concept of global vs local control is of little use if piecewise modeling is used).

$C^{[0]}$ continuity is easily obtained since the curves interpolate their end vertices. The equation for ensuring $C^{[1]}$ (tangent) continuity is :

$$-P_0-4P_{\frac{1}{2}}-2T_{\frac{1}{2}}+5P_1=-5P_0'+4P_{\frac{1}{2}}'-2T_{\frac{1}{2}}'+P_1' \quad (1.4)$$

for two curves defined by: (P_0, $P_{\frac{1}{2}}$, $T_{\frac{1}{2}}$, P_1) and (P_0', $P_{\frac{1}{2}}'$, $T_{\frac{1}{2}}'$, P_1'). This does not give the user an obvious way to control the continuity. Consider, however, the situation (more and more prevalent) where the user choses interactively the control vertices. The system can, in the proper mode, compute the remaining vertices to satisfy $C^{[1]}$ and/or $C^{[2]}$ continuity. If all are chosen, a change in one will force a change in the others. In this case, the "weakest" should be changed (for instance here the tangent, but this can be modified by the user). In this situation, the exact relationship as given in 1.4 is not very important, as long as the changes are predictable.

Having seen how a particular basis can be chosen, we will now examine the motivations and formulations of more traditional parametric curves and surfaces.

2. Curve and Surface Formulations

2.1. Hermite Interpolation and Coons Surfaces

Hermite interpolation is specified by a set of points and derivatives to interpolate. In the cubic case, these derivatives are first derivatives, and the resulting curve has continuity of position and of first derivative ($C^{[1]}$). In particular, let ($P_0,P_1,\ldots,P_m$) be $m+1$ points to be interpolated and ($P_0^1,P_1^1,\ldots,P_m^1$) be the corresponding values of the first derivative vector. Parametrically, the i^{th} curve segment is described as the parameter u varies. Specifically, a curve segment can be written as

$$Q_i(u)=\sum_{j=0}^{1}\sum_{k=0}^{1} g_{jk}(u)P_{i-1+k}$$

The functions $g_{jk}(u)$ are the cubic Hermite basis functions

$$\begin{aligned} g_{00}(t)&=2t^3-3t^2+1 \\ g_{01}(t)&=-2t^3+3t^2 \qquad (2.1) \\ g_{10}(t)&=t^3-2t^2+t \\ g_{11}(t)&=t^3-t^2 \end{aligned}$$

which can be written in matrix form as

$$[g_{00}(t)g_{01}(t)g_{10}(t)g_{11}(t)]=[t^3t^2t\,1][H] \qquad (2.2)$$

where :

$$H=\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

One of the first methods for surface representation was proposed by Coons [Coons67, Forrest72]. The basic idea is to create a surface by blending four boundary curves. A simple Coons surface can be expressed as

$$Q(u,v)=\sum_{i=0}^{1} f_i(u)P(i,v)+\sum_{j=0}^{1} f_j(v)P(u,j) -\sum_{i=0}^{1}\sum_{j=0}^{1} f_i(u)f_j(v)P(i,j)$$

or in matrix form

$$Q(u,v)=[f_0(u)f_1(u)]\begin{bmatrix} P(0,v) \\ P(1,v) \end{bmatrix} +[P(u,0)P(u,1)]\begin{bmatrix} f_0(v) \\ f_1(v) \end{bmatrix} -[f_0(u)f_1(u)]\begin{bmatrix} P(0,0) & P(0,1) \\ P(1,0) & P(1,1) \end{bmatrix}\begin{bmatrix} f_0(v) \\ f_1(v) \end{bmatrix}$$

Here P(u,0), P(u,1), P(0,v), and P(1,v) are the boundary curves; P(0,0), P(0,1), P(1,0), and P(1,1) are the corner points; and $f_0(t)$ and $f_1(t)$ are the blending functions (see Figure 2.1). Note that the blending functions must satisfy $f_i(j)=\delta_{ij}$, where δ_{ij} is the Kronecker delta. This simple Coons surface does not constrain the cross-boundary derivatives; thus, it is not possible to ensure continuity higher than positional when using composite surfaces.

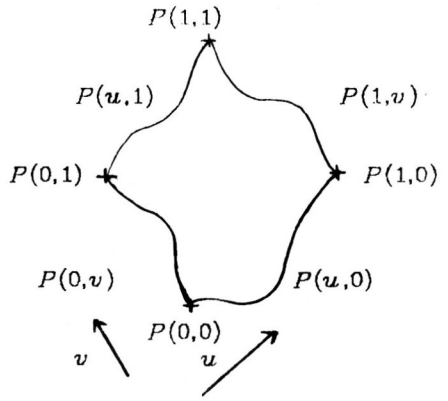

Figure 2.1. Boundary curves and corner points for a Coons surface.

For first derivative continuity this method is extended so that the user is able to specify the cross-boundary derivatives. This requires four blending functions, $g_{00}(t)$, $g_{01}(t)$, $g_{10}(t)$, and $g_{11}(t)$. The surface is now written

$$Q(u,v)=\sum_{i=0}^{1}\sum_{r=0}^{1}P^{(r,0)}(i,v)g_{ri}(u)+\sum_{j=0}^{1}\sum_{s=0}^{1}P^{(0,s)}(u,j)g_{sj}(v)$$
$$-\sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{r=0}^{1}\sum_{s=0}^{1}P^{(r,s)}(i,j)g_{ri}(u)g_{sj}(v)$$

or, in matrix form:

$$Q(u,v)=[g_{00}(u)g_{01}(u)g_{10}(u)g_{11}(u)]\begin{bmatrix}P(0,v)\\P(1,v)\\P^{(1,0)}(0,v)\\P^{(1,0)}(1,v)\end{bmatrix}$$
$$+[P(u,0)P(u,1)P^{(0,1)}(u,0)P^{(0,1)}(u,1)]\begin{bmatrix}g_{00}(v)\\g_{01}(v)\\g_{10}(v)\\g_{11}(v)\end{bmatrix} \quad (2.3)$$
$$-[g_{00}(u)g_{01}(u)g_{10}(u)g_{11}(u)]$$
$$\begin{bmatrix}P(0,0) & P(0,1) & P^{(0,1)}(0,0) & P^{(0,1)}(0,1)\\P(1,0) & P(1,1) & P^{(0,1)}(1,0) & P^{(0,1)}(1,1)\\P^{(1,0)}(0,0) & P^{(1,0)}(0,1) & P^{(1,1)}(0,0) & P^{(1,1)}(0,1)\\P^{(1,0)}(1,0) & P^{(1,0)}(1,1) & P^{(1,1)}(1,0) & P^{(1,1)}(1,1)\end{bmatrix}\begin{bmatrix}g_{00}(v)\\g_{01}(v)\\g_{10}(v)\\g_{11}(v)\end{bmatrix}$$

where

$$P^{(a,b)}(ui,vj)=\frac{\partial^{a+b}P(u,v)}{\partial u^a\partial v^b}\Big|\,u=ui,v=vj$$

While the Coons formulation is useful and very general, it requires the specification of a great deal of data which lack intuitive interpretation. One way to simplify equation (2.3) is to use the following boundary functions:

$$P^{(0,s)}(u,j)=\sum_{i=0}^{1}\sum_{r=0}^{1}P^{(r,s)}(i,j)g_{ir}(u) \quad (2.4)$$
$$P^{(r,0)}(i,v)=\sum_{j=0}^{1}\sum_{s=0}^{1}P^{(r,s)}(i,j)g_{js}(v)$$

Substituting equation (2.4) into equation (2.3), the three terms are now equal, and thus equation (2.3) reduces to

$$Q(u,v)=\sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{r=0}^{1}\sum_{s=0}^{1}P^{(r,s)}(i,j)g_{ri}(u)g_{sj}(v)$$

or in matrix form

$$Q(u,v)=[g_{00}(u)g_{01}(u)g_{10}(u)g_{11}(u)][P]\begin{bmatrix}g_{00}(v)\\g_{01}(v)\\g_{10}(v)\\g_{11}(v)\end{bmatrix} \quad (2.5)$$

where

$$P=\begin{bmatrix}P(0,0) & P(0,1) & P^{(0,1)}(0,0) & P^{(0,1)}(0,1)\\P(1,0) & P(1,1) & P^{(0,1)}(1,0) & P^{(0,1)}(1,1)\\P^{(1,0)}(0,0) & P^{(1,0)}(0,1) & P^{(1,1)}(0,0) & P^{(1,1)}(0,1)\\P^{(1,0)}(1,0) & P^{(1,0)}(1,1) & P^{(1,1)}(1,0) & P^{(1,1)}(1,1)\end{bmatrix}$$

The blending functions have to satisfy

$$g_{0i}(j)=g_{1i}^{(1)}(j)=\delta_{ij}$$
$$g_{1i}(j)=g_{0i}^{(1)}(j)=g_{0i}^{(2)}(j)=g_{1i}^{(2)}(j)=0$$

These conditions are satisfied by the cubic Hermite basis functions which were given in equations (2.1) and (2.2).

