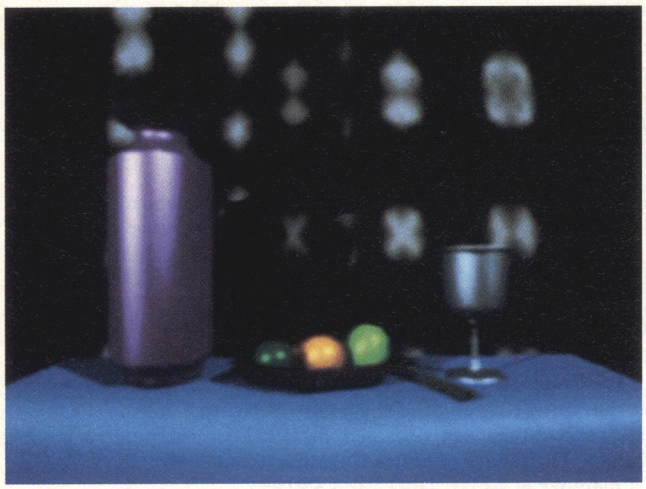
Simpler equations and computationally more efficient algorithms make the Beta2-spline technique easier to understand and useful to the designer.

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

Brian A. Barsky and Tony D. DeRose
University of California, Berkeley

This article 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 the rendering of Beta2-spline curves and surfaces via subdivision are presented.

This technique is proving to be quite useful in the modeling of complex shapes. The representation is sufficiently general and flexible so as to be capable of modeling irregular curved-surface objects such as automobile bodies, aircraft fuselages, ship hulls, turbine blades, and bottles.



This image is composed entirely of Beta2-spline surfaces, showing the wide range of shapes that can be modeled with the Beta2-spline technique. β_2 values for the various objects range from 0 to 30.

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

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

0272-1716/85/0900/0046$01.00© 1985 IEEE

$$\mathbf{Q}_i(u) = (X_i(u), Y_i(u)) \quad (1)$$

We adhere to the 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).

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

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

The functions $B_0(u), B_1(u), \ldots, B_k(u)$ are called the *blending* or *basis* functions. The sequence of vertices $<\mathbf{V}_i, \mathbf{V}_{i+1}, \ldots, \mathbf{V}_{i+k}>$ 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.

Surfaces can also be constructed using a blended technique to form what is known as a *tensor product surface.* A single tensor product *surface patch* is described by two variable parameters u and v. In particular, the i, j^{th} such patch is defined as

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

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, we 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).

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

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

Similarly, the second derivative vector written in component form is

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

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

For *positional* continuity of the spline curve, the ending point of the i^{th} segment must match the starting point of the $i+1^{\text{st}}$ segment. This requirement is not sufficient to make the joint appear to be *smooth*, however. One could impose continuity of the first and second derivative vectors, but this is overly restrictive. The requirement of continuity of *unit tangent* and *curvature vectors* is sufficient for the smoothness of the spline curve.[1-5] It is not immediately obvious that continuity of the unit tangent vector and curvature vector is less restrictive than continuity

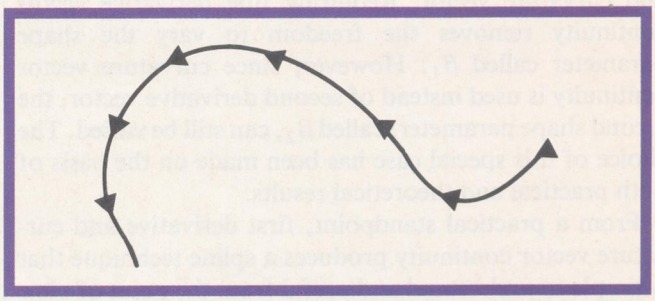


Figure 1. A section of a vector-valued spline curve. Triangles show the locations of the joints.

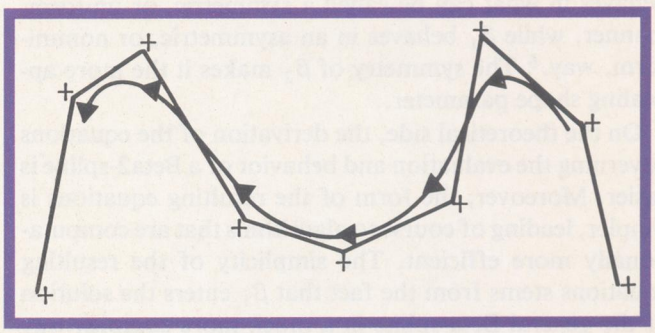


Figure 2. A control polygon and its B-spline curve. The control vertices are highlighted by plus signs; the joints are denoted by triangles.

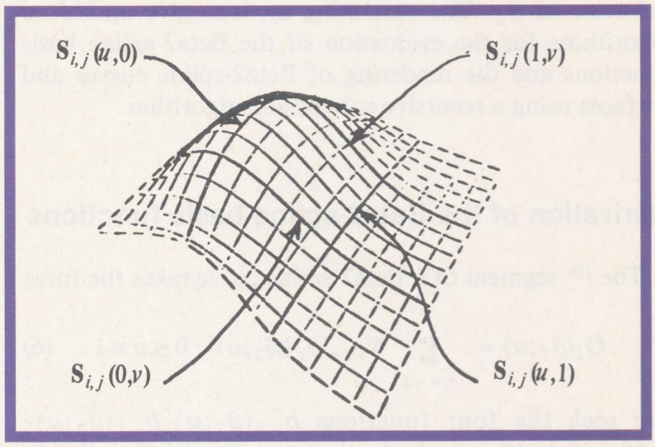


Figure 3. A spline surface in the region of the patch $\mathbf{S}_{i,j}(u,v)$.

of first and second derivative vectors, but this can be seen by considering the definitions of these quantities.[1-5]

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, can still be varied. The choice of this special case has been made on the basis of 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 we will show later, β_2 behaves in what can be called a symmetric, or uniform, manner, while β_1 behaves in an asymmetric, or nonuniform, way.[6] The symmetry of β_2 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 the default value of 1 allows extensive algebraic simplification.

The next section gives the derivation of a Beta2-spline. The section following that examines the qualitative behavior of β_2. The remaining sections give optimized algorithms for the evaluation of the Beta2-spline basis functions and the rendering of Beta2-spline curves and surfaces using a recursive subdivision algorithm.

