The Beta2-spline: A Special Case of the Beta-spline Curve and Surface Representation

Brian A. Barsky

Tony D. DeRose

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

Tech. Report No. UCB/CSD 83/152
November 1983

ABSTRACT

This paper develops a special case of the Beta-spline curve and surface technique called the Beta2-spline. While a general Beta-spline has two parameters (β_1 and β_2) controlling its shape, the special case presented here has only the single parameter β_2. Experience has shown this to be a simple, but very useful special case that is computationally more efficient than the general case. Optimized algorithms for the evaluation of the Beta2-spline basis functions and subdivision of Beta2-spline curves and surfaces are presented.

This work was supported in part by the National Science Foundation under grant number ECS-8204381 and the Defense Advanced Research Projects Agency under contract number N00039-82-C-0235.

1. Introduction

A *spline* curve or surface is a piecewise function with continuity constraints at the locations where the pieces of the function meet (often called the *joints*, in the case of curves, and *borders*, in the case of surfaces). Let $\mathbf{Q}_i(u)$ denote the i^{th} segment of such a piecewise representation*. Since $\mathbf{Q}_i(u)$ is a vector-valued function, it can be expressed as a tuple of scalar-valued functions, one for each spatial dimension. A two-dimensional curve segment can thus be expressed as

$$\mathbf{Q}_i(u) = (X_i(u), Y_i(u)) \tag{I.1}$$

We adhere to the usual convention of restricting the domain parameter u to the range [0,1]. Thus, $\mathbf{Q}_i(0)$ is the starting point of the i^{th} segment and $\mathbf{Q}_i(1)$ is the ending point (see figure 1).

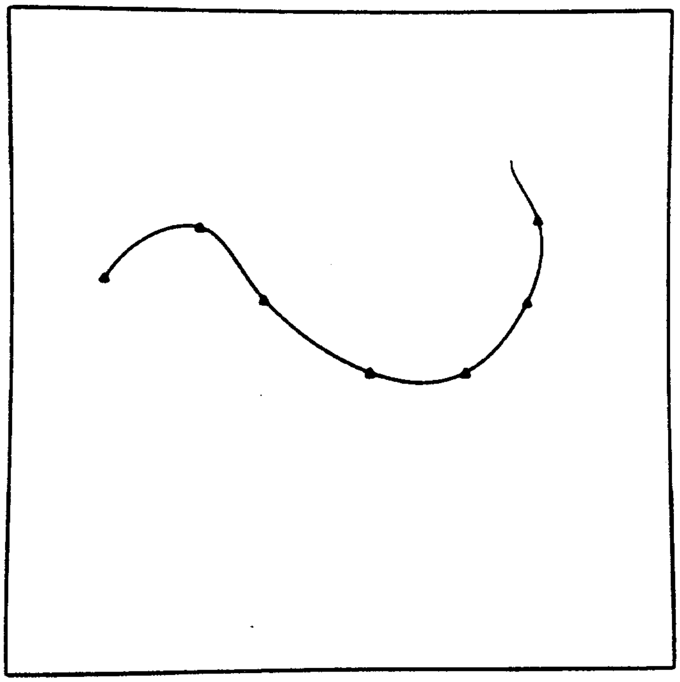

Figure 1.
A section of a vector-valued spline curve. Triangles show the locations of the joints.

A special case of spline functions called a *blended polynomial spline* has become popular in recent years because of its elegance and computational efficiency. B-splines and Bézier curves and surfaces are examples of this type of spline. A segment of a blended spline curve takes the form

$$\mathbf{Q}_i(u) = \sum_{r=0}^{k} \mathbf{V}_{i+r} B_r(u) \tag{I.2}$$

The functions $B_0(u), B_1(u), \cdots, B_k(u)$ are called the *blending* or *basis* functions. The sequence of vertices $<\mathbf{V}_i, \mathbf{V}_{i+1}, \cdots, \mathbf{V}_{i+k}>$ forms the *control polygon* of the curve segment. An example of a B-spline curve and its corresponding control polygon is shown in figure 2.

*Vectors and vector-valued functions are denoted by boldface type, as in $\mathbf{V}$ and $\mathbf{Q}(u)$.

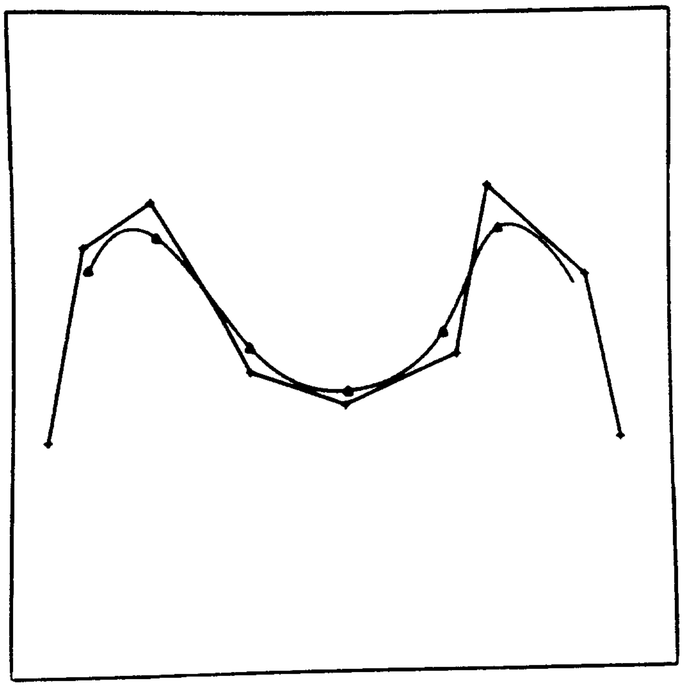

Figure 2.
A control polygon and its B-spline curve. The control vertices are highlighted by plus signs; the joints are denoted by triangles.

Surfaces can also be constructed using a blended technique to form what is known as a *tensor product surface.* A single tensor product surface segment, more often called a *surface patch,* is described by two variable parameters u and v. In particular, the i,jth such patch is defined as

$$\mathbf{S}_{i,j}(u,v) = \sum_{r=0}^{k} \sum_{s=0}^{l} \mathbf{V}_{i+r,j+s} B_r(u) B_s(v) \tag{I.3}$$

The array of vertices $\mathbf{V}_{i+r,j+s}$ is called the *control hull* or *control graph* of the surface patch. These surface patches are pieced together to form a mosaic making up the spline surface. Once again, it is usual to restrict the domain parameters to the range [0,1]. $\mathbf{S}_{i,j}(u,0)$ denotes a curve, the curve bounding the surface at the $v=0$ contour. The other three boundary curves are $\mathbf{S}_{i,j}(u,1)$, $\mathbf{S}_{i,j}(0,v)$, and $\mathbf{S}_{i,j}(1,v)$ (see figure 3).

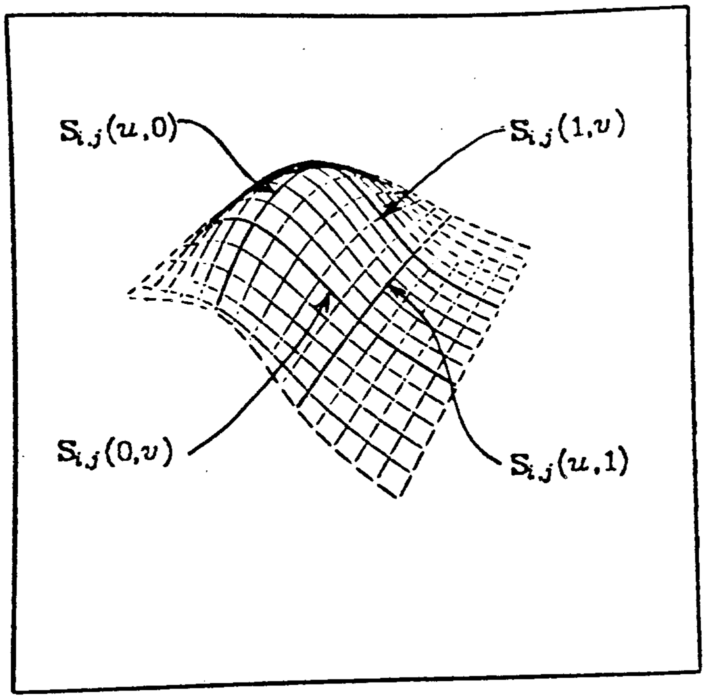

Figure 3.
A spline surface in the region of the patch $\mathbf{S}_{i,j}(u,v)$.

1.1. Continuity

To discuss the continuity of a spline curve at the joints we introduce the concepts of the first and second derivative vectors. Differentiation of a vector-valued function is performed by standard scalar differentiation on each component of the vector. Thus, for a two-dimensional vector function $\mathbf{Q}_i(u)$, the first derivative vector written in component form is

$$\frac{d\mathbf{Q}_i(u)}{du} = \left(\frac{dX_i(u)}{du}, \frac{dY_i(u)}{du}\right) \tag{I.4}$$

Similarly, the second derivative vector written in component form is

$$\frac{d^2\mathbf{Q}_i(u)}{du^2} = \left(\frac{d^2X_i(u)}{du^2}, \frac{d^2Y_i(u)}{du^2}\right) \tag{I.5}$$

For convenience, we denote the first derivative vector as $\mathbf{Q}_i^{(1)}(u)$ and second derivative vector as $\mathbf{Q}_i^{(2)}(u)$.

For *positional* continuity of the spline curve, the ending point of the ith segment must match the starting point of the $i+1$st segment. This requirement is not sufficient to make the joint appear to be *smooth* however. Until recently it has been assumed that for a spline to appear smooth the first and second derivative vectors must be continuous at the joints. However, first and second derivative continuity is too strict; it is sufficient to require that the *unit tangent vector* and *curvature vector* remain continuous at a joint.[1,3] It is not immediately obvious that continuity of the unit tangent vector and curvature vector is less restrictive than continuity of first and second derivative vectors, but this can be shown by considering the definitions of these quantities.[3,1]

2. Motivation for a Beta2-spline

In the general case of the Beta-spline, the unit tangent vector and curvature vectors are continuous across the joints in the spline. In the special case presented here, the requirement of continuity of the unit tangent is replaced with that of the first derivative vector; this results in a spline possessing continuity of the first derivative vector and curvature vector. Requiring first derivative vector continuity removes the freedom to vary the shape parameter called β_1. However, since curvature vector continuity is used instead of second derivative vector, the second shape parameter, called β_2, may still be varied. The choice of this special case has been made based on both practical and theoretical results.

From a practical standpoint, first derivative and curvature vector continuity produces a spline technique that is simple to understand and useful from the point of view of the designer. Experience with the general Beta-spline has shown that users of the technique find β_2 a more intuitive parameter than β_1. As section 4 will show, β_2 behaves in what can be called a symmetric, or uniform manner, while β_1 behaves in an asymmetric, or non-uniform way.[4] It is the symmetry of β_2 that makes it the more appealing shape parameter.

On the theoretical side, the derivation of the equations governing the evaluation and behavior of a Beta2-spline is easier. Moreover, the form of the resulting equations is simpler, leading of course to algorithms that are computationally more efficient. The simplicity of the resulting equations stems from the fact that β_1 enters the solution of the general Beta-spline in a much more complex way than β_2. Thus, requiring β_1 to take on its default value of 1 allows extensive algebraic simplification.

The next section gives the derivation of a Beta2-spline. Section 4 then examines the qualitative behavior of β_2. The remaining sections give optimized algorithms for the evaluation and subdivision of Beta2-spline curves and surfaces.