2.2. Bézier Curves and Surfaces

Recall the binomial distribution from probability theory and statistics. The probability of exactly i successes in m trials, where the underlying probability of success is u, is

$$B_{i,m}(u)=\binom{m}{i}u^i(1-u)^{(m-i)}$$
where $i=0,1,\ldots,m$ and $0\le u\le 1$. (2.6)

Consider now a *control polygon* formed by the ordered sequence of *control vertices*,

$$[V_0,V_1,\ldots,V_m]$$

The probablility $B_{i,m}(u)$ can be related to these vertices by considering the following game. The player starts at the vertex V_0. With probability u, he or she moves to the next vertex, and with probablility $1-u$ stays at the current vertex. Then $B_{i,m}(u)$ is the probability of being at the vertex V_i after m trials. From this, the expected position after m trials must be

$$Q_m(u)=\sum_{i=0}^{m}B_{i,m}(u)V_i \quad (2.7)$$

In addition, since $B_{i,m}(u)$ is a probability density function,

$$\sum_{i=0}^{m}B_{i,m}(u)=1$$

The set of polynomials $B_{i,m}(u)$ are called *Bernstein polynomials*, and they form the *Bernstein basis* since they are a basis for the vector space of all polynomials with degree at most m. The expression (2.7) for the expected position can also be viewed as a Bernstein approximation to the sequence of control vertices. This expression is a weighted average of the $m+1$ control vertices, with the Bernstein polynomials being the weighting factors, and defines an m^{th} degree Bézier curve [Bézier74]. Note that each polynomial is nonzero over the entire domain $0\le u\le 1$, which is why there is global, not local, control.

Consider now the cubic Bézier curve. This means that $m=3$ and there is a control polygon consisting of the four control vertices $[V_0,V_1,V_2,V_3]$. From equation (2.6), the Bernstein polynomials for this case are

$$\begin{aligned}B_{0,3}(u)&=(1-u)^3=-u^3+3u^2-3u+1\\B_{1,3}(u)&=3u(1-u)^2=3u^3-6u^2+3u\\B_{2,3}(u)&=3u^2(1-u)=-3u^3+3u^2\\B_{3,3}(u)&=u^3\end{aligned} \quad (2.8)$$

These polynomials are plotted for $0\le u\le 1$ in Figure 2.2.

Combining equations (2.7) and (2.8), the Bézier curve is

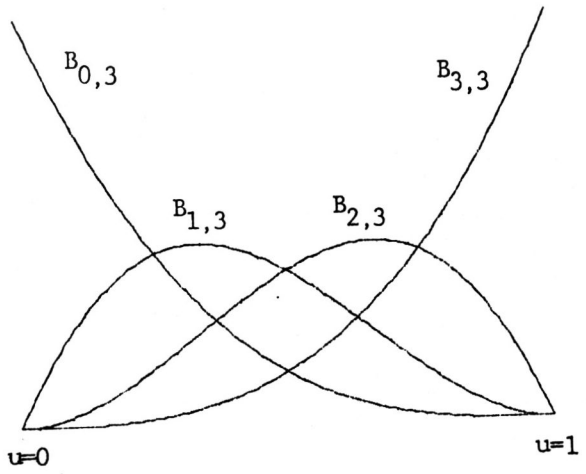

Figure 2.2. The cubic Bernstein polynomials for $0 \le u \le 1$.

$$Q_3(u)=(1-u)^3V_0+3u(1-u)^2V_1+3u^2(1-u)V_2+u^3V_3 \quad (2.9)$$

These equations can be recast in matrix notation. From equation (2.7), the curve can be expressed as

$$Q_3(u)=[B_{0,3}(u)B_{1,3}(u)B_{2,3}(u)B_{3,3}(u)]\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix}$$

From equation (2.8), the polynomials can be written as:

$$[B_{0,3}(u)B_{1,3}(u)B_{2,3}(u)B_{3,3}(u)]=[u^3u^2u\,1][B]$$

where

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

From (2.9), the curve can be rewritten in the following matrix form:

$$Q_3(u)=[u^3u^2u\,1][B]\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix} \quad (2.10)$$

The original motivation in the development of the Bézier formulation was based on the relationship between the derivatives of the polynomial and the edges of the control polygon. From (2.9) or (2.10), it can be readily verified that

$$Q_3=V_0$$
$$Q_3(1)=V_3$$
$$Q_3^{(1)}(0)=3(V_1-V_0)$$
$$Q_3^{(1)}(1)=3(V_3-V_2)$$

This shows a strong relationship between the control polygon and the Bézier curve. The curve begins at the first vertex (V_0) and ends at the last last vertex (V_3) and is tangent to the control polygon at these vertices.

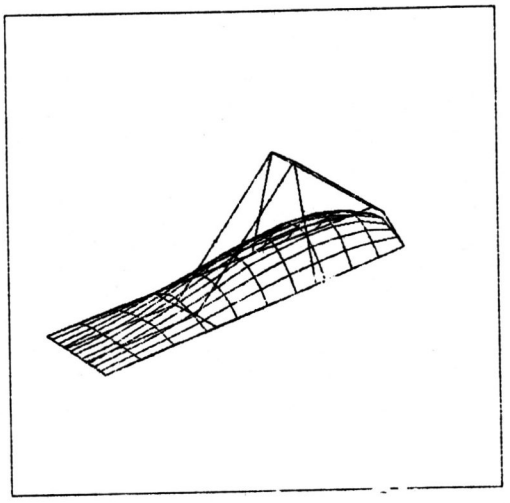

Figure 2.3 A Bézier surface and its control vertices.

A Bézier surface is a tensor product of Bézier curves. It is defined by a set of control vertices, in three-dimensional x-y-z space, which is organized as a two-dimensional graph with a rectangular topology. A point on the surface is a weighted average of these control vertices:

$$Q_{m,n}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}B_{i,m}(u)B_{j,n}(v)V_{ij}$$

or, in matrix form

$$Q_{m,n}(u,v)=[B_{0,m}(u)B_{1,m}(u)\cdots B_{m,m}(u)][V]\begin{bmatrix} B_{0,n}(v) \\ B_{1,n}(v) \\ \vdots \\ B_{n,n}(v) \end{bmatrix}$$

where

$$V=\begin{bmatrix} V_{00} & \cdot & \cdot & \cdot & V_{0n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ V_{m0} & \cdot & \cdot & \cdot & V_{mn} \end{bmatrix} \quad (2.11)$$

In the case of $m=n=3$, this is the bicubic Bézier surface, where the basis functions are those defined in equation (2.8). Comparing the matrix formulations in equations (2.5) and (2.11),

$$HPH^t=BVB^t$$

From this, expressions can be derived for the elements of the P matrix in terms of the control vertices so as to produce an identical surface. Specifically,

$$P=H^{(-1)}BVB^tH^{(-t)}$$

which evaluates to

$$P=\begin{bmatrix} V_{00} & V_{03} \\ V_{30} & V_{33} \\ 3(V_{10}-V_{00}) & 3(V_{13}-V_{03}) \\ 3(V_{30}-V_{20}) & 3(V_{33}-V_{23}) \end{bmatrix}$$

$$\begin{bmatrix} 3(V_{01}-V_{00}) & 3(V_{03}-V_{02}) \\ 3(V_{31}-V_{20}) & 3(V_{33}-V_{32}) \\ 9(V_{00}-V_{10}-V_{01}+V_{11}) & 9(V_{02}-V_{12}-V_{03}+V_{13}) \\ 9(V_{20}-V_{30}-V_{21}+V_{11}) & 9(V_{22}-V_{32}-V_{23}+V_{33}) \end{bmatrix}$$

The following properties of Bézier curves and surfaces should be noted. They have axis-independence, the variation-diminishing property for curves, the convex hull property global (not local) control, and limited ability to ensure continuity between adjacent curves and surfaces.

