An Adaptive Subdivision Method With Crack Prevention for Rendering Beta-spline Objects

Brian A. Barsky
Tony D. DeRose
Mark D. Dippé

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

ABSTRACT

Adaptive subdivision is a method of creating polygonal approximations to spline surfaces. An adaptive subdivision algorithm takes as input a spline surface and a tolerance ε, and outputs a piecewise planar approximation to the surface that is guaranteed to differ from the actual surface by a distance no greater than ε. These algorithms proceed by recursively "splitting" the surface into smaller subsurfaces, ultimately approximating subsurfaces with planar polyhedra. These algorithms are therefore characterized by the mathematics behind the splitting of a surface, the test that is used to determine when to stop the recursion, and the method by which a subsurface is approximated by polyhedra. Algorithms of this type are currently known for spline techniques such as Bézier and B-splines.

The purpose of this paper is two-fold. First, we describe the Beta-spline curve and surface technique and derive the equations governing the splitting of Beta-spline curves and surfaces. Second, we present a very general adaptive subdivision algorithm that can be used with a variety of surface techniques. It incorporates splitting criteria based on *flatness* and prevents cracks from occurring between approximating polyhedra. The testing required to determine when surface splitting can stop can be very expensive and methods for reducing these costs are examined. The tolerance controlling the splitting process may itself be adaptive, so that as an object moves farther away the tolerance is automatically increased. These ideas have all been used in the implementation of a surface design and rendering system.

This work was supported in part by a National Science Foundation Presidential Young Investigator Award (number DCI-8451997) and Engineering Research Initiation Grant (number ECS-8204381), by Defense Advanced Research Projects Agency conXtracts (numbers N00039-82-C-0235 and N00039-84-C-0089), by the Control Data Corporation, and by AT&T Bell Laboratories.

1. Introduction

Most popular rendering algorithms rely on planar polygonal data descriptions due to their compact representations and geometric simplicity. Unfortunately, they are not sufficiently flexible for many of today's graphics applications. For instance, suppose that a common object such as a glass is modeled as a set of abutting triangles. When the glass is rendered from the viewpoint of an observer very far away, the triangular approximation may lead to a large number of sub-pixel sized triangles; the approximation is too accurate to resolve on a discrete medium such as a frame buffer. At the other extreme, as the eyepoint is moved toward the object, the approximation will begin to visibly break down. When viewed from a sufficiently close distance, the silhouette edges of the glass will show discontinuities in slope, detracting from the realism of the rendering.

Fortunately, there is an alternative to polygonal modeling. If the glass is defined using modern spline surfaces, we obtain a mathematical function describing the object. The mathematical model is, in essence, a description that is infinitely detailed. Of course when the object is rendered it must be approximated, possibly by polygons, but the accuracy of the approximation can be governed by viewing parameters. Thus, if viewed from a great distance, it is sufficient to generate a small number of approximants from the mathematical definition. The number of approximants can be increased as the distance to the viewer decreases. This technique guarantees that the object will always appear to be smooth without unwarranted or invisible detail.

Adaptive subdivision is a method of approximating a surface by polygons to a given tolerance in an intelligent fashion (Lane[18]). This is a definite improvement over earlier subdivision schemes that always subdivided the surface down to the size of a pixel (i.e., Catmull[9]). In regions of relative flatness, a good characterization of the surface can be achieved with a small number of large polygons. Regions that are highly curved require a large number of small polygons. Since the surface is approximated with planar polygons, the object can be designed with all the advantages of surfaces but rendered using all of the available polygon rendering algorithms. Therefore, subdivision plays the important role of bridging the gap between modern surface definitions and efficient polygon based renderers.

The ideas of surface modeling, subdivision, and user friendliness can be incorporated in a design system with which someone unfamiliar with mathematics and computers can design and render complex objects. Moreover, the result of such a design session is not merely a "pretty picture"; it is a model describing the object in a concise mathematical form, making this type of system well tailored to CAD/CAM applications.

This paper describes the surface technique we use, its corresponding subdivision algorithm, and a method for preventing the "cracks" that have plagued previous subdivision algorithms.

1.1. Notation

Before delving into the mathematical details, the introduction of notational conventions will prove to be useful.

- Scalars are represented in italics, as in x and Y.
- Vectors are denoted by boldface type, i.e. $\mathbf{V} = (V_x, V_y, V_z)$ defines a three dimensional vector $\mathbf{V}$. The boldface convention also holds for vector-valued functions. If $\mathbf{L}(u)$ represents a vector-valued function of u, $\mathbf{L}(u)$ can be written in component form as

$$\mathbf{L}(u) = (L_x(u), L_y(u), L_z(u))$$

- Sequences of vectors appear in boldface surrounded by $< >$. For example, $<\mathbf{V}> = (\mathbf{V}_1, \mathbf{V}_2, \cdots, \mathbf{V}_m)$ is a sequence of m vectors.
- A matrix is denoted by a boldface character ornamented with a diacritical bar as in

$$\overline{\mathbf{M}} = \begin{bmatrix} x & -y \\ y & x \end{bmatrix}.$$

2. Basic Beta-splines

The next few sections briefly introduce the Beta-spline technique for curves and surfaces. For a more complete treatment see Barsky.[3,4]

The Beta-spline, developed by Barsky,[1] is a departure from its predecessors in that its derivation is based on fundamental geometric measures rather than abstract algebraic quantities. In particular, previous spline formalisms have assumed that derivatives (usually first and second derivatives) must be continuous for the curve or surface to appear smooth. Because of the parametric form of these techniques, conservation of first and second derivative is overly constraining; it is sufficient to require that the unit tangent and curvature vectors be continuous. Since the constraining equations are more relaxed, new design freedom is gained without resorting to a higher order polynomial. To be specific, two new degrees of freedom, *bias* (β_1) and *tension* (β_2) are obtained using a cubic Beta-spline. For appropriate values of bias and tension the Beta-spline reduces to a uniform cubic B-spline, demonstrating that the Beta-spline is actually a generalization of the B-spline.

The *shape parameters* (bias and tension) associated with a Beta-spline may be constant over the entire extent of the curve or surface, or may be allowed to vary.[6,7] This paper only addresses the subdivision and rendering of Beta-splines with constant shape parameters.

2.1. Beta-spline Curves

Given a sequence of vertices in three-space $<\mathbf{V}> = (\mathbf{V}_0,\mathbf{V}_1,\cdots,\mathbf{V}_m)$, the Beta-spline curve defined by the sequence is composed of $m-2$ curve segments, the i^{th} of which, $\mathbf{Q}_i(\beta_1,\beta_2;u)$, is

$$\mathbf{Q}_i(\beta_1,\beta_2;u) = \sum_{r=-2}^{1} \mathbf{V}_{i+r} b_r(\beta_1,\beta_2;u); \quad 0 \le u \le 1,\ i = 2,\cdots,m-1 \tag{2.1}$$

$\mathbf{Q}_i(\beta_1,\beta_2;u)$ can also be expressed in matrix form as

$$\mathbf{Q}_i(\beta_1,\beta_2;u) = [b_{-2}(\beta_1,\beta_2;u)\ b_{-1}(\beta_1,\beta_2;u)\ b_0(\beta_1,\beta_2;u)\ b_1(\beta_1,\beta_2;u)] \begin{bmatrix} \mathbf{V}_{i-2} \\ \mathbf{V}_{i-1} \\ \mathbf{V}_i \\ \mathbf{V}_{i+1} \end{bmatrix} \tag{2.2}$$

$b_k(\beta_1,\beta_2;u)$ is a cubic polynomial called the k^{th} Beta-spline basis function. The four basis functions are derived so as to preserve continuity of unit tangent and curvature vectors. From Barsky,[1] they are:

$$b_{-2}(\beta_1,\beta_2;u) = \frac{2\beta_1^3}{\delta}(1-u)^3 \tag{2.3}$$

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

$$b_0(\beta_1,\beta_2;u) = \frac{1}{\delta}(2\beta_1^2 u^2(-u+3)+2\beta_1 u(-u^2+3)+\beta_2 u^2(-2u+3)+2(-u^3+1))$$

$$b_1(\beta_1,\beta_2;u) = \frac{2u^3}{\delta}$$

where $\delta = 2\beta_1^3+4\beta_1^2+4\beta_1+\beta_2+2$, β_1 is the bias and β_2 is the tension.

A sequence such as $<\mathbf{V}>$, used to define a spline, is called a *control polygon*. Intuitively, a Beta-spline control polygon governs the general behavior of the curve. The curve itself can be thought of as a smoothly blended approximation of the control polygon (Figure 2.1).