3. Derivation of the Beta2-spline Basis Functions

The i^{th} segment of a Beta2-spline curve takes the form

$$\mathbf{Q}_i(\beta_2;u) = \sum_{r=-2}^{1} \mathbf{V}_{i+r}\, b_r(\beta_2;u)\ ;\quad 0 \le u \le 1 \tag{III.1}$$

We seek the four functions $b_{-2}(\beta_2;u), b_{-1}(\beta_2;u), b_0(\beta_2;u), b_1(\beta_2;u)$. If we assume a cubic polynomial form for each of the basis functions, then each can be expressed as

$$b_r(\beta_2;u) = \sum_{g=0}^{3} c_{rg}(\beta_2)u^g \tag{III.2}$$

Thus, discovery of the sixteen unknowns $c_{rg}(\beta_2)$, $-2 \le r \le 1$, $0 \le g \le 3$ is sufficient to completely describe the Beta2-spline basis functions. These coefficients must be constructed such that the resulting curve has positional, first derivative vector, and curvature vector continuity. Positional continuity between the segments $\mathbf{Q}_i(\beta_2;u)$ and $\mathbf{Q}_{i+1}(\beta_2;u)$ implies

$$\mathbf{Q}_{i+1}(\beta_2;0) = \mathbf{Q}_i(\beta_2;1) \tag{III.3}$$

For continuity of first derivative vector, we require that

$$\mathbf{Q}_{i+1}^{(1)}(\beta_2;0) = \mathbf{Q}_i^{(1)}(\beta_2;1) \tag{III.4}$$

It can also be shown [3,1] that if the first derivative vector is to be continuous then the curvature vector is continuous at this joint if and only if

$$\mathbf{Q}_{i+1}^{(2)}(\beta_2;0) = \mathbf{Q}_i^{(2)}(\beta_2;1) + \beta_2\mathbf{Q}_i^{(1)}(\beta_2;1) \tag{III.5}$$

If $\beta_2 = 0$, the constraint equation (III.5) above reduces to the requirement for continuity of the second derivative vector.

Substitution of the blended form of $\mathbf{Q}_i(u)$ into the position constraint equation (III.3) leads to

$$\begin{aligned} \mathbf{V}_{i-1}b_{-2}(\beta_2;0) + \mathbf{V}_i\, b_{-1}(\beta_2;0) + \mathbf{V}_{i+1}b_0(\beta_2;0) + \mathbf{V}_{i+2}b_1(\beta_2;0) &= \\ \mathbf{V}_{i-2}b_{-2}(\beta_2;1) + \mathbf{V}_{i-1}b_{-1}(\beta_2;1) + \mathbf{V}_i\, b_0(\beta_2;1) + \mathbf{V}_{i+1}b_1(\beta_2;1) \end{aligned} \tag{III.6}$$

For this relation to hold for arbitrary vertices the following five constraints must be simultaneously satisfied.

$$\begin{aligned} 0 &= b_{-2}(\beta_2;1) \\ b_{-2}(\beta_2;0) &= b_{-1}(\beta_2;1) \\ b_{-1}(\beta_2;0) &= b_0(\beta_2;1) \\ b_0(\beta_2;0) &= b_1(\beta_2;1) \\ b_1(\beta_2;0) &= 0 \end{aligned} \tag{III.7}$$

In a similar manner, the continuity of first derivative vector can be rewritten as the five constraint equations

$$\begin{aligned} 0 &= b_{-2}^{(1)}(\beta_2;1) \\ b_{-2}^{(1)}(\beta_2;0) &= b_{-1}^{(1)}(\beta_2;1) \\ b_{-1}^{(1)}(\beta_2;0) &= b_{0}^{(1)}(\beta_2;1) \\ b_{0}^{(1)}(\beta_2;0) &= b_{1}^{(1)}(\beta_2;1) \\ b_{1}^{(1)}(\beta_2;0) &= 0 \end{aligned} \qquad \text{(III.8)}$$

Finally, the continuity of the curvature vector results in the five equations

$$\begin{aligned} 0 &= b_{-2}^{(2)}(\beta_2;1) \\ b_{-2}^{(2)}(\beta_2;0) &= b_{-1}^{(2)}(\beta_2;1)+ \beta_2 b_{-1}^{(1)}(\beta_2;1) \\ b_{-1}^{(2)}(\beta_2;0) &= b_{0}^{(2)}(\beta_2;1)+ \beta_2 b_{1}^{(1)}(\beta_2;1) \\ b_{0}^{(2)}(\beta_2;0) &= b_{1}^{(2)}(\beta_2;1)+ \beta_2 b_{1}^{(1)}(\beta_2;1) \\ b_{1}^{(2)}(\beta_2;0) &= 0 \end{aligned} \qquad \text{(III.9)}$$

Substituting the polynomial form chosen for the $b_i(\beta_2;u)'s$ leads to a set of 15 constraints in the 16 unknown coefficients. This amounts to an under-determined system of equations which can be made complete by introducing another, linearly independent, constraint. The constraint we choose is a normalization to guarantee that the curve segment is within the convex hull of the four vertices that define it. To endow the Beta2-spline technique with the convex hull property, the basis functions must sum to unity over the interval [0,1]; i.e.,

$$b_{-2}(\beta_2;u)+ b_{-1}(\beta_2;u)+ b_0(\beta_2;u)+ b_1(\beta_2;u) = 1$$

This is actually four constraints, one for each power of u. Three of the equations can be formed via linear combinations of the previous 15 equations; the remaining constraint is

$$c_{-2,0}(\beta_2)+ c_{-1,0}(\beta_2)+ c_{0,0}(\beta_2)+ c_{1,0}(\beta_2) = 1$$

While it is possible to solve the above system of equations numerically for each value chosen for β_2, it is also possible to solve the system algebraically using an algebraic manipulation system such as Vaxima.[11] The resulting solution can be written as a matrix $\bar{\mathbf{C}}$ of the coefficients of the basis functions**,

$$\bar{\mathbf{C}}(\beta_2) = \begin{bmatrix} c_{-2,0} & c_{-1,0} & c_{0,0} & c_{1,0} \\ c_{-2,1} & c_{-1,1} & c_{0,1} & c_{1,1} \\ c_{-2,2} & c_{-1,2} & c_{0,2} & c_{1,2} \\ c_{-2,3} & c_{-1,3} & c_{0,3} & c_{1,3} \end{bmatrix} = \gamma \begin{bmatrix} 2 & \beta_2+ 8 & 2 & 0 \\ -6 & 0 & 6 & 0 \\ 6 & -3(\beta_2+ 4) & 3(\beta_2+ 2) & 0 \\ -2 & 2(\beta_2+ 3) & -2(\beta_2+ 3) & 2 \end{bmatrix} \qquad \text{(III.10)}$$

or as the basis functions themselves

$$\begin{aligned} b_{-2}(\beta_2;u) &= 2\gamma(1-u)^3 \\ b_{-1}(\beta_2;u) &= \gamma(\beta_2+ 8+ u^2(-3(\beta_2+ 4)+ 2u(\beta_2+ 3))) \\ b_0(\beta_2;u) &= \gamma(2+ u(6+ 3u(\beta_2+ 2)-2u(\beta_2+ 3))) \\ b_1(\beta_2;u) &= 2\gamma u^3 \end{aligned} \qquad \text{(III.11)}$$

where

$$\gamma = \frac{1}{\beta_2+ 12} \qquad \text{(III.12)}$$

4. Behavior of β_2

Examination of the curvature vector constraint equation (III.5) shows that as β_2 becomes large the second derivative (and hence the radius of curvature) of the succeeding segment also becomes large. In the limit of infinite β_2, the radius of curvature of the adjoining segment becomes infinite. In other words the adjoining segment approaches linearity. In this limit, Barsky and Beatty[4] show that the Beta-spline curve becomes a piecewise linear function that interpolates the vertices of the control polygon. This behavior of β_2 suggests that it acts like a tension

**A matrix is denoted by a boldface character with a diacritical bar parameter and thus β_2 is called *tension*. Increasing the tension of a Beta2-spline amounts to increasing the value of β_2 (see figure 4).

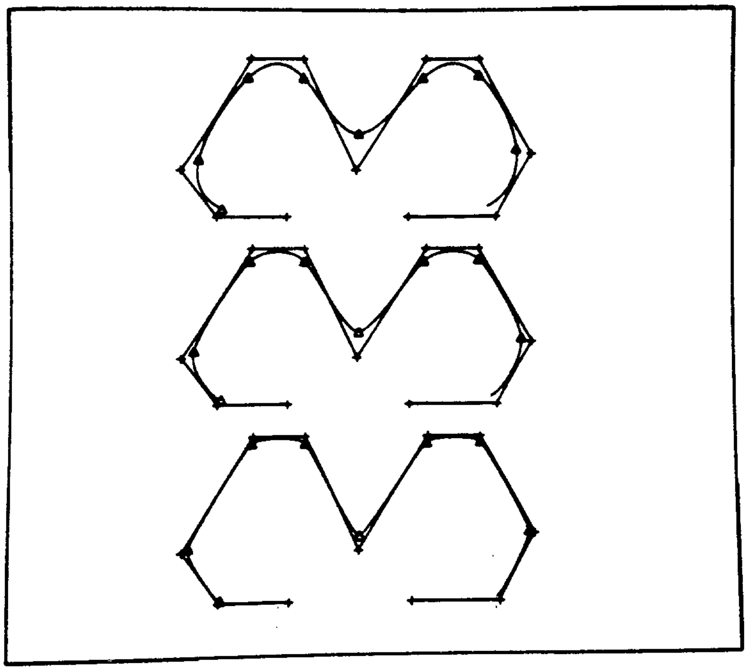

Figure 4.
The curves above differ only in the value of β_2. The value of β_2 is 0 for the top curve, 5 for the middle curve, and 20 for the bottom curve. Plus signs denote control vertices, triangles denote joints. Note that as β_2 increases toward infinity, the curve uniformly approaches the control polygon.

Increasing the tension applied to a Beta2-spline surface has the effect of making the surface more polygonal. In the limit of infinite tension, the surface can be shown to be coincident with the control graph that defines it. This behavior is shown in the figure 5.

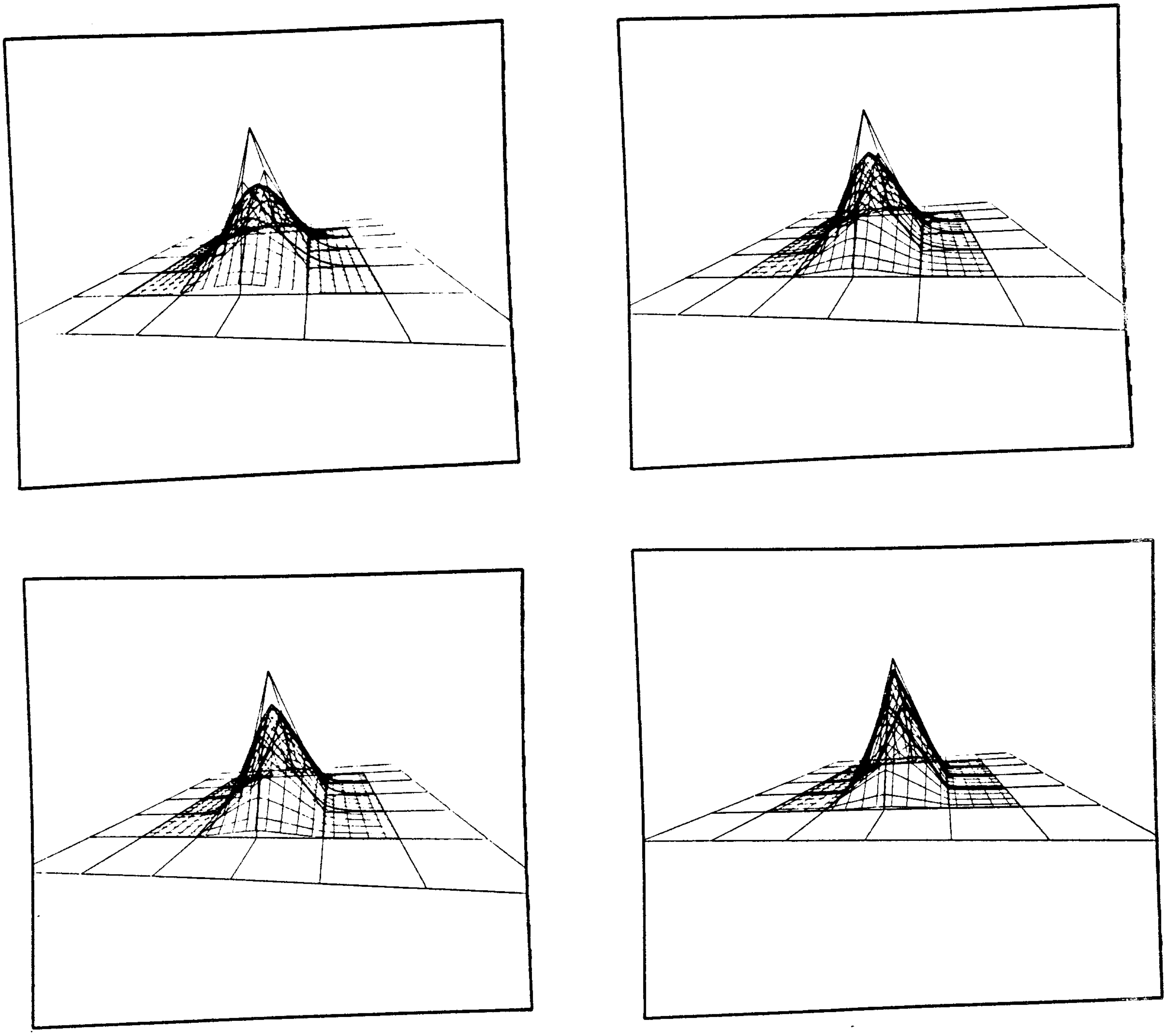

Figure 5.
This sequence of Beta2-spline surfaces and control graphs shows the effect of increasing the value of β_2. The values of β_2 are 0 *for the top left surface,* 5 *for the top right surface,* 10 *for the bottom left surface, and* 50 *for the surface in the bottom right position. As β_2 increases, the surface uniformly approaches the control graph. In the limit of infinite β_2, the surface actually coincides with the control graph.*

Barsky and Beatty [4] also show that a curve segment or surface patch is guaranteed to be within the convex hull of the vertices that define it for non-negative values of β_2.