Derivation of the Beta2-spline basis functions

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

$$\mathbf{Q}_i(\beta_2;u) = \sum_{r=-2}^{1} \mathbf{V}_{i+r}\, b_r(\beta_2;u)\ ;\ 0 \le u \le 1 \qquad (6)$$

We seek the four functions $b_{-2}(\beta_2;u), b_{-1}(\beta_2;u), b_0(\beta_2;u), b_1(\beta_2;u)$. Note that a segment of a Beta2-spline curve is defined by only four control vertices. Thus, modification of a single control vertex affects only four curve segments, a property known as *local control*. If we assume a cubic polynomial form for each of the basis functions, then each can be expressed as

$$b_r(\beta_2;u) = \sum_{g=0}^{3} c_{rg}(\beta_2) u^g \qquad (7)$$

The 16 unknowns $c_{rg}(\beta_2)$, $-2 \le r \le 1$, $0 \le g \le 3$ 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

$$\mathbf{Q}_{i+1}(\beta_2;0) = \mathbf{Q}_i(\beta_2;1) \qquad (8)$$

For continuity of the first derivative vector, we require that

$$\mathbf{Q}_{i+1}^{(1)}(\beta_2;0) = \mathbf{Q}_i^{(1)}(\beta_2;1) \qquad (9)$$

It can also be shown[1-5] that if the first derivative vector is to be continuous, then the curvature vector is continuous at this joint if

$$\mathbf{Q}_{i+1}^{(2)}(\beta_2;0) = \mathbf{Q}_i^{(2)}(\beta_2;1) + \beta_2\, \mathbf{Q}_i^{(1)}(\beta_2;1) \qquad (10)$$

If $\beta_2 = 0$, the constraint equation (number 10) 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 (number 8) leads to

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

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

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

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

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

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

$$\begin{aligned} 0 &= b_{-2}^{(2)}(\beta_2;1) \\ b_{-2}^{(2)}(\beta_2;0) &= b_{-1}^{(2)}(\beta_2;1) + \beta_2 b_{-1}^{(1)}(\beta_2;1) \\ b_{-1}^{(2)}(\beta_2;0) &= b_0^{(2)}(\beta_2;1) + \beta_2 b_0^{(1)}(\beta_2;1) \\ b_0^{(2)}(\beta_2;0) &= b_1^{(2)}(\beta_2;1) + \beta_2 b_1^{(1)}(\beta_2;1) \\ b_1^{(2)}(\beta_2;0) &= 0 \end{aligned}$$

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 underdetermined system of equations which can be made complete by introducing another, linearly independent, constraint. The constraint we choose is a normalization that is necessary (but not sufficient) for the curve segment to stay 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.[7] The resulting solution can be written as a matrix $\bar{\mathbf{C}}$ of the coefficients of the basis functions,*

$$\bar{\mathbf{C}}(\beta_2)=\begin{bmatrix} c_{-2,0} & c_{-1,0} & c_{0,0} & c_{1,0} \\ c_{-2,1} & c_{-1,1} & c_{0,1} & c_{1,1} \\ c_{-2,2} & c_{-1,2} & c_{0,2} & c_{1,2} \\ c_{-2,3} & c_{-1,3} & c_{0,3} & c_{1,3} \end{bmatrix}$$

$$=\gamma\begin{bmatrix} 2 & \beta_2+8 & 2 & 0 \\ -6 & 0 & 6 & 0 \\ 6 & -3(\beta_2+4) & 3(\beta_2+2) & 0 \\ -2 & 2(\beta_2+3) & -2(\beta_2+3) & 2 \end{bmatrix} \quad (11)$$

or as the basis functions themselves

$$\begin{aligned} b_{-2}(\beta_2;u)&=2\gamma(1-u)^3 \\ b_{-1}(\beta_2;u)&=\gamma(\beta_2+8+u^2(-3(\beta_2+4)+2u(\beta_2+3))) \\ b_0(\beta_2;u)&=\gamma(2+u(6+u(3(\beta_2+2)-2u(\beta_2+3)))) \\ b_1(\beta_2;u)&=2\gamma u^3 \end{aligned} \quad (12)$$

where

$$\gamma=\frac{1}{\beta_2+12} \quad (13)$$

The basis functions of Equation 12 are symmetric in the sense that $b_{-2}(\beta_2;u)=b_1(\beta_2;1-u)$ and $b_{-1}(\beta_2;u)=b_0(\beta_2;1-u)$. This symmetry implies that reversing the order of control vertices in a control polygon reverses the resulting curve. This is not the case with the general Beta-spline when $\beta_1\neq 1$.

Behavior of β_2

In the limit of infinite β_2, Barsky and Beatty[6] show that the Beta-spline curve (and hence the Beta2-spline curve) becomes a piecewise linear function that interpolates the interior vertices of the control polygon. This behavior of β_2 suggests that it acts like a tension parameter, and thus β_2 is called *tension*. Increasing the tension of a Beta2-spline corresponds to increasing the value of β_2 (see Figure 4). The ν-spline[8] is another example of a spline representation with tension. However, the ν-spline differs from this work in that it *interpolates* (passes through) the control vertices but does not have the property of local control.

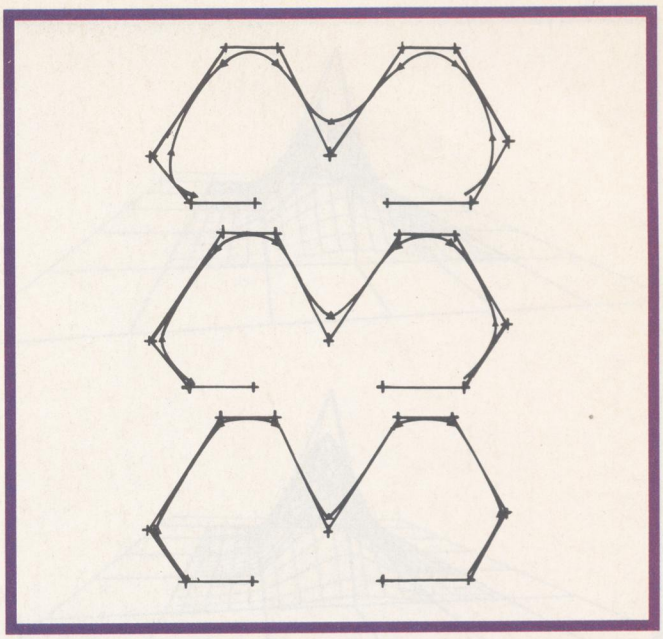


Figure 4. The curves above differ only in the value of β_2. The value of β_2 is 0 for the top curve, 5 for the middle curve, and 20 for the bottom curve. Plus signs denote control vertices; triangles denote joints. Note that as β_2 increases toward infinity, the curve uniformly approaches the control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

Increasing the tension applied to a Beta2-spline surface makes the surface more polygonal. In the limit of infinite tension, the surface can be shown to coincide with the interior panels of the control graph that defines it. This behavior is shown in Figure 5.

Barsky and Beatty[6] also show that a curve segment or surface patch is guaranteed to be within the convex hull of the vertices that define it for nonnegative 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[6,9,10] 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 article.

*A matrix is denoted by a boldface character with a diacritical bar.