2.3. B-spline Curves and Surfaces

Splines were first introduced by Schoenberg [Schoenberg46, Curry47, Curry66] and are named from the devices used by draftsmen and shipbuilders to draw curves. A physical spline is used much like a French curve to fair in a smooth curve between specified data points. It is held in place by attaching lead weights called "ducks". By varying the number and position of the ducks, the spline can be forced to pass through the specified data points. A flexible ruler constrained to go through some points will follow the curve which minimizes the strain energy between the points.

If the physical spline is considered to be a thin elastic beam, then the Bernoulli-Euler equation can be invoked. For small deflections, the first derivative term in the curvature expression can be neglected, and thus the curvature can be approximated by the second derivative of the assumed curve. Assuming that the ducks act as simple supports, it can be shown that the solution to this functional calculus problem is a piecewise cubic polynomial, continuous up to its second derivative at the fixed points.

A spline is defined analytically as a set of polynomials over a a knot vector. A knot vector is a vector of real numbers, called knots, in nondecreasing order; that is,

$$u=[u_0,u_1,\ldots,u_q]$$

$$\text{such that } u_{i-1}\le u_i,\ i=1,\ldots,q$$

A spline of order k (degree k-1) is defined mathematically as a piecewise (k-1)'st degree polynomial which is $C^{[k-2]}$ continuous; that is, it is a polynomial of degree at most k-1 on each interval [u sub i-1 , u sub i), and its position and first k-2 derivatives are continuous.

The i'th B-spline basis function of order k (degree k-1) for the knot vector $[u_i,\ldots,u_{i+k}]$ will be denoted $N_{i,k}(u_i,\ldots,u_{i+k};u)$ and can be expressed as the following recurrence relation:

$$N_{i,k}(u_i,\ldots,u_{i+k};u)=\frac{(u-u_i)}{(u_{i+k-1}-u_i)}N_{i,k-1}(u_i,\ldots,u_{i+k-1};u)+\frac{(u_{i+k}-u)}{(u_{i+k}-u_{i+1})}N_{i+1,k-1}(u_{i+1},\ldots,u_{i+k};u)$$

with (2.12)

$$1u_i\le u<u_{i+1}$$

$$N_{i,1}(u_i,u_{i+1};u)=\begin{cases} 1 & u_i\le u<u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

In words, equation (2.12) means that the B-spline of order k in the i'th span is the weighted average of the B-splines of order k-1 on the i'th and (i+1)'st spans, each weight being the ratio of the distance between the parameter and the end knot to the length of the k-1 spans. Note that the computation of $N_{i,k}(u_i,\ldots,u_{i+k};u)$ involves all the knots from u_i to u_{i+k}, but no others, as it should since the width of support is k spans.

Curry and Schoenberg [Curry47] showed that the $N_{i,k}(u)$ are indeed a basis, so that any spline of order k or less defined over a given knot vector, can be expressed as a linear combination of B-spline basis functions defined over the same knot vector extended at both ends by k-1 arbitrary knots.

The only restrictions on the specification of the knot vector are that the same value cannot appear more than k (the order) times and that the knots must be in nondecreasing order. When the same knot value occurs more than once, this is called a **multiple knot**. Specifically, u_i is a knot of multiplicity M if

$$u_i=u_{i+1}=\cdots=u_{i+m-1} \qquad \text{where } M\le k$$

The continuity at this knot is reduced by M-1. Since the continuity at a knot would otherwise be $C^{[k-2]}$, this means that, in general, the continuity at a knot is $C^{[k-M-1]}$, where M is the multiplicity of the knot. For example, a cubic spline (k=4) usually has continuity $C^{[2]}$; a triple knot (M=3) would produce continuity $C^{[0]}$ at that knot. Thus, discontinuities are easily introduced in a spline curve.

Although the values of the knots are so unconstrained, an especially useful special case is that of **uniform** knot spacing, where $u_i = i$ [Barsky82]. For the case k=4, this generates the canonical uniform cubic B-spline basis function :

$$N_{i,4}=\begin{cases} 0 & u<u_i \\ u_0'^3/6 & u_i\le u<u_{i+1} \\ (-3u_1'^3+3u_1'^2+3u_1'+1)/6 & u_{i+1}\le u<u_{i+2} \\ (3u_2'^3-6u_2'^2+4)/6 & u_{i+2}\le u<u_{i+3} \\ (1-u_3'^3)/6 & u_{i+3}\le u<u_{i+4} \\ 0 & u_{i+4}\le u \end{cases}$$

where $u_j'=u-u_{i+j}, j=i,i+1,\ldots,i+3$

An important observation is that the shape of these basis functions are identical, independent of i; that is, all the $N_{i,k}(u)$ are translates of each other.

From the basis functions it can be noted that there are less than k nonzero basis functions at the extreme values of u. In order to consistently have k nonzero basis functions (except at the knots themselves), a slightly modified version of the above knot vector is used. This knot vector has uniform knot spacing with the first and last knot value each repeated k times.

This case closely resembles the behaviour of Bernstein polynomials (Bézier curves), and if no interior knots are presented in the knot vector, the B-splines specialize exactly to Bernstein polynomials. The corresponding knot vector is:

$$[\underbrace{00\cdots0}_{k \text{ times}}\ \underbrace{11\cdots1}_{k \text{ times}}]$$

To see this, note that equation (2.12) reduces to

$$N_{i,k}(u)=uN_{i,k-1}(u)+(1-u)N_{i+1,k-1}(u)$$

where

$$N_{i,1}(u)=\begin{cases}1 & u_i \le u < u_{i+1}\\ 0 & \text{otherwise}\end{cases}$$

which is the recurrence relation for the Bernstein polynomials.

As with Bernstein polynomials and Bézier curves, B-spline basis functions can be used to approximate a sequence of control vertices. This expression is again a weighted average of control vertices; specifically,

$$Q_k(u)=\sum_{i=0}^{m} N_{i,k}(u)V_i$$

where the knot vector is:

$$[u_0,u_1,\ldots,u_q]$$

Since there are m+1 control vertices in the control polygon, and each control vertex has a corresponding basis function, there are m+1 basis functions. Moving through the knot vector, each basis function is nonzero over a successive set of k+1 knots. Thus, k+m+1 knots define m+1 basis functions which correspond to the m+1 control vertices. From this, it can be seen that the uniform knot vector with multiple end knots is

$$[\underbrace{00\cdots0}_{k-1}\ \underbrace{01\cdots r}_{m-k+2}\ \underbrace{rr\cdots r}_{k-1}]$$

where $r=m-k+2$. That is,

$$u_i=\begin{cases}0 & i=0,\ \ldots,\ k-2\\ i-k+1 & i=k-1,\ \ldots,\ m+1\\ m-k+2 & i=m+2,\ \ldots,\ m+k\end{cases}$$

In the same manner that a Bézier surface was formed from Bézier curves, a B-spline surface is a tensor product of B-spline curves

$$Q_{k,l}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,k}(u)N_{(j,l)}(v)V_{ij}$$