When $\beta_2=0$,. the spline has continuity of first and second derivative vectors; it is thus equivalent to a cubic uniform B-spline when $\beta_2=0$. In other words, a Beta2-spline (or a Beta-spline) is a generalized form of a cubic uniform B-spline.

Recent work[4] has investigated methods of changing the shape parameters in a localized portion of the curve or surface. However, we will not deal with these issues in this paper.

5. Evaluation of the Basis Functions

If the curve is to be drawn by repeated evaluation of (III.1), say at the domain values $(u_0,u_1, \cdots ,u_p)$, the basis functions must be evaluated at these points. Fortunately, it is sufficient to evaluate the coefficients of the basis functions once as an initializing step, then use them in a Horner's rule computation at the $p+1$ values of the domain parameter. The following algorithm can be used to tabulate the values of the basis functions given β_2:

```
compute_c( $\beta_2$, c);

for all $u$ in $\{u_0,u_1, \cdots u_p\}$ do
    compute_b( c, u, b);
```

where

```
procedure compute_c( $\beta_2$, c)
local variable $\gamma$;
begin

    $\gamma := 1/(\beta_2+ 12)$;
    $c_{-2,0} := 2\gamma$;
    $c_{-1,0} := \gamma(\beta_2+ 8)$;
    $c_{-1,2} := -3\gamma(\beta_2+ 4)$;
    $c_{-1,3} := c_{-2,0}(\beta_2+ 3)$;
    $c_{0,0} := c_{-2,0}$;
    $c_{0,1} := 3c_{0,0}$;
    $c_{0,2} := 3\gamma(\beta_2+ 2)$;
    $c_{0,3} := -c_{-1,3}$;
    $c_{1,3} := c_{-2,0}$;
end;

procedure compute_b( c, u, b)
local variable one_minus_u;
begin
    $one\_minus\_u := (1-u)$;
    $b_{-2} := c_{-2,0}\ one\_minus\_u^3$;
    $b_{-1} := c_{-1,0}+ u^2(c_{-1,2}+ u\ c_{-1,3})$;
    $b_0 := c_{0,0}+ u(c_{0,1}+ u(c_{0,2}+ u\ c_{0,3}))$;
    $b_1 := c_{1,3}\ u^3$;
end;
```

The above algorithm requires $5+ 6(p+ 1)$ additions / subtractions, $8+ 11(p+ 1)$ multiplications, and 1 division. When p is small, this cost is roughly 50% of the cost of the algorithms given by Barsky[3, 1] for general Beta-spline evaluation.

6. Subdivision

6.1. General Subdivision Schema for Curves

The previous section described an algorithm for the evaluation of a Beta2-spline curve based on the blended definition given in (III.1). This method of curve evaluation does have its problems when used to approximate the curve, however. If the evaluation algorithm is being used to obtain a piecewise linear approximation to the true curve, the step size for the domain parameter is hard to estimate to achieve a given spatial accuracy. Moreover, if the curve segment is highly curved in one region and relatively flat in another, the approximation will not be uniformly good.

The *subdivision*, or splitting of a curve or surface has received considerable attention in recent years.[7,8,10,12,13,14] For instance, Catmull[6] introduced an algorithm that subdivides a surface that is to be approximated. However, Catmull's technique subdivided the surface until the pieces were of size on the order of a pixel. Lane and Carpenter[12] have improved the technique for splines that have the convex hull property. Very simply, their approach is as follows:

The sequence of vertices describing the segment in question is split into two sequences each containing the same number of vertices as the original. These new sequences describe the "left" portion and "right" portion of the original curve, respectively. We call the sequence describing the "left" part the "left sub-polygon", and the sequence describing the "right" part the "right sub-polygon" (see figure 6). It is common to split the curve at the *parametric midpoint* (the point where the domain parameter $u=1/2$), but it is also possible to perform non-midpoint subdivision.[2,9] Midpoint subdivision is sufficient for our purposes. However, subdivision at the parametric midpoint does not, in general, yield two pieces of equal arclength.

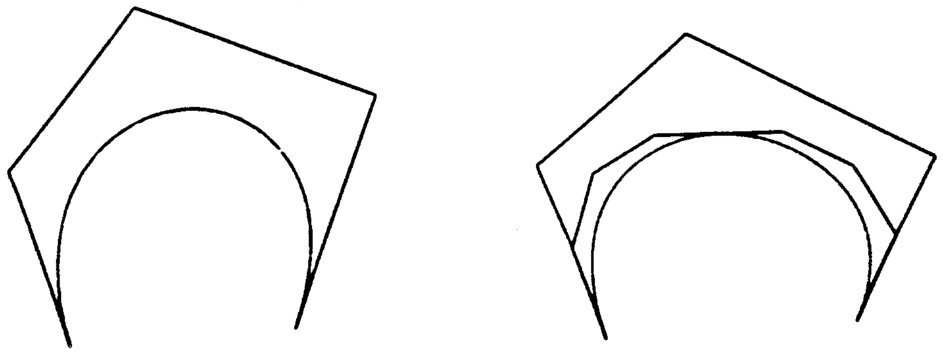

Figure 6.
The figure on the left shows a curve and its control polygon. The figure on the right depicts the curve, its control polygon, and the left and right sub-polygons corresponding to a midpoint subdivision.

Since the original curve lies within the convex hull of its control polygon, the sub-polygons will be flatter than the original. Recursive application of this subdivision procedure must result in successively flatter sub-polygons. The convex hull property also guarantees that a relatively flat control polygon must define a relatively flat curve. Thus, the recursion stops when the sub-polygon is deemed to be flat to within some tolerance. At this point, the curve generated by the polygon can be approximated by a single linear segment. The union of these linear approximants yields a uniformly accurate representation of the original curve segment. If $\overline{\mathbf{V}}$ denotes the original control polygon, $\overline{\mathbf{V}}^L$ represents the left sub-polygon, and $\overline{\mathbf{V}}^R$ the right sub-polygon, then the schema can be stated more precisely by the following recursive procedure:

```
procedure Approximate_Curve( $\overline{\mathbf{V}}$, $\epsilon$)
local variable $\overline{\mathbf{V}}^L$, $\overline{\mathbf{V}}^R$;
begin
   if $\overline{\mathbf{V}}$ is flat to within $\epsilon$ then
      Linearly_Approximate( $\overline{\mathbf{V}}$);
   else
      Subdivide( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^L$, $\overline{\mathbf{V}}^R$);
      Approximate_Curve( $\overline{\mathbf{V}}^L$, $\epsilon$);
      Approximate_Curve( $\overline{\mathbf{V}}^R$, $\epsilon$);
   endif;
end;
```

The procedure *Subdivide* splits its first argument into left and right sub-polygons, which are returned in the second and third arguments, respectively. The routine *Linearly_Approximate* generates a single line segment to approximate the curve defined by the control polygon given as its argument. The determination of the flatness of a control polygon can be done by testing the perpendicular distance of interior vertices from the line segment connecting the starting and ending vertices (see figure 7). If the largest such distance is less than the flatness criteria ϵ, then the control polygon passes the flatness test and is approximated by a linear segment.

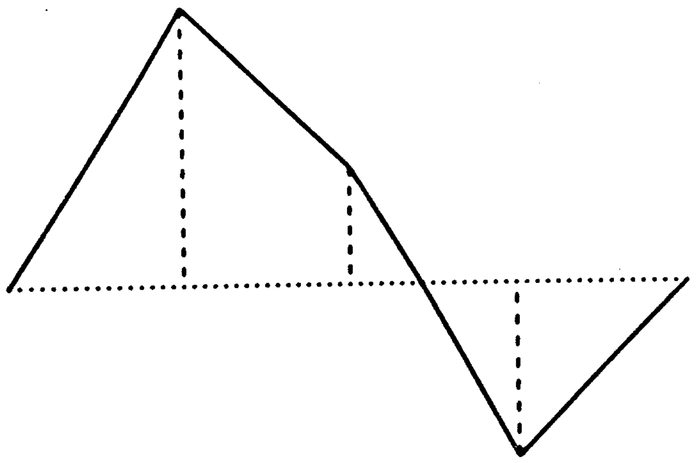

Figure 7.
Determination of a flat control polygon. It is deemed flat if the maximum perpendicular distance (dashed line) between each interior vertex and the line segment connecting the endpoints of the control polygon (dotted line) is less than the tolerance ϵ.

If the spline technique being used the endpoints of the control polygon, then the curve segment generated by a control polygon within the flatness criterion can be approximated by the line connecting the endpoints. Bézier curves are an example of such a technique.

6.2. General Schema for Surfaces

The recursive subdivision algorithm for surfaces proceeds in much the same way as for curves. The primary difference is that each time the surface is to be split there are several ways the subdivision can be done. Should the surface be split along the u parametric direction, or the v parametric direction, or both? One way to answer this question is based on the shape of the control graph describing the surface. The surface should be subdivided along the parametric direction, or directions, in which the surface is highly curved. Although it is in general hard to determine if a surface is flat, the convex hull property guarantees that a relatively flat graph must produce a relatively flat surface. Although it is not generally done in practice, a surface can be split in both directions by splitting along one direction, say the u direction, to obtain two sub-graphs, followed by splitting the sub-graphs along v. When a graph is deemed flat to within the given tolerance, the surface it defines can be approximated by one or more polygonal approximants.

Let $\overline{\mathbf{V}}$ represent the control graph for the surface to be approximated, $\overline{\mathbf{V}}^{LU}$ and $\overline{\mathbf{V}}^{RU}$ be the left and right sub-graphs resulting from a split in the u direction, respectively, and $\overline{\mathbf{V}}^{LV}$ and $\overline{\mathbf{V}}^{RV}$ be the left and right sub-graphs resulting from a split in the v direction, respectively. With these definitions, the recursive subdivision process for approximating a surface by a set of polygons can be stated as:

```
procedure Approximate_Surface( $\overline{\mathbf{V}}$, $\epsilon$)
local variables $\overline{\mathbf{V}}^{LU}$, $\overline{\mathbf{V}}^{RU}$, $\overline{\mathbf{V}}^{LV}$, $\overline{\mathbf{V}}^{RV}$;
begin
  if $\overline{\mathbf{V}}$ is flat within $\epsilon$ along $u$ then
    if $\overline{\mathbf{V}}$ is flat within $\epsilon$ along $v$ then
      Polygon_Approximate( $\overline{\mathbf{V}}$);
    else
      Subdivide_Along_v( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LV}$, $\overline{\mathbf{V}}^{RV}$);
      Approximate_Surface( $\overline{\mathbf{V}}^{LV}$, $\epsilon$);
      Approximate_Surface( $\overline{\mathbf{V}}^{RV}$, $\epsilon$);
    endif
  else
    Subdivide_Along_u( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LU}$, $\overline{\mathbf{V}}^{RU}$);
    Approximate_Surface( $\overline{\mathbf{V}}^{LU}$, $\epsilon$);
    Approximate_Surface( $\overline{\mathbf{V}}^{RU}$, $\epsilon$);
  endif;
end;
```

The flatness of a control graph can be tested by computing a plane containing three of the four corner vertices of the graph. The perpendicular distances of the other vertices to this plane are then computed. The graph passes the flatness test if the largest such distance is less than the tolerance ϵ (see figure 8).