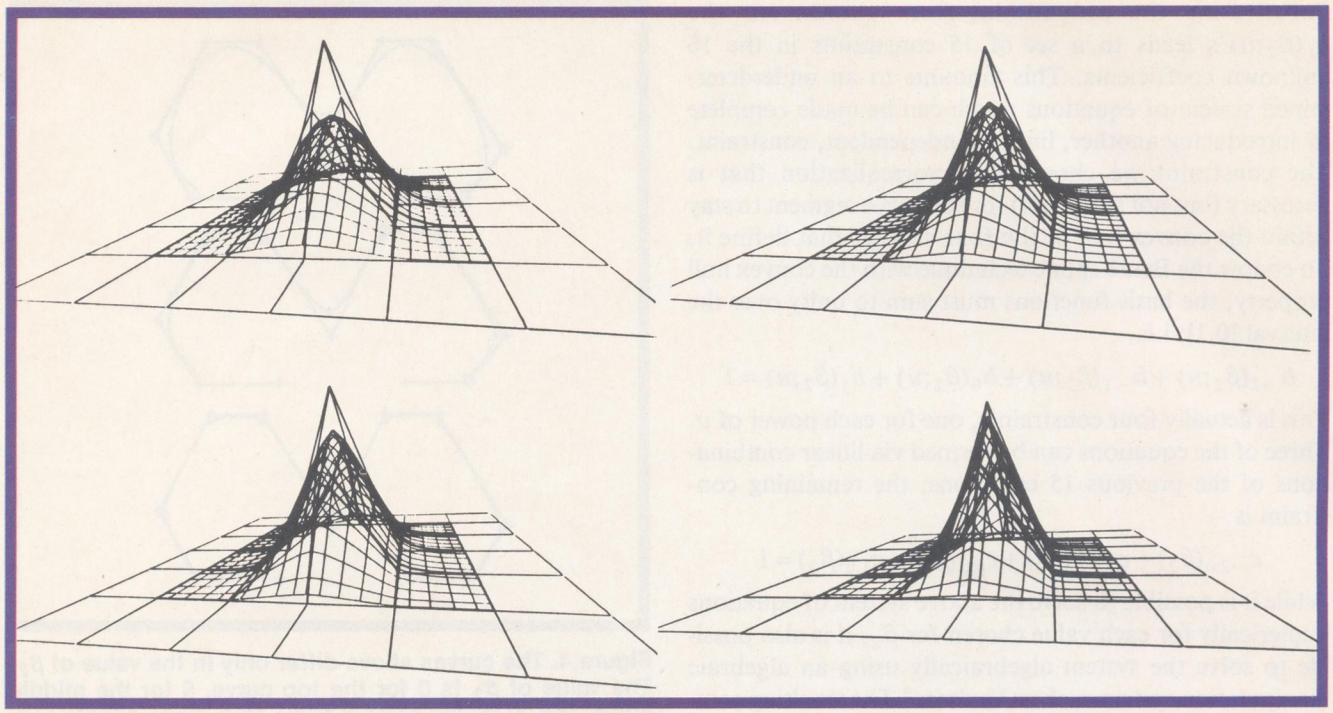


Figure 5. This sequence of Beta2-spline surfaces and control graphs shows the effect of increasing the value of β_2. The values of β_2 are 0 for the top-left surface, 5 for the top-right surface, 10 for the bottom-left surface, and 50 for the surface in the bottom-right position. As β_2 increases, the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

Evaluation of the basis functions

If a Beta2-spline curve segment is to be drawn by repeated evaluation of Equation 6, say at the domain values $(u_0, u_1, \ldots, u_p)$, 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. Furthermore, only three functions need to be evaluated from their coefficients; the fourth is obtained by using the fact that the basis functions sum to unity. The following algorithm can therefore be used to tabulate the values of the basis functions given a value of β_2:

```
compute_c(β2,c);
for all u in {u0,u1, . . . , up} do
   compute_b(c,u,b);
```

where

```
procedure compute_c(β2,c)
local variable γ;
begin
   γ := 1/(β2 + 12);
   c_{-2,0} := 2*γ;
   c_{-1,0} := γ*(β2 + 8);
   c_{-1,2} := -3*γ*(β2 + 4);
   c_{-1,3} := c_{-2,0}*(β2 + 3);
   c_{1,3} := c_{-2,0};
end;

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

The above algorithm requires $4+6(p+1)$ additions/subtractions, $5+8(p+1)$ multiplications, and one division. When p is small, this cost is roughly 50 percent of the cost of the algorithms given by Barsky[1-3] for general Beta-spline evaluation.

Subdivision

General subdivision schema for curves. We have described an algorithm for the evaluation of a Beta2-spline curve based on the blended definition given in Equation 6. 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

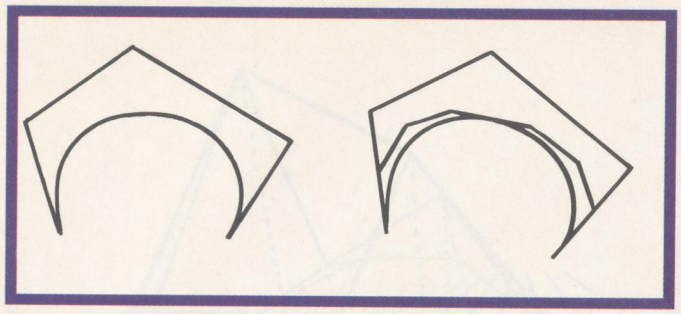


Figure 6. The figure on the left shows a curve and its control polygon. The figure on the right depicts the Bézier curve, its control polygon, and the left and right subpolygons corresponding to a midpoint subdivision.

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.[11-16] For instance, Catmull[17] introduced an algorithm that subdivides a surface that is to be approximated. However, Catmull's technique subdivided the surface until the pieces were of a size on the order of a pixel. Lane and Carpenter[14] 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 and are called the *left subpolygon* and *right subpolygon* (see Figure 6). Although it is common to split the curve at the *parametric midpoint* (the point where the domain parameter $u = \frac{1}{2}$), it is also possible to perform non-midpoint subdivision.[18-20] Midpoint subdivision is sufficient for our purposes, but it does not, in general, yield two pieces of equal arc length.

Goldman[21] has recently shown that if the basis functions are linearly independent and sum to one, then recursive application of this subdivision procedure must result in successively flatter subpolygons. The convex hull property then guarantees that a relatively flat control polygon must define a relatively flat curve. Thus, the recursion stops when the subpolygon is deemed to be flat to within some tolerance ϵ. The curve generated by the polygon can then be approximated by a single linear segment. The union of these linear approximants yields a uniformly accurate representation of the original curve segment. If $\bar{\mathbf{V}}$ denotes the original control polygon, $\bar{\mathbf{V}}^L$ represents the left subpolygon, and $\bar{\mathbf{V}}^R$ the right subpolygon, then the schema can be stated more precisely by the following recursive procedure:

```
procedure Approximate_Curve(V̄, ε)
local variable V̄^L, V̄^R;
begin
  if V̄ is flat to within ε then
      Linearly_Approximate(V̄);
  else
      Subdivide(V̄, V̄^L, V̄^R);
      Approximate_Curve(V̄^L, ε);
      Approximate_Curve(V̄^R, ε);
  endif;
end;
```

The procedure *Subdivide* splits its first argument into left and right subpolygons, 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. One way to test a control polygon for flatness is to check 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 criterion ϵ, then the control polygon passes the flatness test and is approximated by a linear segment. If the spline technique being used interpolates 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.

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. It is generally hard to determine if a surface is flat, but the

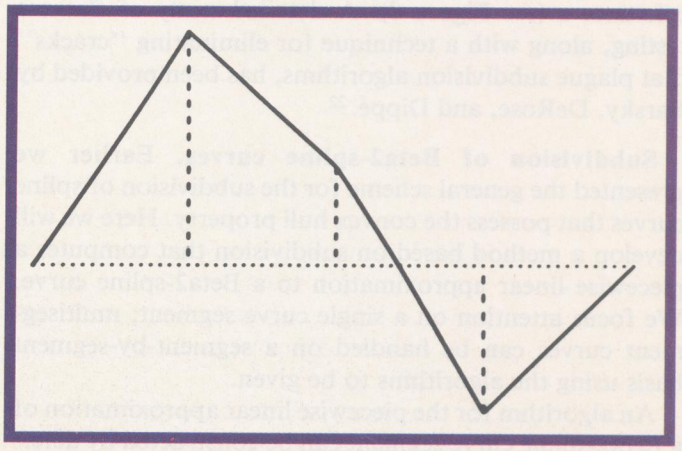


Figure 7. Determination of a flat control polygon. It is deemed flat if the maximum perpendicular distance (dashed line) between each interior vertex and the line segment connecting the endpoints of the control polygon (dotted line) is less than the tolerance ϵ.

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