Like Bézier curves, B-splines have axis-independence, the variation-diminishing property for curves, and the convex hull property. In addition, B-splines have the advantages of local control (since each B-spline basis function is nonzero on only k spans or $k \times k$ surfaces) and ease of maintaining high order continuity. The formulation of B-splines curves or surfaces can be given in a manner similar to equation 2.10:

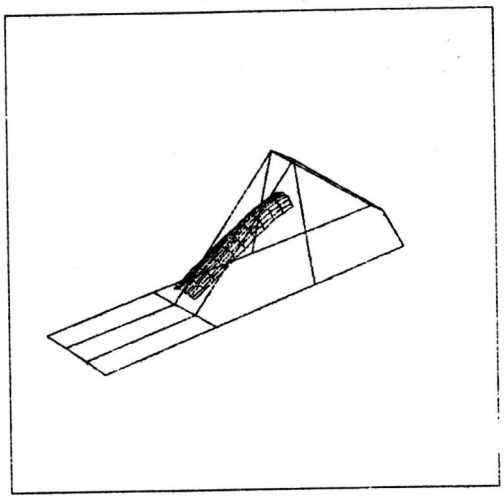

Figure 2.4 B-spline surface and control vertices.

$$x(u)=[u^3u^2u\,1][S]\begin{bmatrix}V_0\\V_1\\V_2\\V_3\end{bmatrix}$$

where :

$$S=\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 3 & 0\end{bmatrix}$$

And for the surface:

$$x(u)=[u^3u^2u\,1][S]\,[V]\,\,[S]^T\begin{bmatrix}v^3\\v^2\\v\\1\end{bmatrix}$$

where [S] is as above, and [V] is the matrix of control vertices as in (2.11).

2.4. β-spline Curves and Surfaces

The β-spline [Barsky81b] is a new mathematical technique for curve and surface representation that has been developed expressly for geometrical and graphical applications. Interaction with the user is via *control vertices* and *shape parameters*, while the underlying mathematical formulation is based on the constraints of continuous unit tangent and curvature vectors. These fundamental geometric measures are more appropriate than traditional algebraic ones based on derivatives. The use of geometric measures also adds degrees of freedom that can be captured to provide further control of shape via two inherent shape parameters, β_1 and β_2, that are related to tension. The β-spline representation also has the important advantage of *local control*.

A β-spline curve or surface is specified by a set of *control vertices*. A point on the i^{th} curve segment is a weighted average of the four control vertices $V_{i+r}, r = -2,-1,0,1$. The coordinates of the point $Q_i(u)$ on the i^{th} curve segment are then given by

$$Q_i(u) = \sum_{r=-2}^{1} b_r(\beta_1,\beta_2;u)\, V_{ir} \text{ for } 0 \le u < 1.$$

As the domain parameter u varies from zero to unity the i^{th} curve segment is traced out.

The weighting factors are the scalar-valued *basis functions* evaluated at some value of the domain parameter u, and of each shape parameter β_1 and β_2.

The β-spline basis functions were derived in [Barsky81b]. They are

$$b_{-2}(\beta_1,\beta_2;u)=2\beta_1^3(1-u)^3/\delta$$

$$b_{-1}(\beta_1,\beta_2;u)=[2\beta_1^3u[u^2-3u+3]$$
$$+2\beta_1^2[u^3-3u^2+2]+2\beta_1[u^3-3u+2]+\beta_2[2u^3-3u^2+1]]/\delta$$

$$b_0(\beta_1,\beta_2;u)=[2\beta_1^2u^2[3-u]+2\beta_1u[3-u^2]$$
$$+\beta_2u^2[3-2u]+2(1-u^3)]/\delta$$

$$b_1(\beta_1,\beta_2;u)=2u^3/\delta$$

where $\delta=2\beta_1^3+4\beta_1^2+4\beta_1+\beta_2+2$

A point on the $(i,j)^{th}$ β-spline surface patch is a weighted average of the sixteen control vertices $V_{i+r,j+s}$, $r = -2,-1,0,1$, and $s = -2,-1,0,1$. The mathematical formulation for the surface $Q_{ij}(u,v)$ is then

$$Q_{ij}(u,v) = \sum_{r=-2}^{1}\sum_{s=-2}^{1} b_r(\beta_1,\beta_2;u)\, V_{i+r,j+s}\, b_s(\beta_1,\beta_2,v)$$

for $0 \le u < 1$ and $0 \le v < 1$.

3. Computational Methods

3.1. Introduction

It is important to distinguish the different kinds of primitives used through a graphics system. The terminology is not well standardized yet, and the boundaries still shifting. We will use here the following definitions:

> *modeling primitives:* the building blocks for the objects in the modeling systems. For example, they can be solids in a mechanical CAD system, spheres in a molecule model, parametric surfaces in a shape design system.
> *graphic primitives:* the primitives used at the graphics package level, on which the viewing transformations, the clipping and the shading is performed.
> *output primitives:* the entities recognized by the output device (instructions to the display processor unit). They can be lines, points, pixels in a "dumb" frame buffer, or more complex entities like filled polygons, or filled circles.

If all the graphics systems used parametric curves or surfaces as both graphic and output primitives, the rest of this paper would be much shorter. Since no system uses parametric curves and surfaces as output primitives, and few use them as graphic primitives, we have to consider algorithms to convert parametric curves and surfaces to the commonly used primitives. Since most primitives are defined by a finite set of vertices (points in 2 or 3 dimensions) we will start by looking at methods to evaluate a point on a parametric curve or surface.

3.2. Matrix computation

A point, for a given value of the parameter u, can be computed directly from the matrix representation of the parametric curve:

$$x(u)=[u^3u^2u^1 1]\,[M]\begin{bmatrix}V_0\\V_1\\V_2\\V_3\end{bmatrix}$$

and similarly for the other coordinate(s). A direct evaluation from this formula takes 2 multiplications for the row vector, 12 multiplications and 12 additions for the vector-matrix multiplication, and 4 multiplications and 3 additions for the final multiplications by the vertices. This is a total of 18 multiplications and 15 additions. For the case of the surfaces:

$$x(u,v)=[u^3\ u^2\ u^1\ 1]\,[M]\,[V]\,[M]^T\begin{bmatrix}v^3\\v^2\\v^1\\1\end{bmatrix}$$

this adds up to 51 multiplications and 39 adds. If the control vertices are known, and the computation is done for a large number of points, then the multiplication $[M]\,[V]$ or for the surface $[M]\,[V]\,[M]^T$ can be done in preprocessing, at a cost of 16 multiplications and 12 adds, and then for each point evaluation only 5 multiplications and 3 adds are necessary. For a surface, the corresponding figures are 32 multiplications and 24 adds in preprocessing, and 19 multiplications and 15 adds per point evaluation. These numbers have to be multiplied by the number of coordinates.

If vector or matrix multipliers are available, these could lead to a significant improvement in speed [England78]. It is interesting to note that 4 element vector multipliers, and 4 × 4 matrix multipliers are available because of the use of 4 × 4 homogeneous coordinate transformation matrices. It makes cubic equations even more appropriate. If a circuit is available that multiplies and sums 4 pairs of numbers in parallel, then the point evaluation takes 5 steps, once the vector $[u^3\ u^2\ u^1\ 1]$ is computed. In the surface case it takes 13 steps. If the vertices are known in advance, these numbers become 1 step and 5 steps, respectively. The evaluation of a point can then take a time of the order of a μsecond.