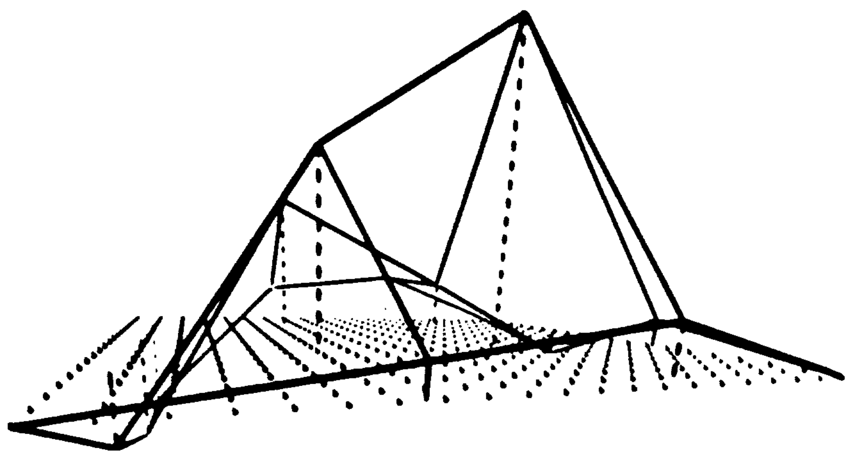

Figure 8.
Flatness testing for a control graph. Three arbitrarily chosen corner vertices are used to define a plane (shown dotted). The perpendicular distances from the other vertices in the graph to this plane are computed (shown dashed). If the largest such distance is less than the tolerance ϵ, the graph is deemed flat.

6.3. Subdivision of Beta2-spline Curves

6.3.1. Approach 1

Section 6.1 presented the general scheme for the subdivision of spline curves that possess the convex hull property. In this section, we develop the method by which a Beta2-spline curve can be subdivided into left and right sub-polygons. We are, in essence, constructing the procedures *Subdivide* and *Linearly_Approximate* referred to in section 6.1. However, this section does not go into detail concerning the derivation of the mathematics upon which these procedures are based. The interested reader is referred to Barsky[5] for a more complete development.

We focus attention on the single segment $\mathbf{Q}_2(\beta_2;u)$ of the Beta2-spline curve, and for convenience we drop the subscript on $\mathbf{Q}_2(\beta_2;u)$ and just write $\mathbf{Q}(\beta_2;u)$. A multi-segment curve can be handled on a segment by segment basis using the algorithms given in this section.

Let $\overline{\mathbf{V}}$ represent a column matrix composed of the vertices defining $\mathbf{Q}(\beta_2;u)$, i.e.

$$\overline{\mathbf{V}} = \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} \qquad \text{(VI.1)}$$

Since each element of $\overline{\mathbf{V}}$ is itself a vector, (VI.1) is a shorthand for several scalar-valued column matrices. For instance, if the vertices are two-dimensional, then $\overline{\mathbf{V}}$ can be expressed as

$$\overline{\mathbf{V}} = \left(\begin{bmatrix} V_0^x \\ V_1^x \\ V_2^x \\ V_3^x \end{bmatrix}, \begin{bmatrix} V_0^y \\ V_1^y \\ V_2^y \\ V_3^y \end{bmatrix} \right) \qquad \text{(VI.2)}$$

Using this representation, Barsky et al[5] have shown that the left and right sub-polygons $\overline{\mathbf{V}}^L$ and $\overline{\mathbf{V}}^R$, respectively, are given by

$$\overline{\mathbf{V}}^L = \begin{bmatrix} \mathbf{V}_0^L \\ \mathbf{V}_1^L \\ \mathbf{V}_2^L \\ \mathbf{V}_3^L \end{bmatrix} = \overline{\mathbf{L}}_\beta \overline{\mathbf{V}} \quad \text{and} \quad \overline{\mathbf{V}}^R = \begin{bmatrix} \mathbf{V}_0^R \\ \mathbf{V}_1^R \\ \mathbf{V}_2^R \\ \mathbf{V}_3^R \end{bmatrix} = \overline{\mathbf{R}}_\beta \overline{\mathbf{V}} \qquad \text{(VI.3)}$$

where, for a Beta2-spline, the $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ matrices are given by

$$\overline{\mathbf{L}}_\beta = \begin{bmatrix} \gamma\frac{(\beta_2^2+17\beta_2+48)}{2(\beta_2+4)} & \frac{3\gamma(\beta_2+8)}{4} & -\frac{\gamma\beta_2(\beta_2+6)}{4(\beta_2+4)} & 0 \\ \gamma\frac{(\beta_2+6)}{(\beta_2+4)} & \gamma(\beta_2+9) & \frac{2\gamma(\beta_2+3)}{(\beta_2+4)} & 0 \\ \gamma\frac{\beta_2}{2(\beta_2+4)} & \frac{3\gamma(\beta_2+8)}{4} & \gamma\frac{(\beta_2^2+26\beta_2+96)}{4(\beta_2+4)} & 0 \\ -\frac{\gamma\beta_2(\beta_2+8)}{8(\beta_2+4)} & -\gamma\frac{(\beta_2^2+9\beta_2-12)}{8} & \gamma\frac{(\beta_2^3+21\beta_2^2+144\beta_2+288)}{8(\beta_2+4)} & \frac{1}{8} \end{bmatrix} \qquad \text{(VI.4)}$$

and

$$\overline{\mathbf{R}}_\beta = \begin{bmatrix} \frac{1}{8} & \gamma\frac{(\beta_2^3+21\beta_2^2+144\beta_2+288)}{8(\beta_2+4)} & -\gamma\frac{(\beta_2^2+9\beta_2-12)}{8} & -\frac{\gamma\beta_2(\beta_2+8)}{8(\beta_2+4)} \\ 0 & \gamma\frac{(\beta_2^2+26\beta_2+96)}{4(\beta_2+4)} & \frac{3\gamma(\beta_2+8)}{4} & \gamma\frac{\beta_2}{2(\beta_2+4)} \\ 0 & \frac{2\gamma(\beta_2+3)}{(\beta_2+4)} & \gamma(\beta_2+9) & \gamma\frac{(\beta_2+6)}{(\beta_2+4)} \\ 0 & -\frac{\gamma\beta_2(\beta_2+6)}{4(\beta_2+4)} & \frac{3\gamma(\beta_2+8)}{4} & \gamma\frac{(\beta_2^2+17\beta_2+48)}{2(\beta_2+4)} \end{bmatrix} \qquad \text{(VI.5)}$$

If the vertices are two-dimensional, because of our shorthand form, equation (VI.3) written in component form becomes

$$\overline{\mathbf{V}}^L = (\overline{\mathbf{L}}_\beta V^x, \overline{\mathbf{L}}_\beta V^y) = (\overline{\mathbf{L}}_\beta \begin{bmatrix} V_0^x \\ V_1^x \\ V_2^x \\ V_3^x \end{bmatrix}, \overline{\mathbf{L}}_\beta \begin{bmatrix} V_0^y \\ V_1^y \\ V_2^y \\ V_3^y \end{bmatrix}) \qquad \text{(VI.6)}$$

The matrices $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ are called the Beta2-spline *midpoint subdivision matrices.* It is also possible to do non-midpoint subdivision,[5,2,9] but for our purposes midpoint subdivision suffices. Notice that $\overline{\mathbf{R}}_\beta$ is a permutation of $\overline{\mathbf{L}}_\beta$, meaning that only the elements of $\overline{\mathbf{L}}_\beta$ need to be computed. This observation leads to the following procedures for the evaluation of $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$:

```
procedure compute_L( $\beta_2$, L)
local variable $\beta_2^2$, $t1$, $\gamma_1$,$\gamma_2$,$\gamma_3$, $\gamma_4$, $\gamma_5$;
begin
  $\beta_2^2 := \beta_2\beta_2;$
  $t_1 := \beta_2 + 8;$
  $\gamma_1 := 1/(\beta_2 + 12);$
  $\gamma_2 := \gamma_1/(\beta_2 + 4);$
  $\gamma_3 := \gamma_2/2;$
  $\gamma_4 := \gamma_3/2;$
  $\gamma_5 := \gamma_4/2;$

  $L_{1,1} := \gamma_3(\beta_2^2 + 17\beta_2 + 48);$
  $L_{2,1} := \gamma_2(\beta_2 + 6);$
  $L_{3,1} := \gamma_3\beta_2;$
  $L_{4,1} := -\beta_2 t_1 \gamma_5;$

  $L_{1,2} := 0.75 t_1 \gamma_4;$
  $L_{2,2} := \gamma_1(\beta_2 + 9);$
  $L_{3,2} := L_{1,2};$
  $L_{4,2} := -0.125\gamma_1(\beta_2^2 + 9\beta_2 - 12);$

  $L_{1,3} := -0.25\beta_2 L_{3,1};$
  $L_{2,3} := 2\gamma_2(\beta_2 + 3);$
  $L_{3,3} := \gamma_4(\beta_2^2 + 26\beta_2 + 96);$
  $L_{4,3} := \gamma_5(\beta_2(\beta_2^2 + 21\beta_2 + 144) + 288);$

  $L_{4,4} := 0.125;$
end;

procedure compute_R( L, R)
begin
  for $i$ = 1 to 4 do
    for $j$ = 1 to 4 do
      $R_{i,j} := L_{5-i,5-j};$
end;
```

These procedures require a total of 15 additions / subtractions, 22 multiplications, and 5 divisions. Once the $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ matrices are computed in the initialization step, the procedure *Subdivide_β_2* given below can be used to split a Beta2-spline curve segment into left and right subpolygons.

```
procedure Subdivide_$\beta_2$( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^L$, $\overline{\mathbf{V}}^R$)
begin
  $\mathbf{V}_0^L := L_{1,1}\mathbf{V}_0 + L_{1,2}\mathbf{V}_1 + L_{1,3}\mathbf{V}_2;$
  $\mathbf{V}_1^L := L_{2,1}\mathbf{V}_0 + L_{2,2}\mathbf{V}_1 + L_{2,3}\mathbf{V}_2;$
  $\mathbf{V}_2^L := L_{3,1}\mathbf{V}_0 + L_{3,2}\mathbf{V}_1 + L_{3,3}\mathbf{V}_2;$
  $\mathbf{V}_3^L := L_{4,1}\mathbf{V}_0 + L_{4,2}\mathbf{V}_1 + L_{4,3}\mathbf{V}_2 + L_{4,4}\mathbf{V}_3;$

  $\mathbf{V}_0^R := R_{1,1}\mathbf{V}_0 + R_{1,2}\mathbf{V}_1 + R_{1,3}\mathbf{V}_2 + R_{1,4}\mathbf{V}_3;$
  $\mathbf{V}_1^R := R_{2,2}\mathbf{V}_1 + R_{2,3}\mathbf{V}_2 + R_{2,4}\mathbf{V}_3;$
  $\mathbf{V}_2^R := R_{3,2}\mathbf{V}_1 + R_{3,3}\mathbf{V}_2 + R_{3,4}\mathbf{V}_3;$
  $\mathbf{V}_3^R := R_{4,2}\mathbf{V}_1 + R_{4,3}\mathbf{V}_2 + R_{4,4}\mathbf{V}_3;$
end;
```

Since each expression in the procedure above is vector-valued, most programming languages require the corresponding scalar expressions for each spatial dimension. Thus, if d denotes the number of spatial dimensions, the above procedure requires $18d$ additions/subtractions, $26d$ multiplications, and no divisions.