```
procedure Approximate_Surface(V̄,ϵ)
local variables V̄^LU,V̄^RU,V̄^LV,V̄^RV;
begin
  if V̄ is flat within ϵ along u then
    if V̄ is flat within ϵ along v then
      Polygon_Approximate(V̄);
    else
      Subdivide_Along_v(V̄,V̄^LV,V̄^RV);
      Approximate_Surface(V̄^LV,ϵ);
      Approximate_Surface(V̄^RV,ϵ);
    endif
  else
    Subdivide_Along_u(V̄,V̄^LU,V̄^RU);
    Approximate_Surface(V̄^LU,ϵ);
    Approximate_Surface(V̄^RU,ϵ);
  endif;
end;
```

One way to test the flatness of a control graph is to compute 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). A detailed study of flatness testing, along with a technique for eliminating "cracks" that plague subdivision algorithms, has been provided by Barsky, DeRose, and Dippé.[22]

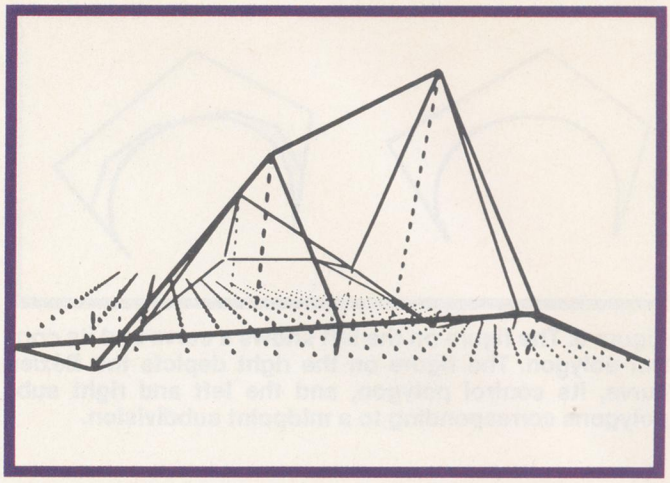


Figure 8. Flatness testing for a control graph. Three arbitrarily chosen corner vertices are used to define a plane (dotted area). The perpendicular distances from the other vertices in the graph to this plane are computed (dashed lines). If the largest such distance is less than the tolerance ϵ, the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

Subdivision of Beta2-spline curves. Earlier we presented the general scheme for the subdivision of spline curves that possess the convex hull property. Here we will develop a method based on subdivision that computes a piecewise linear approximation to a Beta2-spline curve. We focus attention on a single curve segment; multisegment curves can be handled on a segment-by-segment basis using the algorithms to be given.

An algorithm for the piecewise linear approximation of a Beta2-spline curve segment can be constructed by determining the procedure *Subdivide* previously introduced. This procedure consists of three basic steps:

1. Convert the Beta2-spline control polygon into a Bézier control polygon that defines the same curve.
2. Subdivide the Bézier control polygon using the midpoint subdivision algorithm of Lane and Riesenfeld.[15]
3. Convert the resulting Bézier subpolygons back into equivalent Beta2-spline subpolygons.

However, a computationally more efficient approach is to convert the Beta2-spline control polygon into an equivalent Bézier control polygon as in step 1 above, then linearly approximate the resulting Bézier curve, eliminating the inverse conversion of step 3. Pseudocode for this process is

```
procedure Approximate_Curve_β2(V̄, β2, ϵ)
local variable W̄;
begin
  Map_β2_Curve_To_Bézier(V̄, β2, W̄);
  Approximate_Bézier_Curve(W̄,ϵ);
end;
```

The procedure *Map_β_2_Curve_To_Bézier* converts its first argument, assumed to represent a Beta2-spline control polygon, into an equivalent Bézier control polygon, returned as the third argument. If $(\mathbf{V}_0,\mathbf{V}_1,\mathbf{V}_2,\mathbf{V}_3)$ are the vertices of the Beta2-spline control polygon, and $(\mathbf{W}_0,\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3)$ are the vertices of the Bézier control polygon, then we require that

$$\sum_{r=-2}^{1} \mathbf{V}_{r+2} b_r(\beta_2;u) = \sum_{r=0}^{3} \mathbf{W}_r B_r(u) \qquad (14)$$

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

$$B_i(u) = \begin{bmatrix} 3 \\ i \end{bmatrix} u^i (1-u)^{3-i} \quad i = 0,1,2,3$$

Equation 14 can alternatively be written in matrix form as

$$[b_{-2}(\beta_2;u)\ b_{-1}(\beta_2;u)\ b_0(\beta_2;u)\ b_1(\beta_2;u)] \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = \tag{15}$$

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

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

$$[b_{-2}(\beta_2;u)\ b_{-1}(\beta_2;u)\ b_0(\beta_2;u)\ b_1(\beta_2;u)] = [1\ u\ u^2\ u^3]\bar{\mathbf{C}} \tag{16}$$

and

$$[B_0(u)\ B_1(u)\ B_2(u)\ B_3(u)] = [1\ u\ u^2\ u^3]\bar{\mathbf{D}} \tag{17}$$

where $\bar{\mathbf{C}}$ is defined in Equation 11 and the matrix $\bar{\mathbf{D}}$ is the cubic Bézier coefficient matrix; specifically

$$\bar{\mathbf{D}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \tag{18}$$

Substituting Equations 16 and 17 into Equation 15 yields

$$[1\ u\ u^2\ u^3]\bar{\mathbf{C}} \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = [1\ u\ u^2\ u^3]\bar{\mathbf{D}} \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix} \tag{19}$$

Since the powers of u are linearly independent, Equation 19 can hold if and only if

$$\bar{\mathbf{W}} = \bar{\mathbf{D}}^{-1}\bar{\mathbf{C}}\bar{\mathbf{V}} \tag{20}$$

where

$$\bar{\mathbf{W}} = \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \\ \mathbf{W}_3 \end{bmatrix} \quad \bar{\mathbf{V}} = \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix}$$

Thus, Equation 20 defines the mapping between a Beta2-spline control polygon and a cubic Bézier control polygon. The matrix that does this mapping $\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}}$ can be written in component form (using Vaxima) as

$$\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}} = \begin{bmatrix} \tau_1 & \tau_2 & \tau_1 & 0 \\ 0 & \tau_2 & 2\tau_1 & 0 \\ 0 & 2\tau_1 & \tau_2 & 0 \\ 0 & \tau_1 & \tau_2 & \tau_1 \end{bmatrix} \tag{21}$$

where

$$\tau_1 = 2\gamma$$
$$\tau_2 = \gamma(\beta_2 + 8)$$

and γ is as defined in Equation 13. The procedure that accomplishes the mapping is