3.3. Polynomial Evaluation

If we rearrange slightly the order of computation, we have

$$[A]=[M]\begin{bmatrix}V_0\\V_1\\V_2\\V_3\end{bmatrix}$$

and we obtain an ordinary third degree polynomial:

$$x(u)=[u^3\ u^2\ u^1\ 1]$$

Applying Horner's rule, this could be rewritten:

$$x(u)=((a_3u+a_2)u+a_1)u+a_0$$

After the preprocessing (the computation of $[A]$) the number of operations is then 3 multiplications and 3 additions (compare with 5 multiplications in the matrix approach). But the price to pay is a certain loss of regularity in the operations. Note that since we are dealing with small size polynomials and matrices, more elaborate methods to lower the asymptotic complexity like Fast Fourier Transform for polynomial evaluation are not appropriate. However, methods like Winograd's [Knuth69] or Strassen's can be used to save on matrix multiplication even on 4 × 4 matrices.

3.4. Table lookup

Often in practice, for reasons elaborated upon in section 6, what is needed is a series of points regularly spaced in parametric space. In this case a table lookup technique allows substantial savings in computation. For each value u_i of u needed, the value of the vector $[u^3\ u^2\ u^1\ 1]\ [M]$ is computed and stored in a table. Then to compute $x(u_i)$, the vector is retrieved from the table using i as an index. Then only 3 multiplications and 3 adds remain. If the vertices are known in advance, then of course the final value itself is stored in the table. This becomes equivalent to precomputing a set of points on the curve. In the surface case, only one table is needed for u and v, if the same number of subdivision is needed for both. It is because

$$[u^3\ u^2\ u^1\ 1]\,[M] \text{ and } [M]^T\begin{bmatrix}v^3\\v^2\\v^1\\1\end{bmatrix}$$

are transposed of each other when $u=v$. In this case, the computations remaining are 15 multiplications and 15 adds.

3.5. Forward Differencing Techniques.

When the whole surface is computed, or significant portions of a whole surface, at equally spaced points, then it makes sense to exploit the regularity of the computation to save on operations. In other contexts, the same concept is known in computer graphics as *coherence* (in this case "object coherence"). An old and well known technique is called forward differencing [Ralston65]. It uses the fact that the "n^{th} difference" of a n^{th} degree polynomial is constant. The forward difference of a function $f(u)$ is:

$$\Delta f(u)=f(u+h)-f(u)$$

The second difference is:

$$\Delta^2 f(u)=\Delta f(u+h)-\Delta f(u)$$

and so on to the n^{th} forward difference :

$$\Delta^n f(u)=\Delta^{n-1} f(u+h)-\Delta^{n-1} f(u)$$

In the case of a third degree polynomial:

$$x(u)=a_3u^3+a_2u^2+a_1u+a_0$$

$$\Delta x(u)=3a_3u^2h+(3a_3h^2+2a_2h)u+a_3h^3+a_2h^2+a_1h$$

$$\Delta^2 x(u)=6a_3h^2u+6a_3h^3+2a_2h^2$$

$$\Delta^3 x(u)=6a_3h^3$$

which is a constant for a given h.

From the original definition of the forward difference, we can compute $f(n+1)h)=f(nh)+\Delta f(nh)$.

We now can use the following algorithm: We initially compute

$$x(0),\ \Delta x(0),\ \Delta^2 x(0),\ \text{and } \Delta^3 x(0)$$

for a given h. Since $\Delta^3 x(u)$ is independent of u, we will write it $\Delta^3 x$. Then for $n=1$ to $1/h$ we compute:

$$x((n+1)h)=x(nh)+\Delta x(nh)$$

$$\Delta x((n+1)h)=\Delta x(nh)+\Delta^2 x(nh)$$

$$\Delta^2 x((n+1)h)=\Delta^2 x(nh)+\Delta^3 x$$

For each new point only 3 additions are needed. This method has the drawback of having the roundoff errors accumulating during the computations, which can make the last points computed quite inaccurate. To extend the technique to surfaces we have to compute the forward differences of $x(u,v)$ twice, once as a polynomial of u (v constant) and once as a polynomial of v (u constant).

But each forward difference is a polynomial in both u and v , and as such has a forward difference for both. We then have a 4×4 matrix of forward differences.

Starting with $x(0,0)$, the next values $x(h,0)$, $x(2h,0)$, ··· can be computed, using the forward differences for u. Then $x(0,h)$, $x(0,2h)$... can be computed using the forward differences for v. Then $x(0,h)$, $x(h,h)$, $x(2h,h)$... can be computed.

At each step, each row (if we go in the u direction) or column (if we go in the v direction) has to be updated by replacing it by the sum of itself and the next row or column. Each new point then costs 12 additions.

3.6. Recursive Subdivision

In the previous approach, the question was given $x(u)$, compute $x(u+h)$. This is an incremental approach. If instead we ask given $x(u_1)$ and $x(u_2)$, compute $x(u)$ with $u_1<u<u_2$, this is a subdivision approach. It also has a strongly hierarchical approach, since $x(u)$ can properly be viewed as a descendant of $x(u_1)$ and $x(u_2)$. In particular, if $u=\frac{u_1+u_2}{2}$, it is have a mid-point subdivision, which can be much simpler to compute.

Catmull [Catmull74] first introduced this technique for parametric curves and surfaces.

As an example take the cubic polynomial:

$$x(u)=a_3u^3+a_2u^2+a_1u+a_0$$

Assume we know the values of $x(u+d)$ and $x(u-d)$. It is easy to show that:

$$x(u)=\frac{x(u+d)+x(u-d)}{2}-(a_2+3a_3u)d^2$$

So the value at the mid-point $x(u)$ is the average of the values at the end points, minus a correction factor:

$$C(u,d)=(a_2+3a_3u)d^2=x^{(2)}(u)\frac{d^2}{2}$$

The last equality can be verified by computing the second derivative. The correction term can itself be computed recursively by the same method, and the the cubic case it is a linear function of u which does not need any correction term. The algorithm for subdivision is then first to compute ($d=1$):

$$x(0),\ x(1),\ C(0,1)=\frac{x^{(2)}(0)}{2},\ C(1,1)=\frac{x^{(2)}(1)}{2}$$

then in each following steps to compute ($d=0.5$):

$$C(0.5,0.5)=\frac{C(0,1)+C(1,1)}{8}$$

$$x(0.5)=\frac{x(0)+x(1)}{2}-C(0.5,0.5)$$

and the new correction terms:

$$C(0,0.5)=\frac{C(0,1)}{4},\quad C(1,0.5)=\frac{C(1,1)}{4}$$

So to compute each new midpoint involves 3 adds (one to compute the new correction factor) and 8 shifts. This is the fastest way to compute a new point on the curve.

Another way to look at the subdivision of a surface/curve, is : given a set of control points, generate 2 sets of control points that together generate the same curve as the original set. Pictorially the answer is given for the Bézier curve by the geometric construction for the case $u=0.5$. If the original vertices are (V_0, V_1, V_2, V_3) and the two new sets of control vertices are (Q_0, Q_1, Q_2, Q_3) and (R_0, R_1, R_2, R_3) then:

$$Q_0=V_0 \qquad R_0=Q_3$$

$$Q_1=\frac{V_0+V_1}{2} \qquad R_1=\frac{V_1+V_2}{4}+\frac{R_2}{2}$$

$$Q_2=\frac{Q_1}{2}+\frac{V_1+V_2}{4} \qquad R_2=\frac{V_2+V_3}{2}$$

$$Q_3=\frac{Q_2+R_1}{2} \qquad R_3=V_3$$

This subdivision method requires 6 adds and 6 shifts per point. It has the advantage of creating subsurfaces, keeping the nature of the primitive used. A similar method can be applied to the control vertices of a B-spline curve [Lane80a].

The preceding subdivision method can be applied recursively without in effect ever computing the points on the curve itself. The proof that the series of computed control vertices converges to the curve is easy once the convex-hull property is applied. The more difficult problem of subdividing arbitrary B-splines, including non-uniform ones, has been solved by the "Oslo" algorithm [Cohen80]. It also computes new control vertices, but can be used to split the curves at arbitrary parameter values.

4. Viewing Transformations

4.1. Rotation, Translation and Scaling

In this section, we discuss parametric curves and surfaces as graphic primitives, that is as entities that are subject to geometric computations in general, and transformations in particular. The viewing transformations, which are linear transformations, and generally expressed as 4x4 matrix using homogeneous coordinates [Foley82, Newman79], can always be expressed as a concatenation of rotation, translation and scaling matrices.