We must now determine how to approximate the Beta2-spline curve generated by a flat control polygon. As was mentioned above, if the curve is known to interpolate the endpoints of the control polygon, then a good linear characterization is the line connecting the endpoints. A Beta2-spline curve does not interpolate the endpoints, however. Instead, we use a line connecting the starting point of the curve $\mathbf{Q}(\beta_2;0)$ and the ending point $\mathbf{Q}(\beta_2;1)$. The starting point is

$$\mathbf{Q}(\beta_2;0) = \mathbf{V}_0 b_{-2}(\beta_2;0) + \mathbf{V}_1 b_{-1}(\beta_2;0) + \mathbf{V}_2 b_0(\beta_2;0) + \mathbf{V}_3 b_1(\beta_2;0) \qquad \text{(VI.7)}$$

Using (III.11) this expression reduces to

$$\mathbf{Q}(\beta_2;0) = \gamma(2(\mathbf{V}_0 + \mathbf{V}_2) + (\beta_2 + 8)\mathbf{V}_1) \qquad \text{(VI.8)}$$

For computational efficiency, we introduce the quantities

$$\tau_1 = 2\gamma$$

$$\tau_2 = \gamma(\beta_2 + 8)$$

Equation (VI.8) can then be rewritten as

$$\mathbf{Q}(\beta_2;0) = \tau_1(\mathbf{V}_0 + \mathbf{V}_2) + \tau_2\mathbf{V}_1$$

The ending point $\mathbf{Q}(\beta_2;1)$ can similarly be shown to be given by

$$\mathbf{Q}(\beta_2;1) = \gamma(2(\mathbf{V}_1 + \mathbf{V}_3) + (\beta_2 + 8)\mathbf{V}_2) = \tau_1(\mathbf{V}_{-1} + \mathbf{V}_1) + \tau_2\mathbf{V}_0 \qquad \text{(VI.10)}$$

The coefficients τ_1 and τ_2 only need to be computed once, during the initialization step; pseudo-code for their evaluation is

```
procedure Compute_tau( $\beta_2$, $\tau_1$, $\tau_2$)
local variable $\gamma$;
begin
    $\gamma := 1/(\beta_2 + 12)$;
    $\tau_1 := 2\gamma$;
    $\tau_2 := \gamma(\beta_2 + 8)$;
end;
```

Once the taus have been computed, the following procedure can be used to approximate a relatively flat control polygon by a single line segment:

```
procedure Linearly_Approximate_$\beta_2$( $\overline{\mathbf{V}}$, $\tau_1$, $\tau_2$)
local variable $\mathbf{P}_1$, $\mathbf{P}_2$;
begin
    $\mathbf{P}_1 := \tau_1(\mathbf{V}_{-2} + \mathbf{V}_0) + \tau_2\mathbf{V}_{-1}$;
    $\mathbf{P}_2 := \tau_1(\mathbf{V}_{-1} + \mathbf{V}_1) + \tau_2\mathbf{V}_0$;

    Output_Segment( $\mathbf{P}_1$, $\mathbf{P}_2$);
end;
```

The procedure *Output_Segment* outputs a line from the first argument to the second argument. Finally, the algorithm that approximates a Beta2-spline curve using Approach 1 can be stated as

```
procedure Approximate_Curve_$\beta_2$_1( $\overline{\mathbf{V}}$, $\epsilon$, $\tau_1$, $\tau_2$)
local variable $\mathbf{V}^L$, $\mathbf{V}^R$;
begin
   if $\overline{\mathbf{V}}$ is flat to within $\epsilon$ then
      Linearly_Approximate_$\beta_2$_1( $\overline{\mathbf{V}}$, $\tau_1$, $\tau_2$);
   else
      Subdivide_$\beta_2$( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^L$, $\overline{\mathbf{V}}^R$);
      Approximate_Curve_$\beta_2$_1( $\overline{\mathbf{V}}^L$, $\epsilon$, $\tau_1$, $\tau_2$);
      Approximate_Curve_$\beta_2$_1( $\overline{\mathbf{V}}^R$, $\epsilon$, $\tau_1$, $\tau_2$);
   endif;
end;


Compute_L( $\beta_2$, $L$ );
Compute_R( $L$ , $R$ );
Compute_tau( $\beta_2$, $\tau_1$, $\tau_2$);
Approximate_Curve_$\beta_2$_1( $\overline{\mathbf{V}}$, $\epsilon$, $\tau_1$, $\tau_2$);
```

Operation Count

The calls to *Compute_L*, *Compute_R*, and *Compute_tau* combine to require 17 additions / subtractions, 24 multiplications, and 6 divisions. For vertices of dimension d, k calls to *Subdivide_*β_2 costs $18dk$ additions / subtractions, $26dk$ multiplications, and no divisions. If there are k calls to *Subdivide_*β_2, there must be $k+1$ calls to the linear approximation routine *Linearly_Approximate_*β_2. Thus, using Approach 1, k subdivisions of vertices of spatial dimension d will require $17+4d+22kd$ additions / subtractions, $24+4d+30kd$ multiplications, and 6 divisions. We are purposely ignoring the details of the polygon flatness testing as this is dealt with in detail in Barsky.[5]

6.3.2. Approach 2

Section 6.3.1 presented what could be called the straightforward approach to the approximation of a Beta2-spline curve by a set of linear segments. However, there is a more convenient and computationally more efficient method of linear approximation. This method, which we call Approach 2, converts the Beta2-spline segment to be subdivided into a cubic Be'zier curve. Although recent work allows Be'zier subdivision at any arbitrary parametric value,[2,9] we use the efficient midpoint subdivision algorithm of Lane and Riesenfeld[13] Not only is the procedure Linearly_Approximate trivial for Be'zier curves, but the procedure Subdivide is also more efficient computationally. Thus, Approach 2 can be stated as

```
procedure Approximate_Curve_$\beta_2$_2( $\overline{\mathbf{V}}$, $\beta_2$, $\epsilon$)
local variable $\overline{\mathbf{W}}$;
begin
   Map_$\beta_2$_Curve_To_Be'zier( $\overline{\mathbf{V}}_\beta$, $\beta_2$, $\overline{\mathbf{W}}$);
   Approximate_Be'zier( $\overline{\mathbf{W}}$, $\epsilon$);
end;
```

The procedure Map_β_2_Curve_To_Be'zier converts its first argument, assumed to represent a Beta2-spline control polygon, into a Be'zier control polygon, returned as the third argument. The Be'zier polygon is constructed so that it will generate the same curve when blended with a Be'zier basis set as the Beta2-spline control polygon generates when blended with the Beta2-spline basis set. Mathematically stated, if $(\overline{\mathbf{V}}_0,\overline{\mathbf{V}}_1,\overline{\mathbf{V}}_2,\overline{\mathbf{V}}_3)$ are the vertices of the Beta2-spline control polygon, and $(\overline{\mathbf{W}}_0,\overline{\mathbf{W}}_1,\overline{\mathbf{W}}_2,\overline{\mathbf{W}}_3)$ are the vertices of the Be'zier control polygon, then we require that

$$\sum_{r=-2}^{1} \overline{\mathbf{V}}_{r+2} b_r(\beta_2; u) = \sum_{r=0}^{3} \overline{\mathbf{W}}_r B_r(u) \qquad \text{(VI.11)}$$

where the functions $B_0(u), B_1(u), B_2(u), B_3(u)$ are the cubic Be'zier basis functions:

$$B_i(u) = \binom{3}{i} u^i (1-u)^{3-i} \quad i = 0,1,2,3 \tag{VI.12}$$

Equation (VI.10) can alternately be written in matrix form as

$$[b_{-2}(\beta_2;u) \;\; b_{-1}(\beta_2;u) \;\; b_0(\beta_2;u) \;\; b_1(\beta_2;u)] \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = \tag{VI.13}$$

$$[B_0(u) \;\; B_1(u) \;\; B_2(u) \;\; B_3(u)] \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix}$$

Since both sets of basis functions are cubic polynomials, we can write them as

$$[b_{-2}(\beta_2;u) \;\; b_{-1}(\beta_2;u) \;\; b_0(\beta_2;u) \;\; b_1(\beta_2;u)] = [1 \;\; u \;\; u^2 \;\; u^3]\overline{\mathbf{C}} \tag{VI.14}$$

and

$$[B_0(u) \;\; B_1(u) \;\; B_2(u) \;\; B_3(u)] = [1 \;\; u \;\; u^2 \;\; u^3]\overline{\mathbf{D}} \tag{VI.15}$$

where $\overline{\mathbf{C}}$ is defined in (III.10) and the matrix $\overline{\mathbf{D}}$ is the Be´zier coefficient matrix; specifically

$$\overline{\mathbf{D}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \tag{VI.16}$$

Substituting (VI.15) and (VI.16) into (VI.13) yields

$$[1 \;\; u \;\; u^2 \;\; u^3]\overline{\mathbf{C}} \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = [1 \;\; u \;\; u^2 \;\; u^3]\overline{\mathbf{D}} \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix} \tag{VI.17}$$

Since the powers of u are linearly independent, equation (VI.17) can hold if and only if

$$\overline{\mathbf{W}} = \overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}\overline{\mathbf{V}} \tag{VI.18}$$

where

$$\overline{\mathbf{W}} = \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix} \quad \overline{\mathbf{V}} = \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} \tag{VI.19}$$

Thus, equation (VI.18) defines the mapping between a Beta2-spline control polygon and a cubic Be´zier control polygon. The matrix that does this mapping $\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}$ can be written in component form (using Vaxima) as

$$\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}} = \begin{bmatrix} \tau_1 & \tau_2 & \tau_1 & 0 \\ 0 & \tau_2 & 2\tau_1 & 0 \\ 0 & 2\tau_1 & \tau_2 & 0 \\ 0 & \tau_1 & \tau_2 & \tau_1 \end{bmatrix} \tag{VI.20}$$

We now present the procedure that accomplishes the mapping:

```
procedure Map_$\beta_2$_Curve_To_Bézier( $\overline{\mathbf{V}}$, $\beta_2$, $\overline{\mathbf{W}}$)
local variable $\tau_1$, $two\tau_1$, $\tau_2$, $\gamma$;
begin
   $\gamma := 1/(\beta_2 + 12)$;
   $\tau_1 := 2\gamma$;
   $two\tau_1 := 2\tau_1$;
   $\tau_2 := (\beta_2 + 8)\gamma$;
   $\mathbf{W}_0 := \tau_1(\mathbf{V}_0 + \mathbf{V}_2) + \tau_2\mathbf{V}_1$;
   $\mathbf{W}_1 := two\tau_1\mathbf{V}_2 + \tau_2\mathbf{V}_1$;
   $\mathbf{W}_2 := two\tau_1\mathbf{V}_1 + \tau_2\mathbf{V}_2$;
   $\mathbf{W}_3 := \tau_1(\mathbf{V}_1 + \mathbf{V}_3) + \tau_2\mathbf{V}_2$;
end;
```

This procedure requires $2+6d$ additions / subtractions, $3+8d$ multiplications, and 1 division for d dimensional vertices.

The procedure to subdivide a cubic Bézier curve at the parametric midpoint is adapted from Lane and Riesenfeld[13] and can be written as

```
procedure Subdivide_Bézier( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^L$, $\overline{\mathbf{W}}^R$ )
begin
   $\mathbf{W}_0^L := \mathbf{W}_0$;
   $\mathbf{W}_1^L := 0.5(\mathbf{W}_0 + \mathbf{W}_1)$;
   $\mathbf{W}_2^L := 0.25(\mathbf{W}_0 + \mathbf{W}_2) + 0.5\mathbf{W}_1$;
   $\mathbf{W}_3^L := 0.125(\mathbf{W}_0 + \mathbf{W}_3) + 0.375(\mathbf{W}_1 + \mathbf{W}_2)$;

   $\mathbf{W}_0^R := \mathbf{W}_3^L$;
   $\mathbf{W}_0^R := 0.25(\mathbf{W}_1 + \mathbf{W}_3) + 0.5\mathbf{W}_2$;
   $\mathbf{W}_0^R := 0.5(\mathbf{W}_2 + \mathbf{W}_3)$;
   $\mathbf{W}_0^R := \mathbf{W}_3$;
end;
```

This procedure requires $9d$ additions / subtractions, $8d$ multiplications, and no divisions for vertices of spatial dimensional d.