Equations (2.1) (2.3) show that a Beta-spline curve is composed of a set of smoothly joined cubic curve segments; in other words, a Beta-spline curve is a piecewise cubic polynomial. Note that each curve segment is defined by only four adjacent vertices of the control polygon. For instance, the i^{th} curve segment is defined by the vertices $\mathbf{V}_{i-2},\mathbf{V}_{i-1},\mathbf{V}_i,\mathbf{V}_{i+1}$. The $i+1^{\text{st}}$ segment depends only on the vertices $\mathbf{V}_{i-1}$, $\mathbf{V}_i$, $\mathbf{V}_{i+1}$, and $\mathbf{V}_{i+2}$. This fact has some very important and useful consequences, as follows.

First, a Beta-spline defined by a control polygon of more than four vertices is equivalent to a set of Beta-splines, each with a control polygon of exactly four vertices. The equivalence allows derivations of the properties of the Beta-spline to

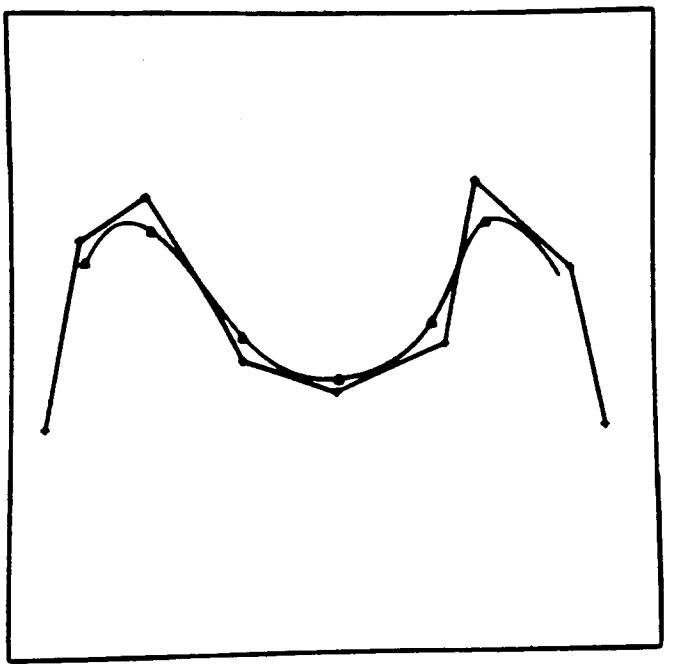


Figure 2.1.
A control polygon and the corresponding Beta-spline curve with $\beta_1 = 1$ *and* $\beta_2 = 0$. *Crosses denote control vertices and triangles denote the joints (points where the polynomial segments meet).*

concentrate on a spline comprised of a single segment.

Second, if a single vertex is moved, only four curve segments will be affected. More specifically, movement of the j^{th} vertex only alters $\mathbf{Q}_{j-2}$, $\mathbf{Q}_{j-1}$, $\mathbf{Q}_j$, and $\mathbf{Q}_{j+1}$. This is known as the property of *local control*.

Third, the basis functions guarantee that the curve segments will join smoothly, independent of the positions of the control vertices. The basis functions also ensure that the curve will not leave the region defined by the *convex hull* of $<\mathbf{V}>^{\dagger}$. The convex hull of $<\mathbf{V}>$ can be found by imagining a rubber band stretched around $<\mathbf{V}>$. The area within the boundaries of the rubber band defines the convex hull. This property allows bounding regions for the curve to be easily calculated, a fact that greatly increases the efficiency of many algorithms dealing with the curve form.

What makes Beta-splines different from the user's point of view are the shape parameters β_1 and β_2. Both parameters apply tension to the curve, but they differ in rather subtle ways. β_1 measures the relative amount of bias, or asymmetry.

---

† Actually, a stronger statement can be made. A given curve segment is guaranteed to remain within the convex hull of the four vertices that define it. Therefore, the whole curve, due to the equivalence property above, remains within the union of the convex hull of each of the curve segments.

When $\beta_1 = 1$ the curve is said to be unbiased. As β_1 is increased, the curve begins to skew to one side and approach the control polygon in an asymmetric fashion. Replacing β_1 by its reciprocal value causes the curve to be skewed an equal amount in the opposite direction. Figure 2.2 shows three curves defined by the same control polygon. The top curve has $\beta_1 = 1$, the one in the middle has $\beta_1 = 1/2$, and the one on bottom has $\beta_1 = 2$. Observe that the two lower curves have begun to approach the control polygon, but in a non-uniform way. Note also that the joints of the lower curves have been skewed in opposite directions.

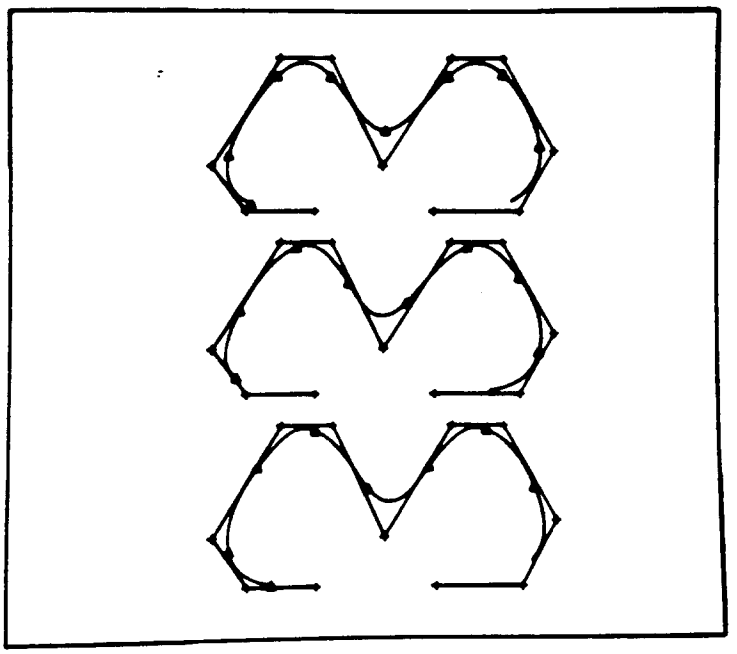


Figure 2.2.

The effect of bias (β_1) on a Beta-spline curve. Note how the lower curves approach the control polygon asymmetrically, and how the endpoints have been biased in opposite directions.

The tension parameter (β_2) measures the amount of symmetric tension applied to the curve. When $\beta_2 = 0$, the curve is said to be untensed. As β_2 is increased, the curve $\mathbf{Q}(u)$ flattens and uniformly approaches the control polygon $<\mathbf{V}>$; in the limit of infinite tension, $\mathbf{Q}(u)$ becomes indistinguishable from $<\mathbf{V}>$.

The extra freedom provided by the shape parameters means that the design procedure using Beta-splines is somewhat more flexible than the procedure for B-splines. A typical design application using B-splines involves the definition of a control polygon, generation of the corresponding curve, then movement of existing vertices or insertion of new vertices to refine the curve in regions of particular interest. A design process exploiting Beta-splines does not always require that vertices be moved or added; the shape parameters can also be used to refine the curve. For example, Figure 2.3 shows that tension provides the designer with an

intuitively pleasing parameter not available with other polynomial splines, cubic or otherwise.

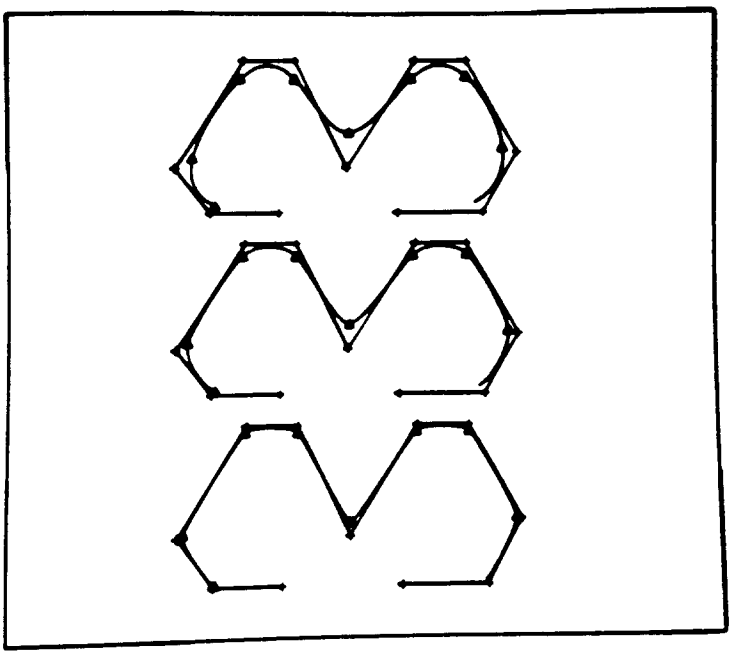


Figure 2.3.
The effect of tension (β_2) is shown by the diagrams above. The value of β_2 is 0 for the top curve, 5 for the middle curve, and 20 for the bottom curve.

When $\beta_1 = 1$ and $\beta_2 = 0$, the Beta-spline basis functions reduce to the uniform cubic B-spline basis functions. In other words, a uniform cubic B-spline is an unbiased, untensed Beta-spline.