```
procedure Map_β2_Curve_To_Bézier(V̄,β2,W̄)
local variable τ1,τ2,τ3,T;
begin
  τ1 := 1/(β2 + 12);
  τ3 := 2*τ1;
  τ2 := 1 − τ3;
  T := τ3*(V2 − V0);
  W0 := τ1*(V0 + V2) + τ2*V1;
  W1 := V1 + T;
  W2 := V2 − T;
  W3 := τ1*(V1 + V3) + τ2*V2;
end;
```

This procedure requires $2+7d$ additions/subtractions, $1+5d$ multiplications, and one division for d dimensional vertices.

The Lane and Riesenfeld[15] algorithm for splitting a cubic Bézier control polygon at the parametric midpoint into left and right subpolygons can be written in matrix form as

$$\bar{\mathbf{W}}^L = \bar{\mathbf{L}}\bar{\mathbf{W}}$$

and

$$\bar{\mathbf{W}}^R = \bar{\mathbf{R}}\bar{\mathbf{W}}$$

where $\bar{\mathbf{L}}$ and $\bar{\mathbf{R}}$ are the *cubic Bézier midpoint subdivision matrices;* they are given by

$$\bar{\mathbf{L}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{bmatrix} \tag{22}$$

and

$$\bar{\mathbf{R}} = \begin{bmatrix} \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

The procedure *Subdivide_Bézier* can therefore be written as

```
procedure Subdivide_Bézier(W̄,W̄^L,W̄^R)
begin
  W0^L := W0;
  W1^L := 0.5*(W0 + W1);
  W2^L := 0.25*(W0 + W2) + 0.5*W1;
  W3^L := 0.125*(W0 + W3) + 0.375*(W1 + W2);

  W0^R := W3^L;
  W1^R := 0.25*(W1 + W3) + 0.5*W2;
  W2^R := 0.5*(W2 + W3);
  W3^R := W3;
end;
```

This procedure requires $9d$ additions/subtractions, $8d$ multiplications, and no divisions for vertices of spatial dimension 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 simply outputs a line segment connecting the first and last vertices of the control polygon. The procedure *Approximate_Bézier_Curve* referred to previously is therefore given by

```
procedure Approximate_Bézier_Curve(W̄, ε)
local variable W̄^L, W̄^R;
begin
  if W̄ is flat to within ε then
      Output_Segment(W_0, W_3);
  else
      Subdivide_Bézier(W̄, W̄^L, W̄^R);
      Approximate_Bézier_Curve(W̄^L, ε);
      Approximate_Bézier_Curve(W̄^R, ε);
  endif
end;
```

Operation count: Only one call to *Map_β_2_Curve_To_Bézier* is needed at a cost of $2+7d$ additions/subtractions, $1+5d$ multiplications, and one division. k calls to *Subdivide_Bézier* will cost $9dk$ additions/subtractions, $8dk$ multiplications, and no divisions. Thus, the total cost to approximate a Beta2-spline curve with k subdivision steps is $2+7d+9dk$ additions/subtractions, $1+5d+8dk$ multiplications, and one division.

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 next deal with a single surface patch. While it is possible to subdivide a Beta2-spline surface patch by performing the three steps listed earlier, a computationally more efficient approach is to convert the Beta2-spline control graph 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. This is completely analogous to the approach just taken for curves.

We now develop the mathematics for the transformation of a Beta2-spline control graph into an equivalent Bézier control graph. Let $\bar{\mathbf{V}}$ represent the 4×4 matrix of vertices defining the Beta2-spline patch to be approximated,

$$\bar{\mathbf{V}}=\begin{bmatrix}\mathbf{V}_{0,0} & \mathbf{V}_{0,1} & \mathbf{V}_{0,2} & \mathbf{V}_{0,3}\\ \mathbf{V}_{1,0} & \mathbf{V}_{1,1} & \mathbf{V}_{1,2} & \mathbf{V}_{1,3}\\ \mathbf{V}_{2,0} & \mathbf{V}_{2,1} & \mathbf{V}_{2,2} & \mathbf{V}_{2,3}\\ \mathbf{V}_{3,0} & \mathbf{V}_{3,1} & \mathbf{V}_{3,2} & \mathbf{V}_{3,3}\end{bmatrix} \tag{24}$$

and let $\bar{\mathbf{W}}$ denote the vertices for an equivalent Bézier surface

$$\bar{\mathbf{W}}=\begin{bmatrix}\mathbf{W}_{0,0} & \mathbf{W}_{0,1} & \mathbf{W}_{0,2} & \mathbf{W}_{0,3}\\ \mathbf{W}_{1,0} & \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \mathbf{W}_{1,3}\\ \mathbf{W}_{2,0} & \mathbf{W}_{2,1} & \mathbf{W}_{2,2} & \mathbf{W}_{2,3}\\ \mathbf{W}_{3,0} & \mathbf{W}_{3,1} & \mathbf{W}_{3,2} & \mathbf{W}_{3,3}\end{bmatrix} \tag{25}$$

We require that

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

$$=\sum_{r=0}^{3}\sum_{s=0}^{3}\mathbf{W}_{r,s}B_r(u)B_s(v)$$

Because of the polynomial nature of the basis functions, Equation 26 can be written in matrix form as

$$[1\ u\ u^2\ u^3]\bar{\mathbf{C}}\bar{\mathbf{V}}\bar{\mathbf{C}}^T\begin{bmatrix}1\\ v\\ v^2\\ v^3\end{bmatrix} \tag{27}$$

$$=[1\ u\ u^2\ u^3]\bar{\mathbf{D}}\bar{\mathbf{W}}\bar{\mathbf{D}}^T\begin{bmatrix}1\\ v\\ v^2\\ v^3\end{bmatrix}$$

where $\bar{\mathbf{C}}$, $\bar{\mathbf{D}}$, and $\bar{\mathbf{V}}$ are defined in Equations 11, 18, and 24, respectively. Solving Equation 27 for $\bar{\mathbf{W}}$, followed by simplification, yields

$$\bar{\mathbf{W}}=(\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}})\bar{\mathbf{V}}(\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}})^T \tag{28}$$

This form is to be expected, since the matrix $\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}}$ was shown earlier to be the matrix that transforms a Beta2-spline control polygon into an equivalent Bézier control polygon. Equation 28 describes the transformation of control graphs as the result of "sandwiching" the Beta2-spline control graph between the matrix $\bar{\mathbf{D}}^{-1}\bar{\mathbf{C}}$ and its transpose. Using Vaxima, the components of $\bar{\mathbf{W}}$ can be explicitly obtained, resulting in the following procedure for transforming a Beta2-spline control graph into an equivalent Bézier control graph:

```
procedure Map_β2_Surface_To_Bézier(V̄, β2, W̄)
local variable τ1, τ2, τ3, τ4, τ5, τ6, τ7, τ8;
local variable T11, T21, T12, T22, T2112, T2211;
begin
  τ1 := 2*γ;
  τ2 := γ*(β2 + 8);
  τ3 := τ1*τ2;
  τ4 := τ1*τ1;
  τ5 := τ2*τ2;
  τ6 := 2*τ3;
  τ7 := 2*τ4;
  τ8 := 4*τ4;
  T11 := τ5*V1,1;
  T21 := τ5*V2,1;
  T12 := τ5*V1,2;
  T22 := τ5*V2,2;
  T2112 := τ6*(V2,1 + V1,2);
  T2211 := τ6*(V2,2 + V1,1);
  W0,0 := τ4*(V2,2 + V2,0 + V0,2 + V0,0)
    + τ3*(V2,1 + V1,2 + V1,0 + V0,1) + T11;
  W1,0 := τ7*(V2,2 + V2,0)
    + τ3*(2*V2,1 + V1,2 + V1,0) + T11;
  W2,0 := τ7*(V1,2 + V1,0)
    + τ3*(V2,2 + V2,0 + 2*V1,1) + T21;
  W3,0 := τ4*(V3,2 + V3,0 + V1,2 + V1,0)
    + τ3*(V3,1 + V2,2 + V2,0 + V1,1) + T21;
```

$$\mathbf{W}_{0,1} := \tau_7*(\mathbf{V}_{2,2} + \mathbf{V}_{0,2}) + \tau_3*(\mathbf{V}_{2,1} + 2*\mathbf{V}_{1,2} + \mathbf{V}_{0,1}) + \mathbf{T}_{11};$$
$$\mathbf{W}_{1,1} := \tau_8*\mathbf{V}_{2,2} + \mathbf{T}_{2112} + \mathbf{T}_{11};$$
$$\mathbf{W}_{2,1} := \tau_8*\mathbf{V}_{1,2} + \mathbf{T}_{2211} + \tau_5*\mathbf{V}_{2,1};$$
$$\mathbf{W}_{3,1} := \tau_7*(\mathbf{V}_{3,2} + \mathbf{V}_{1,2}) + \tau_3*(\mathbf{V}_{3,1} + 2*\mathbf{V}_{2,2} + \mathbf{V}_{3,3}) + \mathbf{T}_{21};$$
$$\mathbf{W}_{0,2} := \tau_7*(\mathbf{V}_{2,1} + \mathbf{V}_{0,1}) + \tau_3*(\mathbf{V}_{2,2} + \mathbf{V}_{0,2} + 2*\mathbf{V}_{1,1}) + \mathbf{T}_{12};$$
$$\mathbf{W}_{1,2} := \tau_8*\mathbf{V}_{2,1} + \mathbf{T}_{2211} + \mathbf{T}_{12};$$
$$\mathbf{W}_{2,2} := \tau_8*\mathbf{V}_{1,1} + \mathbf{T}_{2112} + \mathbf{T}_{22};$$
$$\mathbf{W}_{3,2} := \tau_7*(\mathbf{V}_{3,1} + \mathbf{V}_{1,1}) + \tau_3*(\mathbf{V}_{3,2} + \mathbf{V}_{1,3} + 2*\mathbf{V}_{2,1}) + \mathbf{T}_{22};$$
$$\mathbf{W}_{0,3} := \tau_4*(\mathbf{V}_{2,3} + \mathbf{V}_{2,1} + \mathbf{V}_{0,3} + \mathbf{V}_{0,1}) + \tau_3*(\mathbf{V}_{2,2} + \mathbf{V}_{1,3} + \mathbf{V}_{0,2}) + \mathbf{T}_{12};$$
$$\mathbf{W}_{1,3} := \tau_7*(\mathbf{V}_{2,3} + \mathbf{V}_{2,1}) + \tau_3*(\mathbf{V}_{1,3} + \mathbf{V}_{1,1} + 2*\mathbf{V}_{2,2}) + \mathbf{T}_{12};$$
$$\mathbf{W}_{2,3} := \tau_7*(\mathbf{V}_{1,3} + \mathbf{V}_{1,1}) + \tau_3*(\mathbf{V}_{2,3} + \mathbf{V}_{2,1} + 2*\mathbf{V}_{1,2}) + \mathbf{T}_{22};$$
$$\mathbf{W}_{3,3} := \tau_4*(\mathbf{V}_{3,3} + \mathbf{V}_{3,1} + \mathbf{V}_{1,3} + \mathbf{V}_{1,1}) + \tau_3*(\mathbf{V}_{3,2} + \mathbf{V}_{2,3} + \mathbf{V}_{2,1} + \mathbf{V}_{1,2}) + \mathbf{T}_{22};$$

end;

We now develop the procedure that splits the Bézier control graph $\bar{\mathbf{W}}$ in the u direction to produce the subgraphs $\bar{\mathbf{W}}^{LU}$ and $\bar{\mathbf{W}}^{RU}$. The procedure is based on the algorithm of Lane and Riesenfeld,[15] which can be written in matrix form as

$$\bar{\mathbf{W}}^{LU} = \bar{\mathbf{L}}\bar{\mathbf{W}} \tag{29}$$

and

$$\bar{\mathbf{W}}^{RU} = \bar{\mathbf{R}}\bar{\mathbf{W}} \tag{30}$$

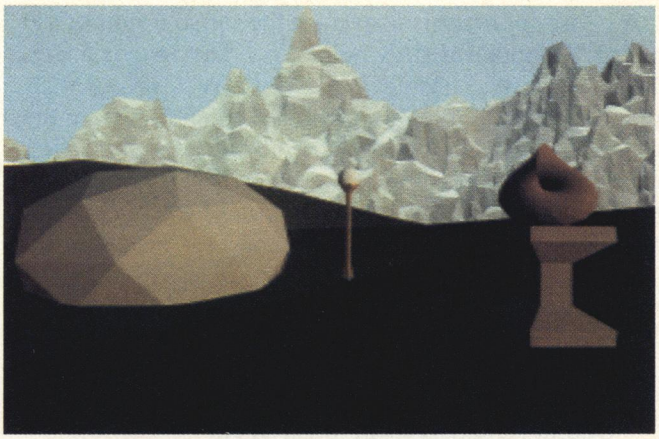
where $\bar{\mathbf{L}}$ and $\bar{\mathbf{R}}$ are the midpoint subdivision matrices from Equations 22 and 23.

A straightforward algorithm to implement Equations 29 and 30 could either call a routine that multiplies 4×4 matrices, or evaluate the expanded expression for each element. However, inspection shows that we can reuse the curve subdivision routine *Subdivide_Bézier* to accomplish our goal. Equation 30 shows that the columns of $\bar{\mathbf{W}}^{LU}$ can be found by applying $\bar{\mathbf{L}}$ to the columns of $\bar{\mathbf{W}}$; but this is exactly what *Subdivide_Bézier* does. The same is true for the right subgraph $\bar{\mathbf{W}}^{RU}$. Thus, a subdivision of a control graph in the u direction can be done by calling *Subdivide_Bézier* on the columns of $\bar{\mathbf{W}}$. To aid in the following discussion, let $Col_i(\bar{\mathbf{M}})$ denote the i^{th} column of matrix $\bar{\mathbf{M}}$. The procedure can now be stated as