Since a Bézier curve interpolates the end vertices of its control polygon, the procedure *Linearly_Approximate* that generates a single line segment to approximate a Bézier curve is simply the routine *Ouput_Segment*. Thus, Approach 2 may be stated as:

```
procedure Approximate_Bézier_Curve( $\overline{\mathbf{W}}$, $\epsilon$)
local variable $\overline{\mathbf{W}}^L$, $\overline{\mathbf{W}}^R$;
begin
   if $\overline{\mathbf{W}}$ is flat to within $\epsilon$ then
      Output_Segment( $\mathbf{W}_0$, $\mathbf{W}_3$);
   else
      Subdivide_Bézier( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^L$, $\overline{\mathbf{W}}^R$);
      Approximate_Bézier_Curve( $\overline{\mathbf{W}}^L$, $\epsilon$);
      Approximate_Bézier_Curve( $\overline{\mathbf{W}}^R$, $\epsilon$);
   endif
end;
```

Operation Count

Only one call to *Map_β_2_*Curve_To_Bézier is needed; this costs $2+6d$ additions / subtractions, $3+8d$ multiplications, and 1 division. k calls to *Subdivide_Bézier* will cost $9dk$ additions / subtractions, $8dk$ multiplications, and no divisions. Thus, the total cost for k subdivisions of a Beta2-spline curve, using Approach 2, requires a total of $2+6d+9dk$ additions / subtractions, $3+8d+8dk$ multiplications, and 1 division.

6.4. Subdivision of Beta2-spline Surfaces

Just as the algorithms for the subdivision of Beta2-spline curves dealt with a single curve segment, the algorithms to be presented in this section deal with a single surface patch. For simplicity we consider the patch $\mathbf{S}_{2,2}(\beta_2;u,v)$, normally written without subscripts as $\mathbf{S}(\beta_2;u,v)$. The control graph for $\mathbf{S}(\beta_2;u,v)$ will be denoted by $\overline{\mathbf{V}}$ and written as the 4x4 matrix

$$\overline{\mathbf{V}} = \begin{bmatrix} \mathbf{V}_{0,0} & \mathbf{V}_{0,1} & \mathbf{V}_{0,2} & \mathbf{V}_{0,3} \\ \mathbf{V}_{1,0} & \mathbf{V}_{1,1} & \mathbf{V}_{1,2} & \mathbf{V}_{1,3} \\ \mathbf{V}_{2,0} & \mathbf{V}_{2,1} & \mathbf{V}_{2,2} & \mathbf{V}_{2,3} \\ \mathbf{V}_{3,0} & \mathbf{V}_{3,1} & \mathbf{V}_{3,2} & \mathbf{V}_{3,3} \end{bmatrix} \tag{VI.21}$$

6.4.1. Approach 1

Approach 1 for surfaces follows the same general form as Approach 1 for curves; a Beta2-spline surface patch is approximated by polygons by using the general schema for surfaces given in section 6.2.

We begin by developing the procedure which will split the control graph $\overline{\mathbf{V}}$ in the u direction to produce the sub-graphs $\overline{\mathbf{V}}^{LU}$ and $\overline{\mathbf{V}}^{RU}$. Barsky et al[5] have shown that $\overline{\mathbf{V}}^{LU}$ and $\overline{\mathbf{V}}^{RU}$ can be computed from

$$\overline{\mathbf{V}}^{LU} = \overline{\mathbf{L}}_\beta \overline{\mathbf{V}} \tag{VI.22}$$
$$\overline{\mathbf{V}}^{RU} = \overline{\mathbf{R}}_\beta \overline{\mathbf{V}} \tag{VI.23}$$

where $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ are the Beta2-spline midpoint subdivision matrices from (VI.4) and (VI.5), respectively.

A straightforward algorithm to implement the above mathematics could either call a routine that multiplies 4x4 matrices, or the expression for each element could be written out. However, simple inspection shows that we can reuse the curve subdivision routine Subdivide_β_2 to accomplish our goal. Equation (VI.22) shows that the columns of $\overline{\mathbf{V}}^{LU}$ can be found by applying $\overline{\mathbf{L}}_\beta$ to the columns of $\overline{\mathbf{V}}$ – but this is exactly what Subdivide_β_2 does. The same is true for the right sub-graph $\overline{\mathbf{V}}^{RU}$. Thus, a subdivision of a control graph in the u direction can be done by calling Subdivide_β_2 on the columns of $\overline{\mathbf{V}}$. To help make this clearer in the algorithm, let $Col_i(\overline{\mathbf{M}})$ denote the i^{th} column of matrix $\overline{\mathbf{M}}$. The procedure can now be stated as

```
procedure Subdivide_$\beta_2$_Along_u( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LU}$, $\overline{\mathbf{V}}^{RU}$)
local variable i;
begin
   for i = 1 to 4 do
      Subdivide_$\beta_2$( $Col_i(\overline{\mathbf{V}})$, $Col_i(\overline{\mathbf{V}}^{LU})$, $Col_i(\overline{\mathbf{V}}^{RU})$);
end;
```

Barsky et al[5] also show that a split in the v direction is described by†

$$\overline{\mathbf{V}}^{LV} = \overline{\mathbf{V}}\overline{\mathbf{L}}_\beta^T \tag{VI.24}$$

and

$$\overline{\mathbf{V}}^{RV} = \overline{\mathbf{V}}\overline{\mathbf{R}}_\beta^T \tag{VI.25}$$

Once again, we could use a routine to multiply 4x4 matrices or write out the expression for each element of $\overline{\mathbf{V}}^{LV}$ and $\overline{\mathbf{V}}^{RV}$. However, examination of equations (VI.24) and (VI.25) reveals that the rows of $\overline{\mathbf{V}}^{LV}$ are the result of applying $\overline{\mathbf{L}}_\beta$ to the rows of $\overline{\mathbf{V}}$. The same is true of $\overline{\mathbf{V}}^{RV}$. Thus, a subdivision of a control graph in the v direction can be accomplished by applying the matrices $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ to the rows of $\overline{\mathbf{V}}$. The one slight complication is that *Subdivide_*β_2 expects to receive column matrices, not row matrices; the transpose operator can be used to relieve this type-clash. To make the following algorithm clearer, let $Row_i(\overline{\mathbf{M}})$ denote the i^{th} row of matrix $\overline{\mathbf{M}}$.

† Superscript T denotes the transpose operation.

```
procedure Subdivide_$\beta_2$_Along_v( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LV}$, $\overline{\mathbf{V}}^{RV}$)
local variable i;
begin
    for i = 1 to 4 do
        Subdivide_$\beta_2$( $Row_i(\overline{\mathbf{V}})^T$, $Row_i(\overline{\mathbf{V}}^{LV})^T$, $Row_i(\overline{\mathbf{V}}^{RV})^T$);
end;
```

Next we examine how the surface patch defined by a relatively flat control graph can be approximated by polygons. The complication here is that a Beta2-spline surface patch does not interpolate the corner vertices of the control graph, so the corner points of the surface must be computed explicitly from (I.3). Since $\mathbf{S}(\beta_2;u,v)$ is defined as

$$\mathbf{S}(\beta_2;u,v) = \sum_{r=-2}^{1}\sum_{s=-2}^{1} \mathbf{V}_{r+2,s+2}b_r(\beta_2;u)b_s(\beta_2;v) \tag{VI.26}$$

the corner points $\mathbf{S}(\beta_2;0,0)$, $\mathbf{S}(\beta_2;0,1)$, $\mathbf{S}(\beta_2;1,0)$, and $\mathbf{S}(\beta_2;1,1)$, denoted as $\mathbf{P}_{0,0}$, $\mathbf{P}_{0,1}$, $\mathbf{P}_{1,0}$, and $\mathbf{P}_{1,1}$, respectively, reduce to

$$\mathbf{P}_{0,0} = \tau_1[\tau_1(\mathbf{V}_{0,0}+\mathbf{V}_{0,2}+\mathbf{V}_{2,0}+\mathbf{V}_{2,2})+\tau_2(\mathbf{V}_{0,1}+\mathbf{V}_{2,1})]+ \\ \tau_2[\tau_1(\mathbf{V}_{1,0}+\mathbf{V}_{1,2})+\tau_2\mathbf{V}_{1,1}] \tag{VI.27}$$

$$\mathbf{P}_{0,1} = \tau_1[\tau_1(\mathbf{V}_{0,1}+\mathbf{V}_{2,1}+\mathbf{V}_{0,3}+\mathbf{V}_{2,3})+\tau_2(\mathbf{V}_{1,1}+\mathbf{V}_{1,3})]+ \\ \tau_2[\tau_1(\mathbf{V}_{2,2}+\mathbf{V}_{2,2})+\tau_2\mathbf{V}_{1,2}] \tag{VI.28}$$

$$\mathbf{P}_{1,0} = \tau_1[\tau_1(\mathbf{V}_{1,0}+\mathbf{V}_{1,2}+\mathbf{V}_{3,0}+\mathbf{V}_{3,2})+\tau_2(\mathbf{V}_{1,1}+\mathbf{V}_{3,1})]+ \\ \tau_2[\tau_1(\mathbf{V}_{2,0}+\mathbf{V}_{2,2})+\tau_2\mathbf{V}_{2,1}] \tag{VI.29}$$

$$\mathbf{P}_{1,1} = \tau_1[\tau_1(\mathbf{V}_{1,1}+\mathbf{V}_{1,3}+\mathbf{V}_{3,1}+\mathbf{V}_{3,3})+\tau_2(\mathbf{V}_{1,2}+\mathbf{V}_{3,2})]+ \\ \tau_2[\tau_1(\mathbf{V}_{2,1}+\mathbf{V}_{2,3})+\tau_2\mathbf{V}_{2,2}] \tag{VI.30}$$

where τ_1 and τ_2 are as defined in equation (VI.9).

One way of approximating the surface would be to construct a quadrilateral connecting the corner points $\mathbf{P}_{0,0}$, $\mathbf{P}_{0,1}$, $\mathbf{P}_{1,0}$, and $\mathbf{P}_{1,1}$. However, since the points may not be coplanar, the resulting polygon may be non-planar. The planarity can be remedied by using two triangles instead of one quadrilateral. Unfortunately, this solution can introduce visual asymmetries since it treats the diagonals of the quadrilateral with unequal preference.[5]

A better approach is to average the corner points of the surface to obtain a fifth point $\mathbf{P}_a$. The four triangles $(\mathbf{P}_{0,0},\mathbf{P}_{1,0},\mathbf{P}_a)$, $(\mathbf{P}_{1,0},\mathbf{P}_{1,1},\mathbf{P}_a)$, $(\mathbf{P}_{1,1},\mathbf{P}_{0,1},\mathbf{P}_a)$, and $(\mathbf{P}_{0,1},\mathbf{P}_{0,0},\mathbf{P}_a)$ are then used to approximate the surface. This method does not introduce asymmetries since both diagonals of the quadrilateral are treated equally. Pseudo-code for the approximation of a flat Beta2-spline by four triangles is:

```
procedure Polygon_Approximate_$\beta_2$( $\overline{\mathbf{V}}$, $\beta_2$)
local variable $\mathbf{P}_{0,0}$, $\mathbf{P}_{0,1}$, $\mathbf{P}_{1,0}$, $\mathbf{P}_{1,1}$, $\mathbf{P}_a$;
begin
    { Compute the corner points }
    $\mathbf{P}_{0,0} = \tau_1[\tau_1(\mathbf{V}_{-2,-2}+\mathbf{V}_{-2,0}+\mathbf{V}_{0,-2}+\mathbf{V}_{0,0})+\tau_2(\mathbf{V}_{-2,-1}+\mathbf{V}_{-0,-1})]+$
            $\tau_2[\tau_1(\mathbf{V}_{-1,-2}+\mathbf{V}_{-1,0})+\tau_2\mathbf{V}_{-1,-1}]$;
    $\mathbf{P}_{0,1} = \tau_1[\tau_1(\mathbf{V}_{-2,-1}+\mathbf{V}_{0,-1}+\mathbf{V}_{-2,1}+\mathbf{V}_{0,1})+\tau_2(\mathbf{V}_{-1,-1}+\mathbf{V}_{-1,1})]+$
            $\tau_2[\tau_1(\mathbf{V}_{-2,0}+\mathbf{V}_{0,0})+\tau_2\mathbf{V}_{-1,0}]$;
    $\mathbf{P}_{1,0} = \tau_1[\tau_1(\mathbf{V}_{-1,-2}+\mathbf{V}_{-1,0}+\mathbf{V}_{1,-2}+\mathbf{V}_{1,0})+\tau_2(\mathbf{V}_{-1,-1}+\mathbf{V}_{1,-1})]+$
            $\tau_2[\tau_1(\mathbf{V}_{0,-2}+\mathbf{V}_{0,0})+\tau_2\mathbf{V}_{0,-1}]$;
    $\mathbf{P}_{1,1} = \tau_1[\tau_1(\mathbf{V}_{-1,-1}+\mathbf{V}_{-1,1}+\mathbf{V}_{1,-1}+\mathbf{V}_{1,1})+\tau_2(\mathbf{V}_{-1,0}+\mathbf{V}_{1,0})]+$
            $\tau_2[\tau_1(\mathbf{V}_{0,-1}+\mathbf{V}_{0,1})+\tau_2\mathbf{V}_{0,0}]$;

    $\mathbf{P}_a = 0.25(\mathbf{P}_{0,0}+\mathbf{P}_{0,1}+\mathbf{P}_{1,0}+\mathbf{P}_{1,1})$;

    Output_Triangle( $\mathbf{P}_{0,0}$, $\mathbf{P}_{1,0}$, $\mathbf{P}_a$);
    Output_Triangle( $\mathbf{P}_{1,0}$, $\mathbf{P}_{1,1}$, $\mathbf{P}_a$);
    Output_Triangle( $\mathbf{P}_{1,1}$, $\mathbf{P}_{0,1}$, $\mathbf{P}_a$);
    Output_Triangle( $\mathbf{P}_{0,1}$, $\mathbf{P}_{0,0}$, $\mathbf{P}_a$);
end;
```

Polygon_Approximate_β_2 requires $35d$ additions / subtractions, $25d$ multiplications, and no divisions for d dimensional vertices.