2.2. Beta-spline Surfaces

A Beta-spline surface is described by an array of vertices in three-space $<\mathbf{V}> = (\mathbf{V}_{00}, \mathbf{V}_{01}, \cdots \mathbf{V}_{mn})$. In a manner analogous to the Beta-spline curve, a Beta-spline surface is composed of a set of surface patches, the ij^{th} patch $\mathbf{S}_{ij}(\beta_1,\beta_2;u,v)$ defined as

$$\mathbf{S}_{ij}(\beta_1,\beta_2;u,v) = \sum_{r=-2}^{1} \sum_{s=-2}^{1} \mathbf{V}_{i+rj+s} b_r(\beta_1,\beta_2;u) b_s(\beta_1,\beta_2;v) \tag{2.4}$$

which in matrix form is

$$\mathbf{S}_{ij}(\beta_1,\beta_2;u,v) = \tag{2.5}$$

$$[b_{-2}(\beta_1,\beta_2;u)\ b_{-1}(\beta_1,\beta_2;u)\ b_0(\beta_1,\beta_2;u)\ b_1(\beta_1,\beta_2;u)]\overline{\mathbf{V}}_{ij} \begin{bmatrix} b_{-2}(\beta_1,\beta_2;v) \\ b_{-1}(\beta_1,\beta_2;v) \\ b_0(\beta_1,\beta_2;v) \\ b_1(\beta_1,\beta_2;v) \end{bmatrix}$$

where

$$\bar{\mathbf{V}}_{ij} = \begin{bmatrix} \mathbf{V}_{i-2j-2} & \mathbf{V}_{i-2j-1} & \mathbf{V}_{i-2j} & \mathbf{V}_{i-2j+1} \\ \mathbf{V}_{i-1j-2} & \mathbf{V}_{i-1j-1} & \mathbf{V}_{i-1j} & \mathbf{V}_{i-1j+1} \\ \mathbf{V}_{ij-2} & \mathbf{V}_{ij-1} & \mathbf{V}_{ij} & \mathbf{V}_{ij+1} \\ \mathbf{V}_{i+1j-2} & \mathbf{V}_{i+1j-1} & \mathbf{V}_{i+1j} & \mathbf{V}_{i+1j+1} \end{bmatrix}$$

A sequence such as $<\mathbf{V}>$, used to define a surface, is called a *control hull* or *control graph*. An example of a control graph is shown in Figure 2.4 together with the surface defined by it. Note that the connectivity of the graph is implicitly determined by considering the graph as a two-dimensional array of vertices; two vertices share an edge if and only if they are adjacent in the array.

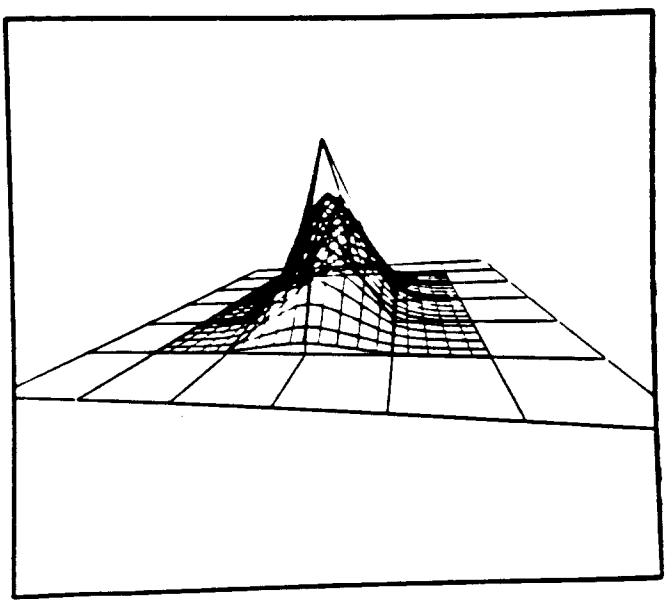


Figure 2.4.
A control graph and the corresponding Beta-spline surface with $\beta_1 = 1$ *and* $\beta_2 = 5$.

For each property of Beta-spline curves, there is a corresponding property for Beta-spline surfaces. Of particular interest are the properties of local control, convex hull, and the behavior of the shape parameters.

When a vertex of a Beta-spline control graph is perturbed, only sixteen patches are affected. If the ij^{th} vertex is moved, only the patches

$$\mathbf{S}_{i+rj+s}(\beta_1,\beta_2;u,v); \quad r = -2.-1,0,1 \quad s = -2,-1,0,1 \tag{2.6}$$

will be altered. Local control of the surface implies that the surface definition need not be entirely recomputed when a vertex is moved (a common operation in the design process). In fact, the algorithm for updating the surface in response to a vertex perturbation need only be concerned with the patches given in (2.6). Moreover, Barsky[5] has shown that a patch (or curve) affected by a vertex perturbation can be updated in an incremental way, requiring only one multiplication and one addition for each point on the surface (or curve).

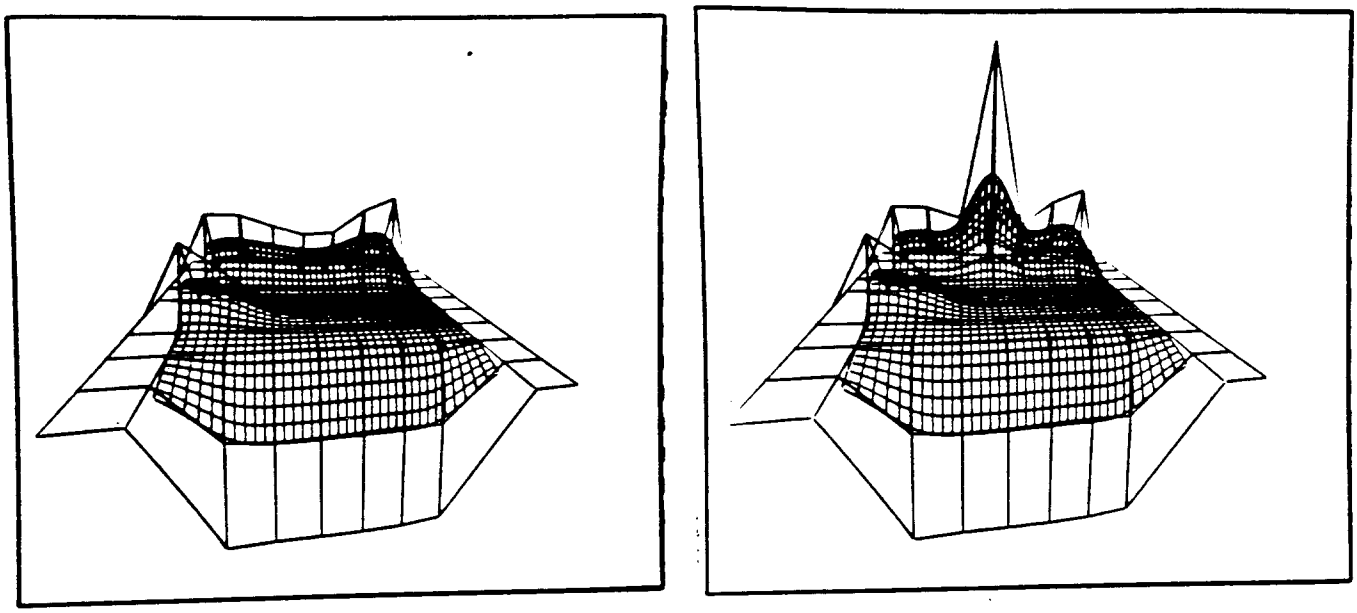


Figure 2.5.

The figures above demonstrate the property of local control for Beta-spline surfaces. The two surfaces are identical except for the position of one of the vertices. Note that only a small region of the surface is affected.

Equation (2.6) shows that sixteen patches are dependent on a particular vertex. It is also the case that each patch depends on only sixteen vertices, arranged in a 4 by 4 array surrounding the patch. This implies that a Beta-spline surface defined by a control graph of size larger than 4 by 4 can be expressed as a set of surfaces, each composed of a single patch defined by a control graph of size exactly 4 by 4.

Since the Beta-spline curve possesses the convex hull property, it is also displayed by the Beta-spline surface. The convex hull of a control graph is obtained by imagining a rubber membrane stretched taut around the graph; the volume within the membrane is the convex hull. The region of space that can be occupied by a single Beta-spline patch is the convex hull of its control graph. For a Beta-spline surface composed of more than one patch, the convex hull of the aggregate is the union of the convex hulls of the individual patch control graphs.

The convex hull property can be exploited in many ways. It can, for example, be used in conjunction with subdivision (to be explained in section 3.1) to develop a fast occlusion algorithm. If two control graphs do not occlude one another, neither can the surfaces they define. If the graphs do interfere, subdivision can be used as a means of resolving the interference.