```
procedure Subdivide_Bézier_Along_u(W̄, W̄^LU, W̄^RU)
local variable i;
begin
  for i = 0 to 3 do
      Subdivide_Bézier(Col_i(W̄), Col_i(W̄^LU),
        Col_i(W̄^RU));
end;
```

The Lane and Riesenfeld[15] algorithm for a split in the v direction can be written as*

*Superscript T denotes the transpose operation.



A landscape containing a variety of geometric primitives, including Beta2-spline surfaces. The lamppost, the hillside, and the pedestal are each defined by Beta2-spline surfaces. The lamppost has $\beta_2 = 0.0$, as does the hillside; the pedestal has $\beta_2 = 20.0$.

$$\bar{\mathbf{W}}^{LV} = \bar{\mathbf{W}}\bar{\mathbf{L}}^T \tag{31}$$

and

$$\bar{\mathbf{W}}^{RV} = \bar{\mathbf{W}}\bar{\mathbf{R}}^T \tag{32}$$

Once again, we could use a routine to multiply 4×4 matrices, or we could write out the expression for each element of $\bar{\mathbf{W}}^{LV}$ and $\bar{\mathbf{W}}^{RV}$. However, examination of Equations 31 and 32 reveals that the rows of $\bar{\mathbf{W}}^{LV}$ are the result of applying $\bar{\mathbf{L}}$ to the rows of $\bar{\mathbf{W}}$. The same is true of $\bar{\mathbf{W}}^{RV}$. Thus, a subdivision of a control graph in the v direction can be accomplished by applying the matrices $\bar{\mathbf{L}}$ and $\bar{\mathbf{R}}$ to the rows of $\bar{\mathbf{W}}$. The one slight complication is that *Subdivide_Bézier* expects to receive column matrices, not row matrices; a transpose operator can be used to relieve this type-clash. If $Row_i(\bar{\mathbf{M}})$ denotes the i^{th} row of matrix $\bar{\mathbf{M}}$, then a split in the v direction is described by

```
procedure Subdivide_Bézier_Along_v(W̄, W̄^LV, W̄^RV)
local variable i;
begin
  for i = 0 to 3 do
      Subdivide_Bézier(Row_i(W̄)^T, Row_i(W̄^LV)^T,
        Row_i(W̄^RV)^T);
end;
```

Next we examine how the surface patch defined by a relatively flat control graph can be approximated by polygons. One way of approximating the surface patch would be to construct a quadrilateral connecting the corner points $\mathbf{W}_{0,0}$, $\mathbf{W}_{0,3}$, $\mathbf{W}_{3,0}$, and $\mathbf{W}_{3,3}$. However, since the points may not be coplanar, the resulting polygon may be nonplanar. 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.[22]

A better approach is to average the corner points of the surface to obtain a fifth point $\mathbf{W}_a$. The four triangles $(\mathbf{W}_{0,0}, \mathbf{W}_{3,0}, \mathbf{W}_a)$, $(\mathbf{W}_{3,0}, \mathbf{W}_{3,3}, \mathbf{W}_a)$, $(\mathbf{W}_{3,3}, \mathbf{W}_{0,3}, \mathbf{W}_a)$, and $(\mathbf{W}_{0,3}, \mathbf{W}_{0,0}, \mathbf{W}_a)$ are then used to approximate the surface. This method does not introduce asymmetries, since both diagonals of the quadrilateral are treated equally. Pseudocode for the approximation of a relatively flat cubic Bézier patch by four triangles is

```
procedure Polygon_Approximate_Bézier(W̄)
local variable W_a;
begin
  W_a = 0.25*(W_0,0 + W_3,0 + W_0,3 + W_3,3);
  Output_Triangle(W_0,0, W_3,0, W_a);
  Output_Triangle(W_3,0, W_3,3, W_a);
  Output_Triangle(W_3,3, W_0,3, W_a);
  Output_Triangle(W_0,3, W_0,0, W_a);
end;
```

If $\bar{\mathbf{V}}$ represents the control graph for the Beta2-spline surface patch to be approximated, then the following pseudocode describes the process necessary to obtain a polygonal approximation to $\bar{\mathbf{V}}$:

```
procedure Approximate_Bézier_Surface(W̄,ε)
local variable W̄^LU, W̄^RU, W̄^LV, W̄^RV;
begin
  if W̄ is flat to within ε along u then
    if W̄ is flat to within ε along v then
      Polygon_Approximate_Bézier(W̄);
    else
      Subdivide_Bézier_Along_v(W̄, W̄^LV, W̄^RV);
      Approximate_Bézier_Surface(W̄^LV, ε);
      Approximate_Bézier_Surface(W̄^RV, ε);
    endif
  else
    Subdivide_Bézier_Along_u(W̄, W̄^LU, W̄^RU);
    Approximate_Bézier_Surface (W̄^LU,ε);
    Approximate_Bézier_Surface(W̄^RU,ε);
  endif
end;

procedure Approximate_β2_Surface(V̄,β2,ε)
local variable W̄;
begin
  Map_β2_Surface_To_Bézier(V̄,β2, W̄);
  Approximate_Bézier_Surface(W̄,ε);
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 $2+79d$ additions/subtractions, $7+42d$ multiplications, and one division. k calls to the Bézier subdivision routines requires $36dk$ additions/subtractions, $32dk$ multiplications, and no divisions. If there are k subdivision steps, there must be $k+1$ calls to *Polygon_Approximate_Bézier,* requiring $(k+1)3d$ additions/subtractions, $k+1$ multiplications, and no divisions. The total cost of approximating a d dimensional Beta2-spline surface patch with k subdivision steps is therefore $2+82d+39dk$ additions/subtractions, $7+43d+33dk$ multiplications, and one division.

Summary

We have 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, and 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 interior vertices of the control polygon; a Beta2-spline surface becomes a piecewise bilinear spline. 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 with the desired properties is obtained.

To aid the implementer of the Beta2-spline technique, detailed algorithms for the evaluation of the Beta2-spline basis functions and the approximation of Beta2-spline curves and surfaces via subdivision were presented. The subdivision algorithm approximates a Beta2-spline by transforming it into an equivalent Bézier curve or surface, then approximates the Bézier spline using recursive subdivision. A Beta2-spline object approximated using this algorithm and rendered on a high-resolution raster display system is shown in Figure 9. Other color images exemplifying this technique are shown on pages 46 and 55. ■

Acknowledgments

We would like to thank three members of the Berkeley Computer Graphics Laboratory: Mark Dippé, whose rendering software generated the shaded images, Steve Upstill, who created the landscape image, and Cecilia Aragon for fine-tuning the evaluation and subdivision algorithms. Thanks are also in order to Richard Fateman and his symbolic computation group at UC Berkeley for providing and supporting Vaxima, without which much of this work would have been impossible. The 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.