Similar to the viewing transformations, the instance transformations (going from the *master* coordinate system to the word coordinate system) can also be represented as a concatenation of these basic transformation matrices.

It is easy to show that for rotation and scaling, transforming the points on the curve or surface is equivalent to transforming first the control matrix, and then using this transformed matrix to compute the points.

$$[x'(u)y'(u)z'(u)]=[x(u)\ \ y(u)\ \ z(u)][T]$$

where the primed coordinates are for the transformed points, and [T] is the 3x3 transformation matrix. Replacing the $x(u)$,$y(u)$ and $z(u)$ by their formulation in a cubic curve:

$$[x'(u)y'(u)z'(u)]=[u^3u^2u\ 1][M]\begin{bmatrix}V_{x0}&V_{y0}&V_{z0}\\V_{x1}&V_{y1}&V_{z1}\\V_{x2}&V_{y2}&V_{z2}\\V_{x3}&V_{y3}&V_{z3}\end{bmatrix}[T]$$

$$=[u^3u^2u\ 1][M][T']$$

where:

$$[T']=\begin{bmatrix}V'_{x0}&V'_{y0}&V'_{z0}\\V'_{x1}&V'_{y1}&V'_{z1}\\V'_{x2}&V'_{y2}&V'_{z2}\\V'_{x3}&V'_{y3}&V'_{z3}\end{bmatrix}$$

The translation transformation is usually reduced to a multiplication by a transformation matrix by using homogeneous coordinates, in essence reducing point translation in 3-D to vector transformation in 4-D. See in particular [Riesenfeld81a] about the nature of the homogeneous coordinates.

We can by extension apply the same technique to the control vertices matrix. In the case where some of control vertices are vectors, however, and not points in 2-D or 3-D space, then the usual extension to homogeneous coordinates by adding 1 as the fourth component is not valid. To keep them unaffected by translation, as they should be, the fourth component should be 0 (in other words, the last row of the transformation

matrix is not used). Care should be taken, in this case, not to try to divide the first 3 components by the last, as is done after the perspective transformation (see below).

4.2. Perspective Transformation

In the classic perspective transformation after the objects are transformed to the eye coordinate system (with the eye at the origin), the new x and y coordinates are obtained by dividing the x and y coordinates by the z coordinate:

$$x_s = \frac{D\ x_e}{S_x\ z_e} \quad y_s = \frac{D\ y_e}{S_y\ z_e}$$

where D is the distance eye-window, and S_x, S_y the window size in the x and y direction. It can be shown easily that the transform of a straight line segment is a straight line segment (but they do not have the same parametrisation).

In the case of a cubic curve or surface, the transform

$$x_s(u) = \frac{x_e(u)\ D}{z_e(u)\ S_x} \quad y_s(u) = \frac{y_e(u)\ D}{z_e(u)\ S_y}$$

is not a cubic polynomial. Therefore it is not rigorously correct to apply the perspective transformation on the control vertices and then compute the surface from the result. Fortunately, in most circumstances the difference will not be visually detectable, and this is commonly done.

Before leaving the subject of transformations, clipping should be mentioned. Simple tests to verify if the whole surface surface is in or out are possible if the formulation used has the *convex hull property* since in this case if all the control vertices are in (out) then the whole surface is in (out). In the case where some vertices are in and other out, then more elaborate tests are necessary, and one should use some of the techniques described in the next section on display algorithms.

5. Display Algorithms

5.1. Polygonal Approximations

The display algorithms can have two purposes: to transform the parametric curves and surfaces into graphic primitives, or to transform them into output primitives. In the first case, perhaps they should no be called display algorithms. To display the graphic primitives, they in turn have to be converted into output primitives. (Note that sometimes the conversion is trivial: often straight line segments are both the graphic and the output primitives.) If the parametric curves and surfaces are used as graphic primitives, as discussed in the previous section, then the second transformation still has to be applied.

The first method used to display parametric surfaces is to subdivide them into polygons, generally quadrilateral or triangles, and then use the polygons obtained as graphic primitives, if necessary applying the required transformations, and displaying the polygons, using one of the many algorithms available for polygon display [Newman79] [Foley82].

The methods to compute the points on the surface have been discussed in section 3. It is here clearly advantageous to subdivide the surface equally in parametric space, and consequently to use table lookup or forward differencing.

If we use only one kind of regular polygon to subdivide the surface in parametric space, then only triangles and squares are geometrically possible (hexagons will leave some unfilled triangles at the 4 corners and boundaries). Subdivision into quadrilaterals offers the advantage of covering the surface with less line segments, and of being more pleasing to the eye (only lines approximating isoparametric curves are seen). This should be the preferred method with a line drawing system.

The drawback with parametric squares is that four points on the surface are not necessarily coplanar. This means that from some viewing position the quadrilateral in screen space might not be convex, and even not simple (i.e. two edges can cross). This would cause difficulty to most filling algorithms used for subsequent display since some deal only with convex polygons, and most cannot deal with non-simple polygons. In this case, a triangulation is preferable, as any 3 points are coplanar. Remains the choice of the subdivision factor. Since adaptive methods will be discussed in the next subsections, we will here only consider *a priori* determination of the subdivision factor. One solution is to let the user pick it (low for quick display, high for final version). If it is to be determined by the system, the two main considerations should be the size of the surface on screen and the curvature of the surface. It should be remembered, however, that equal distance in parametric space does not mean equal distance in world and/or screen space. If a roughly constant distance P_{dist} between sample points is wanted, then one can apply the viewing transformations to the four corners of the surface, and given the maximum distance d_{max} between these on the screen, compute n, the number of intervals on a side by:

$$n = \left\lceil \frac{d_{max}}{P_{dist}} \right\rceil$$

The curvature of the surface is a factor, since we are in effect doing a piecewise linear approximation of a polynomial surface.

One simple way to relate the subdivision factor to the curvature in the case of Bézier and B-spline formulation is through the control points. In the case of Bézier:

$$x^{(2)}(0) = 6[(V_0 - V_1) + (V_2 - V_1)]$$
$$s^{(2)}(1) = 6[(V_1 - V_2) + (V_3 - V_2)]$$

in the case of B-spline:

$$x^{(2)}(0) = (V_0 - V_1) + (V_2 - V_1)$$
$$x^{(2)}(1) = (V_1 - V_2) + (V_3 - V_2)$$

The number of subdivisions should be inversely related to the length of these vectors.

5.2. Scanline Algorithms

In recent years there has been a change in many applications from line drawing systems (refreshed or not) to raster systems. In terms of output primitives, this means going from points and lines to pixels and scanlines (this also means shading and colours - more about that later). As a consequence, algorithms were developed to convert graphic primitives to pixels. In particular algorithms to convert polygons to pixel scanline segment by scanline segments were designed [Newman 79, Foley 80]. These algorithms basically consist of 2 loops, one for the scanlines (the Y direction) and the other for each pixel on the scanline (the X direction). The problem is to determine the intersection of a plane defined by the eye and the scanline in screen coordinate system with the object to be "scan converted". At the same time the Z value of the intersection for each pixel is computed, in order to determine the priority of the objects incident on that pixel. To simplify computation, usually an incremental approach is used. In the case of linear objects like polygons, this means simply that after finding the first intersection (the polygon vertex with the highest Y), the endpoints are updated using the $\frac{\Delta X}{\Delta Y}$ slope for each edge. When a vertex is found signalling a change of intersecting edge, then the slope of the new edge is used. If new convex polygons are allowed, there can be more than one active span per polygon at a given scanline. Adapting this technique to the scanning of a non linear surface surface several problems occur: the highest point on the surface need not be a point on the boundary. The boundary edges are not the only ones that determine the scanline intersection, and the silhouette that define them is not a third degree curve. The silhouette is either a boundary or the locus of points where the Z-component (in the eye coordinate system) of the normal to the surface is 0. New spans are introduced by local maxima, and spans terminated at local minima.