As might be expected, bias and tension alter the shape of a surface in the same way they did for curves. For instance, as the tension is increased on a surface, the surface begins to uniformly approach the control graph. Intuitively, the surface takes on more polygonal character as the tension is increased. In the limit of infinite tension the surface converges to the control graph.

3. Subdivision of Spline Curves

3.1. Mathematical Derivation

Let $\mathbf{C}[a,b](u)$ be the spline curve computed from a linear combination of some set of basis functions $W_i(u)$ when the parameter $u \varepsilon [a,b]$; i.e.

$$\mathbf{C}[a,b](u) = \sum_{r=0}^{m} \mathbf{V}_r W_r(u); \quad a \le u \le b \tag{3.1}$$

When no brackets are shown, as in $\mathbf{C}(u)$, $\mathbf{C}[0,1](u)$ is implied.

The *subdivision problem* for curves may be mathematically stated as: [12] Find the control polygons

$$<\mathbf{V}^L> = (\mathbf{V}_0^L \cdots \mathbf{V}_m^L) \tag{3.2}$$

and

$$<\mathbf{V}^R> = (\mathbf{V}_0^R \cdots \mathbf{V}_m^R) \tag{3.3}$$

such that

$$\mathbf{C}[0,u^*](u) = \mathbf{C}^L[0,1](u^L) \tag{3.4}$$

and

$$\mathbf{C}[u^*,1](u) = \mathbf{C}^R[0,1](u^R) \tag{3.5}$$

where

$$\mathbf{C}^L[0,1](u^L) = \sum_{s=0}^{m} \mathbf{V}_s^L W_k(u^L); \quad 0 \le u^L \le 1 \tag{3.6}$$

and

$$\mathbf{C}^R[0,1](u^R) = \sum_{r=0}^{m} \mathbf{V}_r^R W_r(u^R); \quad 0 \le u^R \le 1 \tag{3.7}$$

The quantity u^* is called the subdivision point. If $u^* = 1/2$, the curve is said to have undergone *midpoint subdivision*.

When $\mathbf{C}(u)$ is subdivided we obtain two new curve segments $\mathbf{C}^L(u)$ and $\mathbf{C}^R(u)$ whose union is exactly $\mathbf{C}(u)$. The superscripts are inspired by the fact that if the vertices of $<\mathbf{V}>$ trace out a polygon from left to right, then $<\mathbf{V}^L>$ is the subdivided polygon on the left, $<\mathbf{V}^R>$ is the right subdivided polygon. In keeping with this convention, all quantities superscripted with L refer to the left subcurve, and superscript R refers the right subcurve.

The subdivision constraint equation (3.4) can be expressed in terms of a single parameter u, instead of the two parameters u and u^L by noting that

$$\mathbf{C}[0,u^*](u) = \mathbf{C}[0,1](u^* u) \tag{3.8}$$

Thus, (3.4) becomes

$$\mathbf{C}[0,1](u^* u) = \mathbf{C}^L[0,1](u) \tag{3.9}$$

Equation (3.5) can also be written in terms of a single parameter by noting that

$$\mathbf{C}[u^*,1](u) = \mathbf{C}[0,1](u^* + (1-u^*)u) \tag{3.10}$$

Substitution of (3.10) into (3.5) gives

$$\mathbf{C}[0,1](u^* + (1-u^*)u) = \mathbf{C}^R[0,1](u) \tag{3.11}$$

If we let $\overline{\mathbf{W}}(u)$ represent a row vector of the basis functions, i.e.

$$\overline{\mathbf{W}}(u) = [W_0(u)\ W_1(u)\ \cdots\ W_m(u)]$$

then the reparametrized constraint equations (3.9) and (3.11) can alternately be written in matrix form as

$$\overline{\mathbf{W}}(u^* u)\overline{\mathbf{V}} = \overline{\mathbf{W}}(u)\overline{\mathbf{V}}^L \tag{3.12}$$

and

$$\overline{\mathbf{W}}(u^*+(1-u^*)u)\overline{\mathbf{V}} = \overline{\mathbf{W}}(u)\overline{\mathbf{V}}^R \tag{3.13}$$

respectively, where

$$\overline{\mathbf{V}} = [\mathbf{V}_0\ \mathbf{V}_1\ \cdots\ \mathbf{V}_m]^T$$

$$\overline{\mathbf{V}}^L = [\mathbf{V}_0^L\ \mathbf{V}_1^L\ \cdots\ \mathbf{V}_m^L]^T$$

$$\overline{\mathbf{V}}^R = [\mathbf{V}_0^R\ \mathbf{V}_1^R\ \cdots\ \mathbf{V}_m^R]^T$$

Since polynomial basis functions can always be linearly reparametrized by matrix multiplication, [12,15] $\overline{\mathbf{W}}(u^*u)$ and $\overline{\mathbf{W}}(u^*+(1-u^*)u)$ can be expressed as

$$\overline{\mathbf{W}}(u^*u) = \overline{\mathbf{W}}(u)\overline{\mathbf{L}} \tag{3.14}$$

and

$$\overline{\mathbf{W}}(u^*(1-u^*)u) = \overline{\mathbf{W}}(u)\overline{\mathbf{R}} \tag{3.15}$$

for some m-by-m matrices $\overline{\mathbf{L}}$ and $\overline{\mathbf{R}}$. Substitution of (3.14) into (3.12) results in

$$\overline{\mathbf{W}}(u)\overline{\mathbf{L}}\overline{\mathbf{V}} = \overline{\mathbf{W}}(u)\overline{\mathbf{V}}^L \tag{3.16}$$

Thus, the left subdivided control polygon $\overline{\mathbf{V}}^L$ is given by

$$\overline{\mathbf{V}}^L = \overline{\mathbf{L}}\overline{\mathbf{V}} \tag{3.17}$$

In a similar fashion, the right subdivided control polygon $\overline{\mathbf{V}}^R$ can be shown to be given by

$$\overline{\mathbf{V}}^R = \overline{\mathbf{R}}\overline{\mathbf{V}} \tag{3.18}$$

In other words, the matrices that reparametrize the basis functions also subdivide the control polygon. This result is independent of the type of polynomial spline being used.[12]

3.2. Subdivision Schema for Curves

Using subdivision we can obtain a piecewise linear approximation to the curve to any desired accuracy as follows. Let us for the moment assume that the basis functions are constructed so that the curve lies within the convex hull of the control polygon and interpolates the first and last control vertices. With these assumptions, if the control polygon defining the curve is flat to within a given tolerance, the curve can be adequately approximated by the linear segment $\mathbf{V}_0\mathbf{V}_m$. If the polygon fails the flatness test, the polygon is subdivided into two parts, both of which are approximated by recursive application of the algorithm. The union of all the linear segments thus produced forms the piecewise linear approximation to the original curve.‡

It is important to note that although we are conceptually subdividing the *curve*, in practice we are using the properties of the basis functions to subdivide the *control polygon*. This obviates the need to explicitly compute points on the curve by direct evaluation of (3.1). Instead, the endpoints of the subdivided control polygons can be thought of as implicitly obtained evaluations.

A pseudo-code version of the recursive subdivision algorithm is

```
Approximate_Curve( <V>, ε)
<V>: Control Polygon;
ε: flatness;

begin
  if <V> is within ε of flatness then
    Output the line from V0 to Vm
  else
    Subdivide(<V>, <V^L>, <V^R>)
    Approximate_Curve(<V^L>, ε)
    Approximate_Curve(<V^R>, ε)
end
```

Recursive subdivision with flatness testing can be thought of as being an adaptive approximation algorithm in that the local flatness of the curve drives the recursion. In a region where the curve is relatively flat, only one linear segment will be produced. If the curve has a region of high curvature, the algorithm will be forced to generate a large number of small linear segments to properly represent the curve.

Older methods have approximated curves by stepping along in parametric space by some constant increment. However, a given distance in parametric space can correspond to an arbitrarily large distance in geometric space since the geometric distance is proportional to the distance between vertices of the control polygon. In other words, the older methods approximated the curve using a *parametric tolerance*. The adaptability of the subdivision algorithm allows an

‡ R. Goldman has recently shown that this process is guaranteed to converge to the original curve.

approximation in terms of a purely *geometric tolerance*. Intuitively, the subdivision algorithm generates an approximation that is uniformly good over the entire extent of the curve; the older method does not.

Although Beta-spline curves have the convex hull property, they do not interpolate the endpoints of the control polygon. Thus, a flat control polygon cannot be approximated by a line segment connecting the first and last vertices; instead, the endpoints of the curve must be explicitly computed. The line segment joining the curve endpoints can be used as a piecewise linear approximant, but from an algorithmic point of view it is more efficient to convert the Beta-spline control polygon into an equivalent Bézier control polygon, then subdivide the Bézier polygon to obtain the linear approximation to the curve. This method is more efficient because it obviates the need to explicitly compute the endpoints of a curve segment once a flat control polygon is obtained. For these reasons, we now turn to the subdivision of Bézier curves. The conversion of Beta-spline curves into Bézier curves will then be discussed in section 3.4.