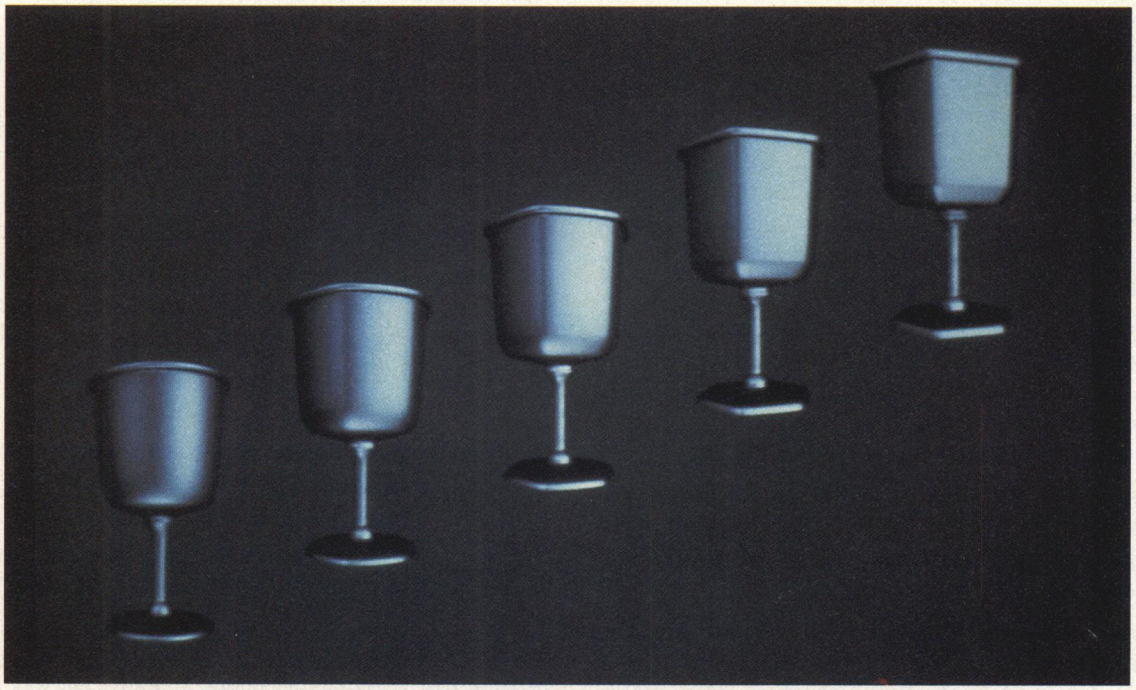


Figure 9. The pewter goblets are all defined by the same control graph. Their shape has been modified by changing only the value of the tension parameter β_2. The values of β_2 from left to right are 0, 5, 10, 20, and 50.

References

1. Brian A. Barsky, "The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures," PhD thesis, Univ. of Utah, Salt Lake City, Dec. 1981.
2. Brian A. Barsky, *Computer Graphics and Geometric Modelling Using Beta-splines,* Springer-Verlag, Tokyo, to appear 1985.
3. Brian A. Barsky, "The Beta-spline: A Curve and Surface Representation for Computer Graphics and Computer Aided Geometric Design," submitted for publication.
4. Ivor D. Faux and Michael J. Pratt, *Computational Geometry for Design and Manufacture,* Ellis Horwood, Ltd., 1979.
5. J. R. Manning, "Continuity Conditions for Spline Curves," *Computer J.,* Vol. 17, No. 2, May 1974, pp. 181-186.
6. Brian A. Barsky and John C. Beatty, "Local Control of Bias and Tension in Beta-splines," *ACM Trans. Graphics,* Vol. 2, No. 2, Apr. 1983, pp. 109-134. Also published in *Computer Graphics* (Proc. SIGGRAPH 83), Vol. 17, No. 3, July 1983, pp. 193-218.
7. Richard J. Fateman, "Addendum to the MACSYMA Reference Manual for the VAX," tech. report, Computer Science Division, Univ. of California, Berkeley, 1982.
8. Gregory M. Nielson, "Some Piecewise Polynomial Alternatives to Splines under Tension," in *Computer Aided Geometric Design,* Robert E. Barnhill and Richard F. Riesenfeld, eds., Academic Press, New York, 1974, pp. 209-235.
9. Richard H. Bartels and John C. Beatty, "Beta-splines with a Difference," tech. report no. CS-83-40, Department of Computer Science, Univ. of Waterloo, Waterloo, Ontario, Canada, May 1984.
10. T.N.T. Goodman, "Properties of Beta-splines," *J. Approximation Theory,* accepted for publication.
11. Edwin E. Catmull and James H. Clark, "Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes," *Computer-Aided Design,* Vol. 10, No. 6, Nov. 1978, pp. 350-355.

12. George M. Chaikin, "An Algorithm for High-Speed Curve Generation," *Computer Graphics and Image Processing,* Vol. 3, 1974, pp. 346-349.

13. D. W. H. Doo and M. A. Sabin, "Behavior of Recursive Division Surfaces Near Extraordinary Points," *Computer-Aided Design,* Vol. 10, No. 6, Nov. 1978, pp. 356-360.

14. 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,* Vol. 11, No. 3, Nov. 1979, pp. 290-297.

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

16. Robert W. Nydegger, "A Data Minimization Algorithm of Analytical Models for Computer Graphics," master's thesis, Univ. of Utah, Salt Lake City, 1972.

17. Edwin E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," PhD thesis, Univ. of Utah, Salt Lake City, Dec. 1974. Also tech. report no. UTEC-CSc-74-133, Department of Computer Science, Univ. of Utah.

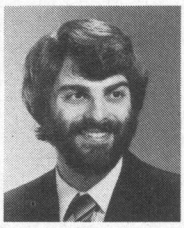
18. Brian A. Barsky, "Arbitrary Subdivision of Bézier Curves," submitted for publication.

19. Ronald N. Goldman, "Using Degenerate Bézier Triangles and Tetrahedra to Subdivide Bézier Curves," *Computer-Aided Design,* Vol. 14, No. 6, Nov. 1982, pp. 307-311.

20. Jeffrey M. Lane and Richard F. Riesenfeld, "Bounds on a Polynomial," *BIT,* Vol. 21, No. 1, 1981, pp. 112-117.

21. Ronald N. Goldman, private communication.

22. Brian A. Barsky, Tony D. DeRose, and Mark D. Dippé, "An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-spline Objects," submitted for publication.



Brian A. Barsky is an assistant professor of computer science at the University of California, Berkeley, where he is director of the Berkeley Computer Graphics Laboratory, and he is an adjunct assistant professor of computer science at the University of Waterloo in Waterloo, Ontario, Canada. He is currently a visiting researcher at the Laboratoire Image at Ecole Nationale Supérieure des Télécommunications in Paris. He was a visiting researcher with the Computer Aided Design and Manufacturing Group at the Sentralinsitutt for Industriell Forskning (Central Institute for Industrial Research) in Oslo.

Barsky's research interests include computer-aided geometric design and modeling, and interactive three-dimensional computer graphics. He attended McGill University, where he received a DCS in engineering and a BSc in mathematics and computer science. He studied computer graphics and computer science at Cornell University, where he earned an MS degree. He earned a PhD degree in computer science from the University of Utah.

Barsky was the technical program chairman of SIGGRAPH 85. He is a member of ACM SIGGRAPH, the National Computer Graphics Association, the IEEE Computer Society, the Canadian Man-Computer Communications Society, and the Society for Industrial and Applied Mathematics.

Barsky's permanent address is Berkeley Computer Graphics Laboratory, Computer Science Division, University of California, Berkeley, CA 94720. His current address is Laboratoire Image, Ecole Nationale Supérieure des Télécommunications, 46 rue Barrault, 75634 Paris CEDEX 13, France.

Tony D. DeRose is currently a doctoral student and research assistant at the University of California, Berkeley. His research interests include the mathematical aspects of computer-aided geometric design and their relationship to graphical user interfaces. In 1981 he received his BS in physics at the University of California, Davis, and is expecting a PhD in computer science in December of this year. Starting in the fall, DeRose will begin an assistant professorship in computer science at the University of Washington.

DeRose's address is Computer Science Department, FR-35, University of Washington, Seattle, WA 98195.