Two techniques to solve these problems have been published by Blinn and by Whitted [Lane80b]. In Blinn algorithm, the major tool to compute the intersection is Newton iteration. This is of course an approximation method, but the coherence of the surface makes it easy to get accurate first guesses of the solutions and the iteration converges quickly. They are special cases however, such as saddle points, simultaneous maxima in the silhouette and the boundary, that have to be checked. There can also be singularities where the Newton iteration method fails.

In Whitted algorithm, the scanning proceeds like for the scanning of a polygon, except that the edges are described as cubic polynomials. Additional edges are needed if the curvature along one parameter is high. The problem of the silhouette edge is solved by approximating them by cubic polynomial, obtained by Hermite interpolation between two points on an edge where the Z-component of the normal vector is zero. The problem here is that numerous special cases make the generation of the silhouette edges fail. After all edges are created, the scan conversion processor first finds their extrema if they have any (by solving quadratic equation) and break each edge into segments monotonic in Y. Then the scanning is done using Newton iteration to get the value of the parameter at the intersection. This also might fail to converge.

Scanline algorithms on the whole are fairly ponderous, long and hard to code, and suffer many exceptions and failures.

5.3. Subdivision Algorithms

In section 3, we pointed out the two main approaches to the evaluation of points by increment and by subdivision. For display purposes the same dichotomy remains, but now the algorithms are driven by external considerations. In the previous methods the factors controlling the increment steps were screen resolution and surface curvature. In the subdivision methods the same factors will drive the subdivision.

The methods differ by the subdivision technique used, and by the criterion used to stop the subdivision. In Catmull's original method [Catmull74] central differencing is used, and the criterion is when the four corner points are inside a pixel. While the subdivision step is very fast, the method requires a substantial amount of storage (O(logN) for N subdivision on each side). The test value to end the subdivision is normally the size of the subsurface, but since the correcting factor $C(u,d)$ needed is the second derivative, it could conceivably be used to test flatness.

Instead of subdividing down to the pixel size, one could stop when the subsurface is "close enough" to a polygon to be displayed as such. In this case, we use polygons as output primitives instead of pixels, even though of course a subsequent procedure can scan convert these polygons to pixel. Carpenter and Lane [Lane80b] used this technique. Their subdivision method computes the new control points of each of the subsurface as shown in 3.6. The stopping criterion uses the convex hull property of the Bézier surfaces. The test for flatness is done on the control vertices, instead of on the surface itself. The distance from inside points of the control network to the plane defined by the four corner points can be used as a measure. The main drawback of the method is that cracks appear on the surface, if a subsurface is subdivided, while another subsurface with a common edge is not.

A third variant proposed by Clark [Clark79] uses the same subdivision as Catmull does, central differencing, but uses the correction term

$$C(u,d)=x^{(2)}(u)\frac{d^2}{2}$$

as the stopping condition. In the Bézier curve

$$x^{(2)}(0)=6[(P_0-P_1)+(P_2-P_1)]$$
$$x^{(2)}(1)=6[(P_3-P_2)+(P_1-P_2)]$$

so that

$$C(0,1)=3[(P_0-P_1)+(P_2-P_1)]$$

So the smaller $C(u,d)$ is the closer to a straight line $(V_0V_1V_2V_3)$ is. Note that this should be considered in relation with the resolution of the screen and the length of $(V_0V_1V_2V_3)$. Clark's termination tests are done first on the boundary curves along the $u=1$ and $u=0$ curves, which one subdivided until they meet the "flatness" criterion, then on the $v=0$, $v=1$ curves treated the same. Then the test is carried out to the middle of the surface, by using the cross derivative $\frac{\partial^4 x(u,v)}{\partial u^2 \partial v^2}$ at the four corners as the test value. By subdividing along the boundary curves first, and then not subtracting a correction term if any further subdivision is needed (because of the centre of the surface) Clark's algorithm avoids the "cracks" along the boundary curves.

The Carpenter-Lane algorithm can be extended to arbitrary B-spline formulations by using the Oslo algorithm for the subdivision. It should be also noted that subdivision can be used to clip (especially when the formulation has the convex hull property, because then if the clipping plane does not intersect the control vertices, it does not intersect the curve or surface) and to compute intersections of two surfaces (as a step in a hidden surface algorithm, for instance).

6. Conclusion

This survey of parametric curves and surfaces and their associated computational techniques is far from complete. Some of the topics, important for implementation, but left out here are:

-transformation form one type of formulation to another [Barsky81c]
-computations related to lighting models, in particular normal vector computations
-secondary lighting effects, like shadows and refraction [Williams78, Whitted80]
-mapping of scalar values onto the parametric surface, for texture mapping [Blinn76, Blinn78] or stochastic modeling [Fournier82].
-relationship with other surface representations, in particular quadric and super quadrics [Barr81]
-techniques, mainly interactive, to effectively design objects with parametric curves or surfaces for the purpose of Computer Graphics applications, with a repertoire of control vertices for standard shapes.

A subsequent paper will address these issues. It is our hope that this partial survey will give the reader a renewed interest in parametric curves and surfaces, and convince him that they are powerful, yet relatively easy to use as building blocks for our imaginary universe.

References

Barnhill74a. Barnhill, Robert E. and Riesenfeld, Richard F.(editors). *Computer Aided Geometric Design*, Academic Press, New York, 1974.

Barr81. Barr, Alan H. "Superquadrics and Angle-Preserving Transformations", IEEE Computer Graphics and Applications, Vol. 1, No 1, January 1981, pp. 11-23.

Barsky81a. Barsky, Brian A. "Computer Aided Geometric Design: A Bibliography with Keywords and Classified Index", IEEE Computer Graphics and Applications, Vol. 1, No. 3, July 1981, pp. 67-109. Also to be reprinted in ACM Computer Graphics.

Barsky81b. Barsky, Brian A. *The Beta*-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures, Ph.D. Thesis, University of Utah, Salt Lake City, Utah, December 1981.

Barsky81c. Barsky, Brian A. and Thomas, Spencer W. "TRANSPLINE -- A System for Representing Curves Using Transformations among Four Spline Formulations", The Computer Journal, Vol. 24, August 1981, pp. 271-277.

Barsky82. Barsky, Brian A. "A Study of the Parametric Uniform B-spline Curve and Surface Representations", in preparation.

Bézier74. Bézier, Pierre E. "Mathematical and Practical Possibilities of UNISURF", In *Computer Aided Geometric Design*, edited by Barnhill, Robert E. and Riesenfeld, Richard F., Academic Press, New York, 1974, pp. 95-126.

Blinn76. Blinn, James F. "Texture and Reflections in Computer Generated Images", Communications of the ACM, Vol. 19, No 10, October 1976, pp.542-547.

Blinn78. Blinn, James F. "Simulation of Wrinkled Surfaces", SIGGRAPH '78 Proceedings, published as Computer Graphics, Vol. 12, No 3, August 1978, pp.286-292.

de Boor78. de Boor, Carl. *A Practical Guide to Splines*, Springer-Verlag, Applied Mathematical Sciences, Vol. 27, 1978.

Carpenter80. Carpenter, Loren C. *Vol Libre*, computer animated film, first showing at SIGGRAPH '80.

Catmull74. Catmull, Edwin E. *Computer Display of Curved Surfaces*, Ph.D. thesis, University of Utah, Salt Lake City, Utah, 1974.

Clark79. Clark, James H. "A Fast Algorithm for Rendering Parametric Surfaces", SIGGRAPH '79 Proceedings (Abstract only), to appear in Communications of the ACM.

Cohen80. Cohen, Elaine; Lyche, Tom; and Riesenfeld, Richard F. "Discrete B-splines and Subdivision Techniques in Computer Aided-Geometric Design and Computer Graphics", Computer Graphics and Image Processing, Vol. 14, No. 2, October 1980, pp. 87-111.