3.3. Subdivision of Bézier Curves

A Bézier curve is constructed from a control polygon as defined in equation (3.1) where the i^{th} Bézier basis function $B_i(u)$ taking the place of $W_i(u)$. The Bézier basis functions may be compactly defined as

$$B_i(u) = \binom{m}{i} u^i (1-u)^{m-i} \tag{3.19}$$

where $\binom{m}{i}$ is the familiar binomial coefficient.

A Bézier curve interpolates the endpoints of the control polygon and obeys the convex hull property; thus, we can compute the control polygons $<\mathbf{V}^L>$ and $<\mathbf{V}^R>$, then recursively subdivide to obtain a piecewise linear approximation.

We now develop the equations governing the subdivision of Bézier curves at an arbitrary subdivision point u^*, the derivation of which is based on work recently done by Barsky.[2] For simplicity we will initially consider the subdivision of cubic Bézier curves, then generalize the result to arbitrary order.

Let the control polygon for the cubic curve to be subdivided be denoted by $<\mathbf{V}^0> = (\mathbf{V}_0^0, \mathbf{V}_1^0, \mathbf{V}_2^0, \mathbf{V}_3^0)$. Locate the vertex $\mathbf{V}_0^1$ by dividing the segment $\mathbf{V}_0^0\mathbf{V}_1^0$ such that

$$u^* = \frac{|\mathbf{V}_0^0\mathbf{V}_0^1|}{|\mathbf{V}_0^0\mathbf{V}_1^0|} \tag{3.20}$$

The vertices $\mathbf{V}_1^1$ and $\mathbf{V}_2^1$ are obtained by a similar division of the segments $\mathbf{V}_1^0\mathbf{V}_2^0$ and $\mathbf{V}_2^0\mathbf{V}_3^0$ respectively. The construction continues by obtaining the vertices $\mathbf{V}_0^2$ and $\mathbf{V}_1^2$ by dividing the segments $\mathbf{V}_0^1\mathbf{V}_1^1$ and $\mathbf{V}_1^1\mathbf{V}_2^1$, respectively. Finally, the vertex $\mathbf{V}_0^3$ is located by dividing the segment $\mathbf{V}_0^2\mathbf{V}_1^2$. The properties of the Bézier basis functions can be used to show that

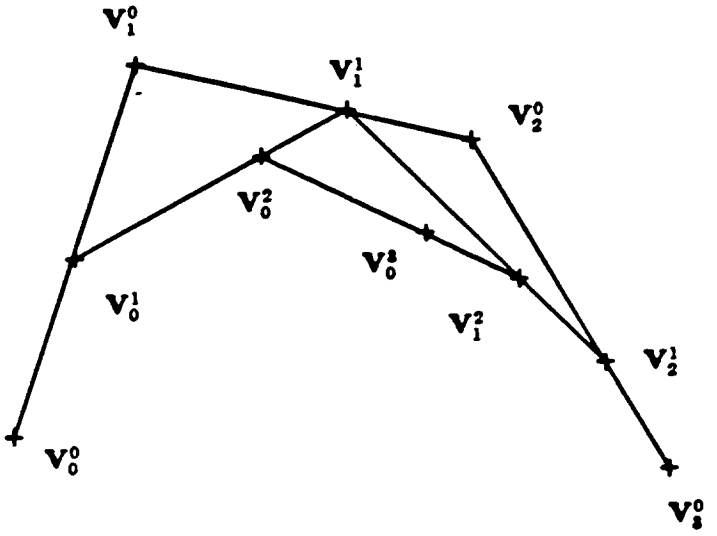


Figure 3.1.

The geometric construction of the subdivided polygons $<\mathbf{V}^L> = (\mathbf{V}_0^0,\mathbf{V}_0^1,\mathbf{V}_0^2,\mathbf{V}_0^3)$ *and* $<\mathbf{V}^R> = (\mathbf{V}_0^3,\mathbf{V}_1^2,\mathbf{V}_2^1,\mathbf{V}_3^0)$ *for a cubic Bézier control polygon.*

$$\mathbf{C}[0,1](u^*) = \mathbf{V}_0^3 \tag{3.21}$$

Moreover, we can identify $<\mathbf{V}^L>$ as the sequence $(\mathbf{V}_0^0,\mathbf{V}_0^1,\mathbf{V}_0^2,\mathbf{V}_0^3)$, and $<\mathbf{V}^R>$ as the sequence $(\mathbf{V}_0^3,\mathbf{V}_1^2,\mathbf{V}_2^1,\mathbf{V}_3^0)$.

The geometric construction can be made algebraic by noting that

$$\mathbf{V}_i^j = (1-u^*)\mathbf{V}_i^{j-1} + u^*\mathbf{V}_{i+1}^{j-1} \quad i=0,1,2 \;\; j=1\cdots 3-i \tag{3.22}$$

In fact, this form holds for all Bézier curves of any degree m. The generalized expression[2] is

$$\mathbf{V}_i^j = (1-u^*)\mathbf{V}_i^{j-1} + u^*\mathbf{V}_{i+1}^{j-1} \quad i=0,1,\ldots,m-1 \;\; j=1\cdots m-i \tag{3.23}$$

Equation (3.23) represents a recurrence relation for the vertices of the various control polygons in terms of the vertices of $<\mathbf{V}^0>$. Since the relation is linear, the recurrence can be unraveled, solved for the vertices comprising $<\mathbf{V}^L>$ and $<\mathbf{V}^R>$, and written in matrix form as

$$\overline{\mathbf{V}}^L = \overline{\mathbf{L}}_B\overline{\mathbf{V}} \tag{3.24}$$

and

$$\bar{\mathbf{V}}^R = \bar{\mathbf{R}}_B \bar{\mathbf{V}} \tag{3.25}$$

where the ij^{th} entry of $\bar{\mathbf{L}}_B$, denoted as L_{ij} is

$$L_{ij} = \binom{i}{j} (u^*)^j (1-u^*)^{i-j} \tag{3.26}$$

Similarly, R_{ij}, the ij^{th} entry of $\bar{\mathbf{R}}_B$, can be expressed as

$$R_{ij} = \binom{m-i}{m-j} (u^*)^{i-j} (1-u^*)^{m-j} \tag{3.27}$$

For cubic Bézier curves, the subdivision matrices become

$$\bar{\mathbf{L}}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ (1-u^*) & u^* & 0 & 0 \\ (1-u^*)^2 & 2(1-u^*)u^* & u^{*2} & 0 \\ (1-u^*)^3 & 3(1-u^*)^2(u^*) & 3(1-u^*)(u^*)^2 & (u^*)^3 \end{bmatrix} \tag{3.28}$$

and

$$\bar{\mathbf{R}}_B = \begin{bmatrix} (1-u^*)^3 & 3(1-u^*)^2 u^* & 3(1-u^*)(u^*)^2 & (u^*)^3 \\ 0 & (1-u^*)^2 & 2(1-u^*)u^* & (u^*)^2 \\ 0 & 0 & (1-u^*) & u^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.29}$$

For the special case of midpoint subdivision ($u^* = \frac{1}{2}$), the cubic *Bézier subdivision matrices* $\bar{\mathbf{L}}_B$ and $\bar{\mathbf{R}}_B$ reduce to

$$\bar{\mathbf{L}}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{bmatrix} \tag{3.30}$$

and

$$\bar{\mathbf{R}}_B = \begin{bmatrix} \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.31}$$

3.4. Subdivision of Beta-spline Curves

In section 3.1 it was shown that discovery of the reparametrization matrices $\bar{\mathbf{L}}$ and $\bar{\mathbf{R}}$ is sufficient to subdivide polynomial spline curves. Thus, to subdivide a Beta-spline curve we seek the matrices $\bar{\mathbf{L}}_\beta$ and $\bar{\mathbf{R}}_\beta$, such that

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u^*u) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{L}}_\beta \tag{3.32}$$

and

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u^*+(1-u^*)u) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{R}}_\beta \tag{3.32}$$

where $\bar{\mathbf{b}}$ represents a row vector of the Beta-spline basis functions parametrized as indicated. We now set out to derive the Beta-spline subdivision matrices $\bar{\mathbf{L}}_\beta$ and $\bar{\mathbf{R}}_\beta$.