If $\overline{\mathbf{V}}$ represents the control graph for the surface to be approximated, then the following pseudo-code describes the process necessary to obtain a polygonal approximation to $\overline{\mathbf{V}}$ using Approach 1.

```
Compute_L( $\beta_2$, L);
Compute_R( L, R);
Compute_tau( $\beta_2$, $\tau_1$, $\tau_2$);
Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}$, $\epsilon$, $\tau_1$, $\tau_2$);
```

where

```
procedure Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}$, $\epsilon$, $\tau_1$, $\tau_2$)
local variables $\overline{\mathbf{V}}^{LU}$, $\overline{\mathbf{V}}^{RU}$, $\overline{\mathbf{V}}^{LV}$, $\overline{\mathbf{V}}^{RV}$;
begin
    if $\overline{\mathbf{V}}$ is flat to within $\epsilon$ along $u$ then
        if $\overline{\mathbf{V}}$ is flat within $\epsilon$ along $v$ then
            Polygon_Approximate_$\beta_2$( $\overline{\mathbf{V}}$, $\tau_1$, $\tau_2$);
        else
            Subdivide_$\beta_2$_Along_v( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LV}$, $\overline{\mathbf{V}}^{RV}$);
            Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}^{LV}$, $\epsilon$, $\tau_1$, $\tau_2$);
            Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}^{RV}$, $\epsilon$, $\tau_1$, $\tau_2$);
        endif
    else
        Subdivide_$\beta_2$_Along_u( $\overline{\mathbf{V}}$, $\overline{\mathbf{V}}^{LU}$, $\overline{\mathbf{V}}^{RU}$);
        Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}^{LU}$, $\epsilon$, $\tau_1$, $\tau_2$);
        Approximate_$\beta_2$_Surface_1( $\overline{\mathbf{V}}^{RU}$, $\epsilon$, $\tau_1$, $\tau_2$);
    endif;
end;
```

Operation Count

If there are k surface splits (k calls to *Subdivide_β_2_Along_u* and *Subdivide_β_2_Along_v*), then there must be $k+1$ calls to the polygon approximation routine *Polygon_Approximate_β_2*; each of these requires $35d$ additions / subtractions, $25d$ multiplications, and no divisions. Each call to the splitting routines requires $76d$ additions / subtractions, $104d$ multiplications, and no divisions. Thus, the pseudo-code above requires a total of $17+35d+111dk$ additions / subtractions, $24+25d+129dk$ multiplications, and 6 divisions.

6.4.2. Approach 2

The second approach for the approximation of Beta2-spline curves, presented in section 6.3.2, proceeded by first transforming the Beta2-spline control polygon into an equivalent Bézier control polygon, then approximating the Bézier control polygon. The second approach for Beta2-spline surfaces proceeds similarly. The Beta2-spline control graph is converted into an equivalent Bézier control graph; the Bézier control graph is then approximated to obtain a polygonal approximation to the original Beta2-spline surface patch. To accomplish this, we develop the mathematics for the transformation to the Bézier surface representation.

Let $\overline{\mathbf{V}}$ represent the 4x4 matrix of vertices defining the Beta2-spline patch to be approximated, and let $\overline{\mathbf{W}}$ denote the vertices for an equivalent Bézier surface. We require that

$$\sum_{r=-2}^{1}\sum_{s=-2}^{1}\mathbf{V}_{r+2,s+2}b_r(\beta_2;u)b_s(\beta_2;v) = \sum_{r=0}^{3}\sum_{s=0}^{3}\mathbf{W}_{r,s}B_r(u)B_s(v) \qquad \text{(VI.31)}$$

Due to the polynomial form for the basis functions, equation (VI.31) can be written in matrix form as

$$[1 \quad u \quad u^2 \quad u^3]\overline{\mathbf{C}}\overline{\mathbf{V}}\overline{\mathbf{C}}^T\begin{bmatrix}1\\v\\v^2\\v^3\end{bmatrix} = [1 \quad u \quad u^2 \quad u^3]\overline{\mathbf{D}}\overline{\mathbf{W}}\overline{\mathbf{D}}^T\begin{bmatrix}1\\v\\v^2\\v^3\end{bmatrix} \qquad \text{(VI.32)}$$

where $\overline{\mathbf{C}}$, $\overline{\mathbf{D}}$, and $\overline{\mathbf{V}}$ are defined in equations (III.10), (VI.16), (VI.21), respectively, and

$$\overline{\mathbf{W}} = \begin{bmatrix}\mathbf{W}_{0,0} & \mathbf{W}_{0,1} & \mathbf{W}_{0,2} & \mathbf{W}_{0,3}\\ \mathbf{W}_{1,0} & \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \mathbf{W}_{1,3}\\ \mathbf{W}_{2,0} & \mathbf{W}_{2,1} & \mathbf{W}_{2,2} & \mathbf{W}_{2,3}\\ \mathbf{W}_{3,0} & \mathbf{W}_{3,1} & \mathbf{W}_{3,2} & \mathbf{W}_{3,3}\end{bmatrix} \qquad \text{(VI.33)}$$

Solving (VI.32) for $\overline{\mathbf{W}}$, followed by simplification yields

$$\overline{\mathbf{W}} = (\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}})\overline{\mathbf{V}}(\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}})^T \qquad \text{(VI.34)}$$

This form is to be expected since the matrix $\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}$ was shown, in section 6.3.2, to be the matrix that transforms a Beta2-spline control polygon into an equivalent Bézier control polygon. Equation (VI.34) describes the transformation of control graphs as the result of "sandwiching" the Beta2-spline control graph between the matrix $\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}$ and its transpose. Using Vaxima, the components of $\overline{\mathbf{W}}$ can be explicitly obtained, resulting the in the following procedure for transforming a Beta2-spline control graph into an equivalent Bézier control graph.

```
procedure Map_$\beta_2$_Surface_To_Bézier( $\overline{\mathbf{V}}$, $\beta_2$, $\overline{\mathbf{W}}$)
local variable $\gamma$, $\tau_1$, $\tau_2$, $\tau_1^2$, $\tau_2^2$, $\tau_1 a 2$;
begin
    $\gamma := 1/(\beta_2 + 12)$;
    $\tau_1 := 2\gamma$;
    $\tau_2 := \gamma(\beta_2 + 8)$;
    $\tau_1^2 := \tau_1\tau_1$;
    $\tau_1\tau_2 := \tau_1 a 2$;
    $\tau_2^2 := \tau_2\tau_2$;
    $\mathbf{W}_{0,0} := \tau_1^2(\mathbf{V}_{2,2} + \mathbf{V}_{2,0} + \mathbf{V}_{0,2} + \mathbf{V}_{0,0}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,1} + \mathbf{V}_{1,2} + \mathbf{V}_{1,0} + \mathbf{V}_{0,1}) + \tau_2^2\mathbf{V}_{1,1}$;
    $\mathbf{W}_{1,0} := 2\tau_1^2(\mathbf{V}_{2,2} + \mathbf{V}_{2,0}) +$
            $\tau_1\tau_2(2\mathbf{V}_{2,1} + \mathbf{V}_{1,2} + \mathbf{V}_{1,0}) + \tau_2^2\mathbf{V}_{1,1}$;
    $\mathbf{W}_{2,0} := 2\tau_1^2(\mathbf{V}_{1,2} + \mathbf{V}_{1,0}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,2} + \mathbf{V}_{2,0} + 2\mathbf{V}_{1,1}) + \tau_2^2\mathbf{V}_{2,1}$;
    $\mathbf{W}_{3,0} := \tau_1^2(\mathbf{V}_{3,2} + \mathbf{V}_{3,0} + \mathbf{V}_{1,2} + \mathbf{V}_{1,0}) +$
            $\tau_1\tau_2(\mathbf{V}_{3,1} + \mathbf{V}_{2,2} + \mathbf{V}_{2,0} + \mathbf{V}_{1,1}) + \tau_2^2\mathbf{V}_{2,1}$;
    $\mathbf{W}_{0,1} := 2\tau_1^2(\mathbf{V}_{2,2} + \mathbf{V}_{0,2}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,1} + 2\mathbf{V}_{1,2} + \mathbf{V}_{0,1}) + \tau_2^2\mathbf{V}_{1,1}$;
    $\mathbf{W}_{1,1} := 4\tau_1^2\mathbf{V}_{2,2} + 2\tau_1\tau_2(\mathbf{V}_{2,1} + \mathbf{V}_{1,2}) + \tau_2^2\mathbf{V}_{1,1}$;
    $\mathbf{W}_{2,1} := 4\tau_1^2\mathbf{V}_{1,2} + 2\tau_1\tau_2(\mathbf{V}_{2,2} + \mathbf{V}_{1,1}) + \tau_2^2\mathbf{V}_{2,1}$;
    $\mathbf{W}_{3,1} := 2\tau_1^2(\mathbf{V}_{3,2} + \mathbf{V}_{1,2}) +$
            $\tau_1\tau_2(\mathbf{V}_{3,1} + 2\mathbf{V}_{2,2} + \mathbf{V}_{3,3}) + \tau_2^2\mathbf{V}_{2,1}$;
    $\mathbf{W}_{0,2} := 2\tau_1^2(\mathbf{V}_{2,1} + \mathbf{V}_{0,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,2} + \mathbf{V}_{0,2} + 2\mathbf{V}_{1,1}) + \tau_2^2\mathbf{V}_{1,2}$;
    $\mathbf{W}_{1,2} := 4\tau_1^2\mathbf{V}_{2,1} +$
            $2\tau_1\tau_2(\mathbf{V}_{2,2} + \mathbf{V}_{1,1}) + \tau_2^2\mathbf{V}_{1,2}$;
    $\mathbf{W}_{2,2} := 4\tau_1^2\mathbf{V}_{1,1} +$
            $2\tau_1\tau_2(\mathbf{V}_{2,1} + \mathbf{V}_{1,2}) + \tau_2^2\mathbf{V}_{2,2}$;
    $\mathbf{W}_{3,2} := 2\tau_1^2(\mathbf{V}_{3,1} + \mathbf{V}_{1,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{3,2} + \mathbf{V}_{1,3} + 2\mathbf{V}_{2,1}) + \tau_2^2\mathbf{V}_{2,2}$;
    $\mathbf{W}_{0,3} := \tau_1^2(\mathbf{V}_{2,3} + \mathbf{V}_{2,1} + \mathbf{V}_{0,3} + \mathbf{V}_{0,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,2} + \mathbf{V}_{1,3} + \mathbf{V}_{0,2}) + \tau_2^2\mathbf{V}_{1,2}$;
    $\mathbf{W}_{1,3} := 2\tau_1^2(\mathbf{V}_{2,3} + \mathbf{V}_{2,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{1,3} + \mathbf{V}_{1,1} + 2\mathbf{V}_{2,2}) + \tau_2^2\mathbf{V}_{1,2}$;
    $\mathbf{W}_{2,3} := 2\tau_1^2(\mathbf{V}_{1,3} + \mathbf{V}_{1,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{2,3} + \mathbf{V}_{2,1} + 2\mathbf{V}_{1,2}) + \tau_2^2\mathbf{V}_{2,2}$;
    $\mathbf{W}_{3,3} := \tau_1^2(\mathbf{V}_{3,3} + \mathbf{V}_{3,1} + \mathbf{V}_{1,3} + \mathbf{V}_{1,1}) +$
            $\tau_1\tau_2(\mathbf{V}_{3,2} + \mathbf{V}_{2,3} + \mathbf{V}_{2,1} + \mathbf{V}_{1,2}) + \tau_2^2\mathbf{V}_{2,2}$;
end;
```

The splitting of a Bézier surface is analogous to that of a Beta2-spline surface; a split in the u direction is accomplished by splitting each column of $\overline{\mathbf{V}}$ using the routine *Subdivide_Bézier*, a split in the v direction is done by splitting each row.