Coons67. Coons, Steven A. "Surfaces for Computer-Aided Design of Space Forms", Tech. Report No. MAC-TR-41, Project MAC, M.I.T., Cambridge, Massachusetts, June 1967.

Coons74. Coons, Steven A. "Surface Patches and B-spline Curves". In Computer Aided Geometric Design, edited by Barnhill, Robert E. and Riesenfeld, Richard F., Academic Press, New York, 1974, pp. 1-16.

Curry47. Curry, H. B. and Schoenberg, I. J. "On Spline Distributions and their Limits: the Polya Distribution Functions, Abstract 380t", Bulletin of the American Math. Society, Vol. 53, 1947, p. 1114.

Curry66. Curry, H. B. and Schoenberg, I. J. "On Polya Frequency Functions IV: The Fundamental Spline Functions and their Limits", Journal d'Analyse Mathematique, Vol. 17, 1966, pp. 71-107.

England78. England, J. N. "A System for Interactive Modeling of Physical Curved Surface Objects", SIGGRAPH '78 Proceedings, published as Computer Graphics, Vol. 12, No 3, August 1978, pp. 336-340.

Faux79. Faux, Ivor D. and Pratt, Michael J. *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., 1979.

Foley82. Foley, James D. and Van Dam, Andries *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982.

Forrest72. Forrest, A. Robin. "On Coons and Other Methods for the Representation of Curved Surfaces", Computer Graphics and Image Processing, Vol. 1, No. 4, December 1972, pp. 341-359.

Fournier82. Fournier, Alain, Fussell, Donald S. and Carpenter, Loren "Computer Rendering of Stochastic Models", to appear in Communications of the ACM, June 1982.

Gordon74a. Gordon, William and Riesenfeld, Richard F. "B-spline Curves and Surfaces", In Computer Aided Geometric Design, edited by Barnhill, Robert E. and Riesenfeld, Richard F., Academic Press, New York, 1974, pp. 95-126.

Gordon74b. Gordon, William J. and Riesenfeld, Richard F. "Bernstein-Bézier Methods for the Computer Aided Design of Free-Form Curves and Surfaces", Journal of the ACM, Vol. 21, No. 2, April 1974, pp.293-310.

Knuth69. Knuth, Donald E. *The Art of Computer Programming*, Volume 2, "Seminumerical Algorithms", Addison-Wesley, 1969.

Lane80a. Lane,Jeffrey M. and Riesenfeld, Richard, F. "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, January 1980, pp 35-46.

Lane80b. Lane, Jeffrey M.; Carpenter, Loren C.; Whitted, J. Turner; and Blinn, James F. "Scan Line Methods for Displaying Parametrically Defined Surfaces", Communications of the ACM, Vol. 23, No. 1, January 1980, pp. 23-34.

Newman79. Newman, William M. and Sproull, Robert F. *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979, second edition.

Ralston65. Ralston, A. *A First Course in Numerical Analysis*, McGraw-Hill, 1965.

Riesenfeld73. Riesenfeld, Richard F. *Applications of B-spline Approximation to Geometric Problems of Computer-Aided Design*, Ph.D. Thesis, Syracuse University, Syracuse, N.Y., May 1973. Also Tech. Report No. UTEC-CSc-73-126, Department of Computer Science, University of Utah.

Riesenfeld81a. Riesenfeld, Richard F. "Homogeneous Coordinates and Projective Planes in Computer Graphics", IEEE Computer Graphics and Applications, Vol. 1, No 1, January 1981, pp. 50-55.

Riesenfeld81b. Riesenfeld, Richard F.; Cohen, Elaine; Fish, Russell D.; Thomas, Spencer W; Cobb, Elizabeth S.; Barsky, Brian A.; Schweitzer, Dino L.; and Lane, Jeffrey M. "Using the Oslo Algorithm as a Basis for CAD/CAM Geometric Modelling", In Proceedings of the Second Annual NCGA National Conference, National Computer Graphics Association, Inc., Baltimore, 14-18 June 1981, to appear.

Schoenberg46. Schoenberg, Isaac J. "Contributions to the Problem of Approximating Equidistant Data by Analytic Functions", Quarterly Applied Math., Vol. 4, No. 1, 1946, pp. 45-99 and 112-141.

Whitted80. Whitted, J. Turner, "An Improved Illumination Model for Shaded Display", Communications of the ACM, Vol. 23, No 6, June 1980, pp. 343-349.

Williams78. Williams, Lance "Casting Curved Shadows on Curved Surfaces", SIGGRAPH '78 Proceedings, published as Computer Graphics, Vol. 12, No 3, August 1978, pp. 270-274.

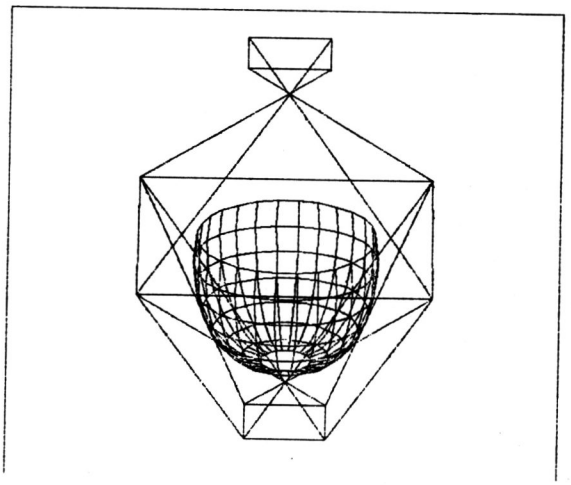

Figure 5.3 Object made of 4 B-spline surfaces with their control vertices.

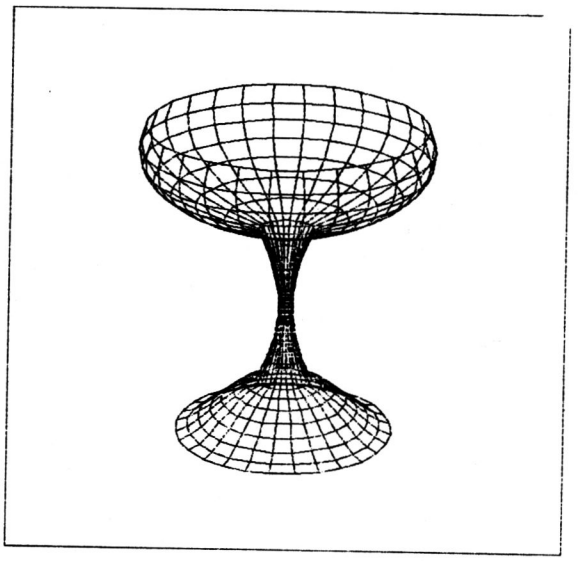

Figure 5.1 Glass made of 12 Bézier surfaces.

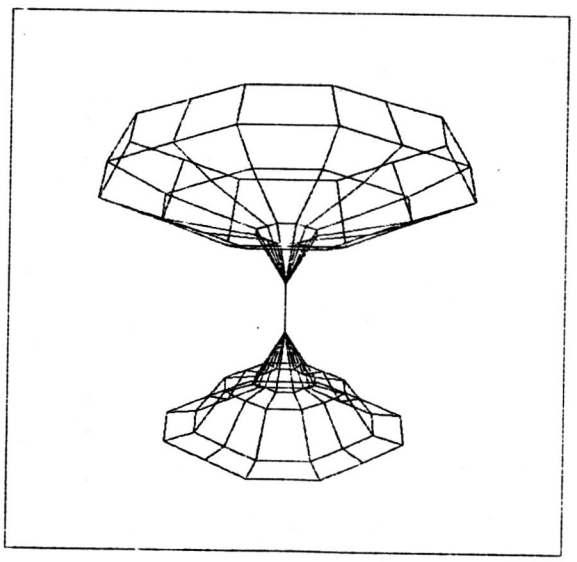

Figure 5.2 Control vertices for the glass.