Since the Beta-spline basis functions span the space of cubic polynomials, we can write the cubic Bézier basis functions as a linear combination of them. Thus, there exists a matrix $\bar{\mathbf{T}}$ such that

$$\bar{\mathbf{B}}(u) = \mathbf{b}(\beta_1,\beta_2;u)\bar{\mathbf{T}} \tag{3.33}$$

Now, from equation (3.14), the matrix $\bar{\mathbf{L}}_B$ reparametrizes the Bézier basis functions, as given by

$$\bar{\mathbf{B}}(u^*u) = \bar{\mathbf{B}}(u)\bar{\mathbf{L}}_B \tag{3.34}$$

Substituting (3.34) twice into (3.33), parametrized once in terms of u, and once in terms of u^*u, gives

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u^*u)\bar{\mathbf{T}} = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{T}}\bar{\mathbf{L}}_B \tag{3.35}$$

Multiplying both sides of (3.35) on the right by $\bar{\mathbf{T}}^{-1}$ yields

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u^*u) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{T}}\bar{\mathbf{L}}_B\bar{\mathbf{T}}^{-1} \tag{3.36}$$

Therefore, we can identify $\bar{L}_\beta$ as

$$\bar{L}_\beta = \bar{T}\bar{L}_B\bar{T}^{-1} \tag{3.37}$$

A similar procedure can be used to show that

$$\bar{R}_\beta = \bar{T}\bar{R}_B\bar{T}^{-1} \tag{3.38}$$

All that remains to be done is the determination of the matrix $\bar{T}$ that maps the Beta-spline basis functions into the cubic Bézier functions. This determination can be done rather easily by writing the both sets of basis functions as cubic polynomials, which in matrix form become

$$\bar{b}(\beta_1,\beta_2;u) = [1\ u\ u^2\ u^3]\bar{C}(\beta_1,\beta_2) \tag{3.39}$$

$$\bar{B}(u) = [1\ u\ u^2\ u^3]\bar{D} \tag{3.40}$$

where

$$\bar{C}(\beta_1,\beta_2) = \begin{bmatrix} c_{0,-2}(\beta_1,\beta_2) & c_{0,-1}(\beta_1,\beta_2) & c_{0,0}(\beta_1,\beta_2) & c_{0,1}(\beta_1,\beta_2) \\ c_{1,-2}(\beta_1,\beta_2) & c_{1,-1}(\beta_1,\beta_2) & c_{1,0}(\beta_1,\beta_2) & c_{1,1}(\beta_1,\beta_2) \\ c_{2,-2}(\beta_1,\beta_2) & c_{2,-1}(\beta_1,\beta_2) & c_{2,0}(\beta_1,\beta_2) & c_{2,1}(\beta_1,\beta_2) \\ c_{3,-2}(\beta_1,\beta_2) & c_{3,-1}(\beta_1,\beta_2) & c_{3,0}(\beta_1,\beta_2) & c_{3,1}(\beta_1,\beta_2) \end{bmatrix} \tag{3.41}$$

$$= \frac{1}{\delta}\begin{bmatrix} 2\beta_1^3 & (4\beta_1^2+4\beta_1+\beta_2) & 2 & 0 \\ -6\beta_1^3 & 6\beta_1(\beta_1^2-1) & 6\beta_1 & 0 \\ 6\beta_1^3 & 3(-2\beta_1^3-2\beta_1^2-\beta_2) & 3(2\beta_1^2+\beta_2) & 0 \\ -2\beta_1^3 & 2(\beta_1^3+\beta_1^2+\beta_1+\beta_2) & -2(\beta_1^2+\beta_1+\beta_2+1) & 2 \end{bmatrix}$$

and

$$\bar{D} = \begin{bmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{bmatrix} \tag{3.42}$$

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

Substitution of (3.39) and (3.40) into (3.33), followed by solution for $\overline{\mathbf{T}}$, yields

$$\overline{\mathbf{T}} = \overline{\mathbf{C}}^{-1}(\beta_1,\beta_2)\overline{\mathbf{D}} \tag{3.43}$$

Thus, the Beta-spline subdivision matrices can be written as

$$\overline{\mathbf{L}}_\beta = \overline{\mathbf{C}}^{-1}(\beta_1,\beta_2)\overline{\mathbf{D}}\overline{\mathbf{L}}_B\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}(\beta_1,\beta_2) \tag{3.44}$$

and

$$\overline{\mathbf{R}}_\beta = \overline{\mathbf{C}}^{-1}(\beta_1,\beta_2)\overline{\mathbf{D}}\overline{\mathbf{R}}_B\overline{\mathbf{D}}^{-1}\overline{\mathbf{C}}(\beta_1,\beta_2) \tag{3.45}$$

The subdivision matrices can be used to subdivide a Beta-spline control polygon $\overline{\mathbf{V}}$ into left and right subpolygons according to

$$\overline{\mathbf{V}}^L = \overline{\mathbf{L}}_\beta\overline{\mathbf{V}}$$

$$\overline{\mathbf{V}}^R = \overline{\mathbf{R}}_\beta\overline{\mathbf{V}}$$

Using Vaxima,[8,13] an algebraic manipulation system, explicit algebraic expressions have been derived for the components of $\overline{\mathbf{L}}_\beta$ and $\overline{\mathbf{R}}_\beta$ for the special case of midpoint subdivision. The results are listed in component form in Appendix 2.

3.5. Converting Beta-spline Curves to Cubic Bézier Curves

Given a Beta-spline control polygon $\overline{\mathbf{V}}$, we wish to find the equivalent Bézier control polygon $\overline{\mathbf{W}}$. That is, $\overline{\mathbf{W}}$ must satisfy

$$\overline{\mathbf{B}}(u)\overline{\mathbf{W}} = \overline{\mathbf{b}}(\beta_1,\beta_2;u)\overline{\mathbf{V}} \tag{3.46}$$

Substitution of (3.33) into (3.46) yields

$$\overline{\mathbf{b}}(\beta_1,\beta_2;u)\overline{\mathbf{T}}\overline{\mathbf{W}} = \overline{\mathbf{b}}(\beta_1,\beta_2;u)\overline{\mathbf{V}} \tag{3.47}$$

Thus,

$$\overline{\mathbf{W}} = \overline{\mathbf{T}}^{-1}\overline{\mathbf{V}} \tag{3.48}$$

In words, $\overline{\mathbf{W}}$ when blended with the Bézier basis, generates the same curve as the control polygon $\overline{\mathbf{V}}$ when blended with the Beta-spline basis set. So, given a Beta-spline control polygon and values for β_1 and β_2, an equivalent Bézier control polygon can be constructed using (3.48).

4. Subdivision of Tensor Product Surfaces

4.1. Mathematical Derivation

Let $\mathbf{S}_{i,j}[a,b;c,d](u,v)$ denote a spline surface constructed from a set of basis functions $W_i(u)$ when $u\varepsilon[a,b]$ and $v\varepsilon[c,d]$, i.e.

$$\mathbf{S}[a,b;c,d](u,v) = \sum_{r=0}^{m}\sum_{s=0}^{n}\mathbf{V}_{r,s}W_r(u)W_s(v); \quad a\le u\le b, \ c\le v\le d \tag{4.1}$$

When no brackets are shown, as in $\mathbf{S}(u,v)$, $\mathbf{S}[0,1;0,1](u,v)$ is implied.

Surfaces of the form (4.1) are called *tensor product* surfaces. Beta-spline surfaces, as described in section 2.2, are examples of such surfaces.

When a spline surface is to be subdivided, the subdivision may split the surface along either or both parametric directions. To split the surface along the u direction at the parametric value u^*, the control graphs $<\mathbf{W}^{LU}>$ and $<\mathbf{W}^{RU}>$ must be found so that their corresponding surfaces $\mathbf{S}^{LU}(u,v)$ and $\mathbf{S}^{RU}(u,v)$ satisfy

$$\mathbf{S}[0,1;0,1](u^*u,v) = \mathbf{S}^{LU}[0,1;0,1](u,v) \tag{4.2}$$

and

$$\mathbf{S}[0,1;0,1](u^*+(1-u^*)u,v) = \mathbf{S}^{RU}[0,1;0,1](u,v) \tag{4.3}$$

A similar set of constraints define the surfaces $\mathbf{S}^{LV}(u,v)$ and $\mathbf{S}^{RV}(u,v)$ produced by splitting $\mathbf{S}(u,v)$ along the v direction. The superscript LU is meant to suggest the "left" part of the u direction, and RU is meant to be mnemonic for the "right" part of the u direction. Similar mnemonics apply for the superscripts LV and RV. The surface can be subdivided in both directions by dividing first say in the u direction, then subdividing $\mathbf{S}^{LU}(u,v)$ and $\mathbf{S}^{RU}(u,v)$ in the v direction. The surfaces thus obtained are denoted by $\mathbf{S}^{LULV}$, $\mathbf{S}^{LURV}$, $\mathbf{S}^{RULV}$, and $\mathbf{S}^{RURV}$. It is relatively easy to show that the choice of which direction to subdivide in first is arbitrary. That is, $\mathbf{S}^{LURV}$ is the same surface as $\mathbf{S}^{RVLU}$.