Since the Bézier surface patch is guaranteed to interpolate its four corner vertices $\mathbf{W}_{0,0}$, $\mathbf{W}_{0,3}$, $\mathbf{W}_{3,0}$, and $\mathbf{W}_{3,3}$, a good polygonal approximation is obtained by using four triangles in a scheme similar to the one used in *Polygon_Approximate_*β_2. Thus, the pseudo-code to subdivide a Beta2-spline surface patch using Approach 2 is

```
procedure Subdivide_Bézier_Along_u( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^{LU}$, $\overline{\mathbf{W}}^{RU}$)
local variable $i$;
begin
   for $i$ = 1 to 4 do
      Subdivide_Bézier( $Col_i(\overline{\mathbf{W}})$, $Col_i(\overline{\mathbf{W}}^{LU})$, $Col_i(\overline{\mathbf{W}}^{RU})$);
end;

procedure Subdivide_Bézier_Along_v( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^{LV}$, $\overline{\mathbf{W}}^{RV}$)
local variable $i$;
begin
   for $i$ = 1 to 4 do
      Subdivide_Bézier( $Row_i(\overline{\mathbf{W}})^T$, $Row_i(\overline{\mathbf{W}}^{LV})^T$, $Row_i(\overline{\mathbf{W}}^{RV})^T$);
end;

procedure Polygon_Approximate_Bézier( $\overline{\mathbf{W}}$)
local variable $\mathbf{W}_a$;
begin

   $\mathbf{W}_a = 0.25(\mathbf{W}_{0,0} + \mathbf{W}_{3,0} + \mathbf{W}_{0,3} + \mathbf{W}_{3,3})$;

   Output_Triangle( $\mathbf{W}_{0,0}$, $\mathbf{W}_{3,0}$, $\mathbf{W}_a$);
   Output_Triangle( $\mathbf{W}_{3,0}$, $\mathbf{W}_{3,3}$, $\mathbf{W}_a$);
   Output_Triangle( $\mathbf{W}_{3,3}$, $\mathbf{W}_{0,3}$, $\mathbf{W}_a$);
   Output_Triangle( $\mathbf{W}_{0,3}$, $\mathbf{W}_{0,0}$, $\mathbf{W}_a$);
end;

procedure Approximate_Bézier_Surface( $\overline{\mathbf{W}}$, $\epsilon$)
local variable $\overline{\mathbf{W}}^{LU}$, $\overline{\mathbf{W}}^{RU}$, $\overline{\mathbf{W}}^{LV}$, $\overline{\mathbf{W}}^{RV}$;
begin
   if $\overline{\mathbf{W}}$ is flat to within $\epsilon$ along $u$ then
      if $\overline{\mathbf{W}}$ is flat to within $\epsilon$ along $v$ then
         Polygon_Approximate_Bézier( $\overline{\mathbf{W}}$);
      else
         Subdivide_Bézier_Along_v( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^{LV}$, $\overline{\mathbf{W}}^{RV}$);
         Approximate_Bézier_Surface( $\overline{\mathbf{W}}^{LV}$, $\epsilon$);
         Approximate_Bézier_Surface( $\overline{\mathbf{W}}^{RV}$, $\epsilon$);
      endif
   else
      Subdivide_Bézier_Along_u( $\overline{\mathbf{W}}$, $\overline{\mathbf{W}}^{LU}$, $\overline{\mathbf{W}}^{RU}$);
      Approximate_Bézier_Surface( $\overline{\mathbf{W}}^{LU}$, $\epsilon$);
      Approximate_Bézier_Surface( $\overline{\mathbf{W}}^{RU}$, $\epsilon$);
   endif
end;

procedure Approximate_$\beta_2$_Surface_2( $\overline{\mathbf{V}}$, $\beta_2$, $\epsilon$)
local variable $\overline{\mathbf{W}}$;
begin
   Map_$\beta_2$_Surface_To_Bézier( $\overline{\mathbf{V}}$, $\beta_2$, $\overline{\mathbf{W}}$);
   Approximate_Bézier_Surface( $\overline{\mathbf{W}}$, $\epsilon$);
end;
```

Operation Count

The mapping of a Beta2-spline surface patch to an equivalent Bézier surface patch using the routine *Map_β_2_Surface_To_Bézier* requires $2+82d$ additions / subtractions, $8+56d$ multiplications, and 1 division; k calls to the Bézier subdivision routines requires $36dk$ additions / subtractions, $32dk$ multiplications, and no divisions; $k+1$ calls to *Polygon_Approximate_Bézier* requires $(k+1)3d$ additions / subtractions, $k+1$ multiplications, and no divisions. Thus, Approach 2 requires a total of $2+85d+39dk$ additions / subtractions, $8+57d+33dk$ multiplications, and 1 division to subdivide a Beta2-spline surface patch k times.

7. Conclusion

This paper has presented a special case of the Beta-spline curve and surface technique known as the *Beta2-spline* technique. A Beta2-spline curve or surface is parametrized in terms of the single tension parameter β_2 instead of the two shape parameters β_1 and β_2 possessed by a standard Beta-spline. Experience has shown that this is a simple, computationally efficient, but very useful special case of the Beta-spline technique.

When $\beta_2=0$, the Beta2-spline representation reduces to that of a uniform cubic B-spline. As β_2 is increased from zero the curve (or surface) uniformly approaches the control polygon (or control graph) that defines it. In the limit of infinite tension, a Beta2-spline curve becomes a piecewise linear spline that interpolates the vertices of the control polygon; a Beta2-spline surface becomes a piecewise planar spline, where the borders between the adjacent surface patches interpolate the edges of the control graph. The Beta2-spline design process generally proceeds by first laying down vertices to "rough-out" the curve or surface. The designer then has the freedom to add new vertices, remove or move existing vertices, or change the value of β_2 until a curve or surface of the desired properties is obtained.

To aid the implementor of the Beta2-spline technique, detailed algorithms for the evaluation and subdivision of Beta2-spline curves and surfaces were presented. Two different subdivision algorithms, both for curves and surfaces, were developed. The first, called Approach 1, approximates a Beta2-spline curve or surface by direct subdivision of the Beta2-spline representation. The second, called Approach 2, approximates a Beta2-spline by transforming it into an equivalent Bézier curve or surface, then approximates the Bézier spline using recursive subdivision. The operation counts tallied for the various algorithms show that Approach 1 is roughly four times more expensive than Approach 2; this holds for both the curve and surface algorithms. A Beta2-spline object converted into triangles via approach 2 and rendered on a high resolution color monitor is shown in figure 9.

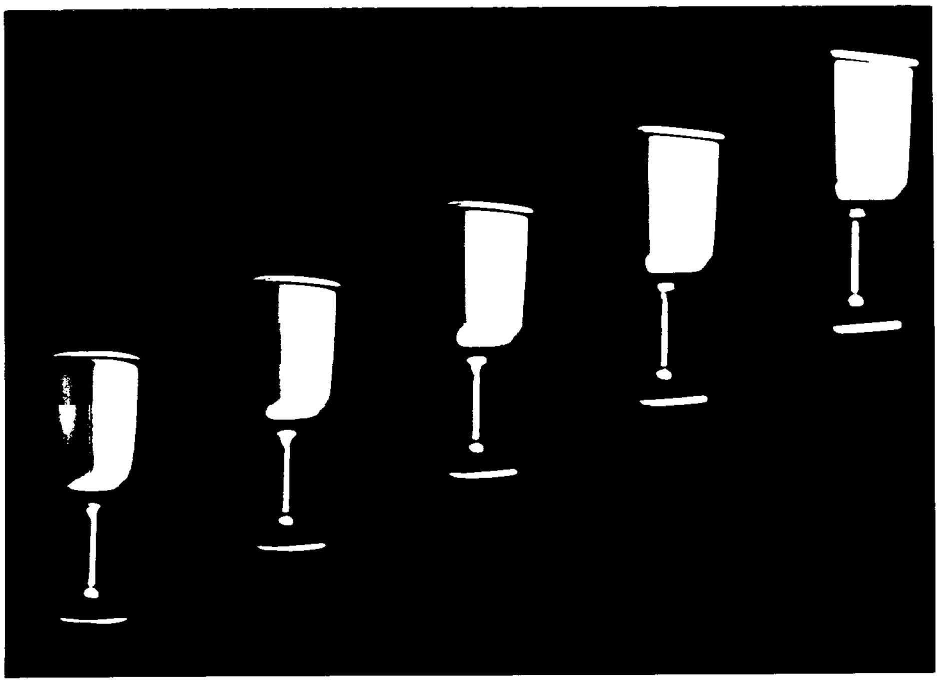

Figure 9.
The pewter goblets above are all defined by the same control graph. Their shape has been modified by changing only the value of the tension parameter β_2. The values of β_2 from left to right are: 0, 5, 10, 20, *and* 50.

8. Acknowledgements

The authors would like to thank Mark Dippé of the Berkeley Computer Graphics Laboratory whose rendering algorithms generated the shaded image. Thanks are also in order to Richard Fateman and his symbolic computation group at U.C. Berkeley for providing and supporting Vaxima, without which much of this work would have been impossible.

References

1. Brian A. Barsky, "The Beta-spline: A Curve and Surface Representation for Computer Graphics and Computer Aided Geometric Design." Submitted for publication.
2. Brian A. Barsky, "Arbitrary Subdivision of Bezier Curves." In preparation.
3. Brian A. Barsky, *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).
4. Brian A. Barsky and John C. Beatty, "Local Control of Bias and Tension in Beta-splines," pp. 193-218 in *Proceedings of SIGGRAPH '83 (Vol. 17, No. 3),* ACM,(25-29 July, 1983). Selected for publication in *ACM Transactions on Graphics,* Vol. 2, No. 2, April 1983, pp. 109-134.
5. Brian A. Barsky, Tony D. DeRose, and Mark D. Dippe, "An Adaptive Subdivision Method With Crack Prevention for Rendering Beta-Spline Objects." In preparation.
6. Edwin E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces,* Ph.D. Thesis, University of Utah, Salt Lake City, Utah (December, 1974). Also Tech. Report No. UTEC-CSc-74-133, Department of Computer Science, University of Utah.
7. Edwin E. Catmull and James H. Clark, "Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes," *Computer-Aided Design* **10**(6) pp. 350-355 (November, 1978).
8. George M. Chaikin, "An Algorithm for High-Speed Curve Generation," *Computer Graphics and Image Processing* **3** pp. 346-349 (1974).
9. Tony D. DeRose, "Arbitrary Subdivision of Blended Splines." In preparation.
10. D. W. H. Doo and M. A. Sabin, "Behaviour of Recursive Division Surfaces Near Extraordinary Points," *Computer-Aided Design* **10**(6) pp. 356-360 (November, 1978).
11. Richard J. Fateman, *Addendum to the MACSYMA Reference Manual for the VAX,* Technical Report, University of California, Berkeley (1982).
12. Jeffrey M. Lane and Loren C. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing* **11**(3) pp. 290-297 (November, 1979).
13. Jeffrey M. Lane and Richard F. Riesenfeld, "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-2**(1) pp. 35-46 (January, 1980).
14. Robert W. Nydegger, *A Data Minimization Algorithm of Analytical Models for Computer Graphics,* Master's Thesis, University of Utah, Salt Lake City, Utah (1972).