Just as for curves, the control graphs produced by the subdivision are guaranteed to converge to the surface.[14] If a graph is deemed flat to within a given tolerance, a small number of polygons will adequately characterize it. One might at first be tempted to approximate a "flat" control graph by a single four-sided polygon using the vertices at the extreme corners of the graph. This procedure is

usually inadequate since the four points do not necessarily lie in the same plane. For this and other reasons, to be discussed in section , we use four triangles to approximate a flat control graph.

The decision of which direction (if any) the surface is to be subdivided is determined by the flatness of the graph. It often happens that a graph is highly curved along say the u direction, but relatively flat in the v direction. In this situation it is sufficient to subdivide only in the u direction, to subdivide in both directions would be unnecessary.

A pseudo-encoding of the surface subdivision algorithm is

```
Approximate_Surface( <V>, ε)
parameters
   <V>: Control_Graph;
   ε: Flatness;
local_variables
   <V^LV>,<V^RV>,<V^LU>,<V^RU>: Control_Graph;
begin
   if <V> is within ε of flatness in the u direction then
        if <V> is within ε of flatness in the v direction then
              Output four triangles to approximate <V>
        else
              Subdivide_Along_V(<V>, <V^LV>, <V^RV>)
              Approximate_Surface( <V^LV>, ε)
              Approximate_Surface( <V^RV>, ε)
        endif
   else
      Subdivide_Along_U(<V>, <V^LU>, <V^RU>)
      Approximate_Surface( <V^LU>, ε)
      Approximate_Surface( <V^RU>, ε)
   endif
end
```

The routines Subdivide_Along_U and Subdivide_Along_V perform a midpoint subdivision of the first argument, placing the left sub-graph in the second argument, the right sub-graph in the third.

Although it is possible to directly subdivide Beta-spline control graphs, it is more efficient to first convert to a cubic Bézier form, then subdivide the Bézier control graph. Section 4.2 discusses the subdivision of surfaces. In section 4.3, the conversion of a Beta-spline control graph into an equivalent Bézier control graph is presented.

4.2. Subdivision of Cubic Bézier and Beta-spline Surfaces

The constraint equation (4.2) for subdividing a patch into the LU subpatch can be written in matrix form for a Beta-spline surface as

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u^*u)\bar{\mathbf{V}}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}^{LU}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) \tag{4.4}$$

where

$$\bar{\mathbf{V}} = \begin{bmatrix} \mathbf{V}_{-2,-2} & \mathbf{V}_{-1,-2} & \mathbf{V}_{0,-2} & \mathbf{V}_{1,-2} \\ \mathbf{V}_{-2,-1} & \mathbf{V}_{-1,-1} & \mathbf{V}_{0,-1} & \mathbf{V}_{1,-1} \\ \mathbf{V}_{-2,0} & \mathbf{V}_{-1,0} & \mathbf{V}_{0,0} & \mathbf{V}_{1,0} \\ \mathbf{V}_{-2,1} & \mathbf{V}_{-1,1} & \mathbf{V}_{0,1} & \mathbf{V}_{1,1} \end{bmatrix} \tag{4.5}$$

and

$$\bar{\mathbf{V}}^{LU} = \begin{bmatrix} \mathbf{V}^{LU}_{-2,-2} & \mathbf{V}^{LU}_{-1,-2} & \mathbf{V}^{LU}_{0,-2} & \mathbf{V}^{LU}_{1,-2} \\ \mathbf{V}^{LU}_{-2,-1} & \mathbf{V}^{LU}_{-1,-1} & \mathbf{V}^{LU}_{0,-1} & \mathbf{V}^{LU}_{1,-1} \\ \mathbf{V}^{LU}_{-2,0} & \mathbf{V}^{LU}_{-1,0} & \mathbf{V}^{LU}_{0,0} & \mathbf{V}^{LU}_{1,0} \\ \mathbf{V}^{LU}_{-2,1} & \mathbf{V}^{LU}_{-1,1} & \mathbf{V}^{LU}_{0,1} & \mathbf{V}^{LU}_{1,1} \end{bmatrix} \tag{4.6}$$

Since the Beta-spline subdivision matrix $\bar{\mathbf{L}}_\beta$, derived in section is a matrix that reparametrizes the Beta-spline basis functions, we can substitute (3.32) into (4.4) to get

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{L}}_\beta\bar{\mathbf{V}}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}^{LU}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v)$$

Thus, we can identify $\bar{\mathbf{V}}^{LU}$ as

$$\bar{\mathbf{V}}^{LU} = \bar{\mathbf{L}}_\beta\bar{\mathbf{V}} \tag{4.7}$$

In a similar fashion, it can be shown that

$$\bar{\mathbf{V}}^{RU} = \bar{\mathbf{R}}_\beta\bar{\mathbf{V}} \tag{4.8}$$

A subdivision in the v direction can be developed by considering the v subdivision constraint, which for the left supatch of a Beta-spline is

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v^*v) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}^{LV}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) \tag{4.9}$$

But

$$\bar{\mathbf{b}}^T(\beta_1,\beta_2;v^*v) = (\bar{\mathbf{b}}(\beta_1,\beta_2;v)\bar{\mathbf{L}}_\beta)^T = \bar{\mathbf{L}}_\beta^T\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) \tag{4.10}$$

so that (4.9) becomes

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}\bar{\mathbf{L}}_\beta^t\bar{\mathbf{b}}^T(\beta_1,\beta_2;v) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}^{LV}\bar{\mathbf{b}}^T(\beta_1,\beta_2;v)$$

Thus we can identify $\bar{\mathbf{V}}^{LV}$ as

$$\bar{\mathbf{V}}^{LV} = \bar{\mathbf{V}}\bar{\mathbf{L}}_\beta^T \tag{4.11}$$

A similar analysis shows that

$$\bar{\mathbf{V}}^{RV} = \bar{\mathbf{V}}\bar{\mathbf{R}}_\beta^T \tag{4.12}$$

To subdivide a Bézier control graph, simply replace $\bar{\mathbf{L}}_\beta$ with $\bar{\mathbf{L}}_B$, and $\bar{\mathbf{R}}_\beta$ with $\bar{\mathbf{R}}_B$ in equations (4.7), (4.8), (4.11), and (4.12).

To reiterate the results of this section, subdivision of a Beta-spline patch in the u direction is accomplished by applying the subdivision matrices $\bar{\mathbf{L}}_\beta$ or $\bar{\mathbf{R}}_\beta$ to the left of the matrix describing the control graph. To subdivide a surface in the v direction, apply the transpose of the subdivision matrices to the right of the control graph matrix.

4.3. Converting Beta-spline Surfaces to Cubic Bézier Surfaces

Given the Beta-spline control graph $\bar{\mathbf{V}}$, we wish to determine the Bézier control graph $\bar{\mathbf{W}}$ such that

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}\mathbf{b}^T(\beta_1,\beta_2;v) = \bar{\mathbf{B}}(u)\bar{\mathbf{W}}\bar{\mathbf{B}}^T(v) \tag{4.13}$$

Two substitutions of (3.33) into (4.13), once in terms of u and once in terms of v, gives

$$\bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{V}}\mathbf{b}^T(\beta_1,\beta_2;v) = \bar{\mathbf{b}}(\beta_1,\beta_2;u)\bar{\mathbf{T}}\bar{\mathbf{W}}\bar{\mathbf{T}}^T\bar{\mathbf{b}}^T(\beta_1,\beta_2;u) \tag{4.14}$$

Thus,

$$\bar{\mathbf{W}} = \bar{\mathbf{T}}^{-1}\bar{\mathbf{V}}\bar{\mathbf{T}}^{-T} \tag{4.15}$$

5. Stopping Criterion

In the previous sections, we developed a recursive method for subdividing spline curves into smaller and smaller pieces. We wish to approximate these pieces with line segments when they become "close" enough to linear. Closeness will be

measured by some error metric. A user specified tolerance provides a threshold for the metric. The error metric and its associated tolerance provide the stopping criterion for the subdivision. Spline *surfaces* are approximated using a similar criterion with polygons instead of line segments. We refer to line segments and polygons as *approximating primitives*. Other approximating primitives are possible but we only consider linear ones because of the resulting efficiencies.

Subdivision is performed locally, so that each curve can be subdivided independently of its neighbors. For a given curve, the endpoints of its approximating line segment are equal to the endpoints of the curve. In this way, the line segments corresponding to neighboring curves remain continuous. A surface is approximated by a quadrilateral defined by the four parametric corners of the surface. Ensuring continuity of the approximating polygons is more complex and *crack prevention* techniques are required.

One could consider more global schemes of approximation where the endpoints of an approximating line segment do not interpolate the endpoints of a subdivided curve. This might help in calculating globally optimal or minimal approximations but the computational efficiences of the local techniques described here seem to be more important.

The precision of a curve approximation is related to how "close" the curve is to the approximating primitives. Areas of greater curvature will require more primitives to provide a given degree of accuracy (Figure 5.1). Calculating the deviation of a curve from a straight line or of a surface from a planar polygon corresponds to measuring the *flatness* of the object. Flatness is a natural geometric criterion for quantifying the precision of our approximations. Use of a flatness measure as the error metric in our stopping criterion will allow us to create approximations of a given precision. In addition, the subdivision cost will only be incurred where it is needed to meet the desired accuracy. However, *the cost of flatness testing can be as great as, or even greater, than that of the actual subdivision calculations*, and thus it is important to reduce the testing costs as much as possible.

We will use the notion of flatness to define our stopping criterion. Intuitively, stopping criteria measure the deviation of an approximating primitive from the object it is approximating. A criterion defines an *acceptable region* about an approximating primitive. The size of this region will vary with the desired accuracy or tolerance of the approximation. We recursively subdivide a curve or surface until it is contained within the acceptable region for the associated approximating primitive. We then take the approximating primitive as representing the object. A flatness criterion measures the *distance* of the object from a linear approximating primitive. Thus, a flatness criterion ensures that a linear approximation stays within a given distance of the true curve or surface.

Calculating whether or not the curve or surface lies within an acceptable region can be quite difficult. These tests can be performed much more easily on the control graph. If the parametric formulation is such that the curve or surface is contained within the convex hull of the control graph, then determining if the control graph is contained within the acceptable region will be a sufficient, although not necessary, condition for the object meeting the criterion. The splines that we will consider satisfy the convex hull property. Since the control graph "surrounds"

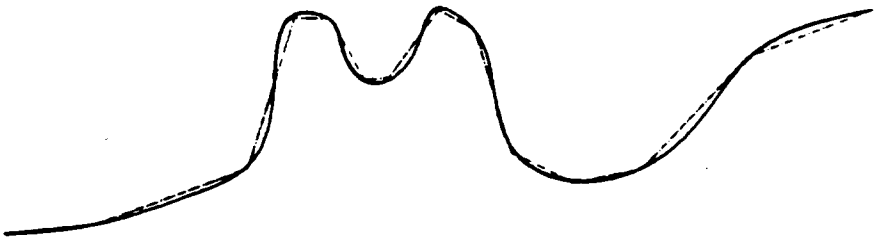


Figure 5.1.
Example of a linear approximation to a curve that demonstrates the relationship of curvature to the approximation.

the object, there may be cases where an unnecessarily large error is calculated. It would be ideal if the control graph could provide a *necessary* and sufficient condition for flatness testing.

For a curve, the acceptable region is defined about the approximating line segment between the endpoints of the curve. In two dimensions, this region is a bar about the line segment with semi-circular ends whose half-width is equal to the tolerance (Figure 5.2). In three dimensions, the acceptable region is a cylinder with hemi-spherical ends. Thus, the flatness criteria for a curve calculate the maximum distance of the vertices of the control polygon from the approximating line segment.

For a surface, the acceptable region is defined about the approximating quadrilateral comprising the corners of the surface. However, this quadrilateral will in general be non-planar. In this case, we define a planar quadrilateral about which to specify the acceptable region. The planar quadrilateral will lie on the plane that most closely matches the non-planar quadrilateral, that is, the plane that minimizes the maximum distance to each of the four corners of the non-planar quadrilateral. This is simply a least squares plane approximation to the non-planar quadrilateral.[11] The corners of the planar quadrilateral are the points on this plane that are closest to the corners of the non-planar quadrilateral. The acceptable region is then defined to be a slab of thickness equal to twice the tolerance, centered about the planar quadrilateral, with half-cylinder sides, and quarter-spherical corners. To avoid the cost of the least squares calculations, an approximation to the planar quadrilateral can be made. The planar quadrilateral can be constructed by choosing three corners of the surface. The fourth corner of the quadrilateral is defined to be the point on the plane, defined by the three chosen points, that is closest to the unchosen corner of the surface. This may give

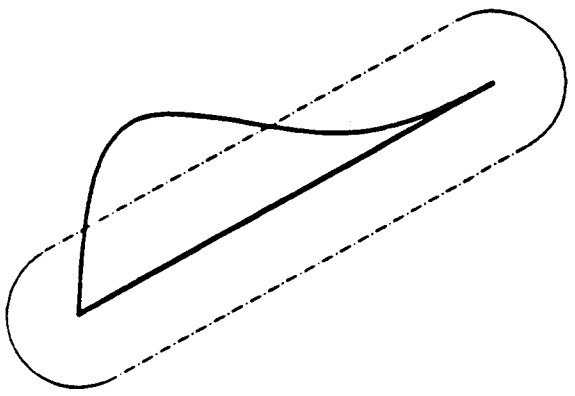


Figure 5.2.
Acceptable region based on flatness for a two dimensional curve approximated by a line segment.

different error values than the ideal region but in practice operates very well.

There is another approximation in evaluating the flatness criterion that can be made to save computational cost. The error metric evaluation consists of testing whether vertices are contained within the acceptable region. It is less costly, but still generally effective, to simply calculate the distance of a vertex from the primitive without considering the boundaries of the primitive. For curves, this corresponds to testing only the distance of vertices from the *line* containing the approximating segment. Similarly for a surface, the distance between vertices and the *plane* defined by three chosen points is measured. In certain cases these approximations will produce errors.

Another natural measure for a stopping criterion is based on length for curves and area for surfaces. For example, the difference in length between a curve and its approximating line segment could be an error metric. However, this measure is not as uniform as a distance criterion. The length of a curve may lie completely on one side of the line segment, thus giving a large deviation, or it may lie equally on both sides, giving a smaller deviation (Figure 5.3). A given tolerance will allow a wide range of deviations from the approximating primitive. This can create obvious visual discrepancies. Thus, subdivision is not controlled as well as with a distance criterion. It is also no cheaper computationally and thus has no advantages over a distance metric.

The flatness tests defined so far are applied to an entire surface. A test of this type is referred to as a *global test*. However, surface flatness can be measured with respect to the two different directions corresponding to the parameters u and v. The surface may be curved only in a single direction and flat in the other, e.g. a parametric cylinder. Thus, the test for a stopping criterion should be directionally selective to prevent unnecessarily subdividing a surface in a flat parametric

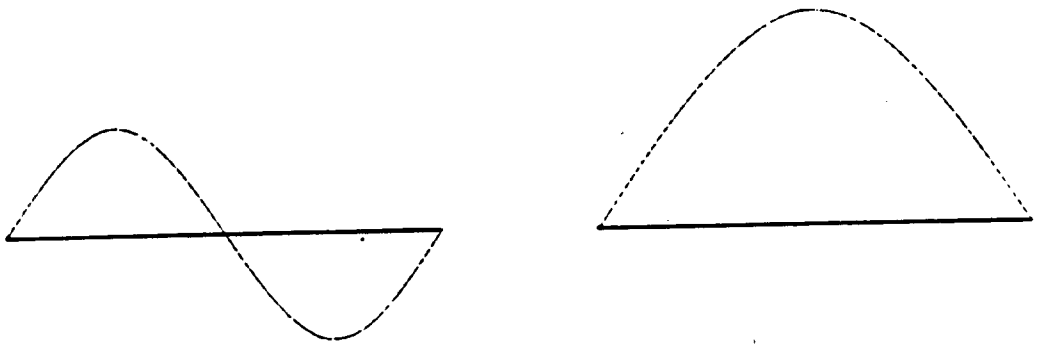


Figure 5.3.
A stopping criteria based on distance will allow the same approximation to be used for curves of equal length that have unequal deviations from the approximating line segment.

direction. We call such a test a *directional test.*

The control graph of a spline surface patch contains four control polygons in both u and v. To measure the flatness along a given parametric direction, the directional test checks the control polygons in that direction. If all of them are flat, then it is not necessary to subdivide across that parameter (subdividing along a single parameter is described in a previous section). When the surface is flat in both directions, we additionally verify that the surface as a whole is flat using the global test. Even if a patch is flat in both parameters the surface may not be. For instance, a twisted patch can be constructed from flat control polygons (Figure 5.4).

The tolerance for the subdivision is a user-supplied value reflecting the accuracy required of the approximation. The tolerance may be a *geometric* tolerance such as when machining a part. This is the meaning of the tolerance we have been discussing so far. However, for purposes of display a *visual* tolerance is what is wanted. In a normal perspective rendering of an object, the size of the object in the image depends on its distance from the viewer. The farther away it is, the larger the geometric errors that can be visually tolerated. Thus, given a visual tolerance we can multiply it by the distance of the object to give the appropriate geometric tolerance. An object's visual tolerance may vary across the surface corresponding to differences in depth. Our approach allows the tolerance to *adapt* in this way. Again, care must be taken to prevent cracks between neighboring approximating polygons.

Once it has been determined that an object is not flat and therefore must be subdivided, the particular parametric point at which to subdivide must be chosen. The parametric point that is selected can greatly influence the effectiveness of the subdivision and hence the amount of subdivision performed. Since there exists an

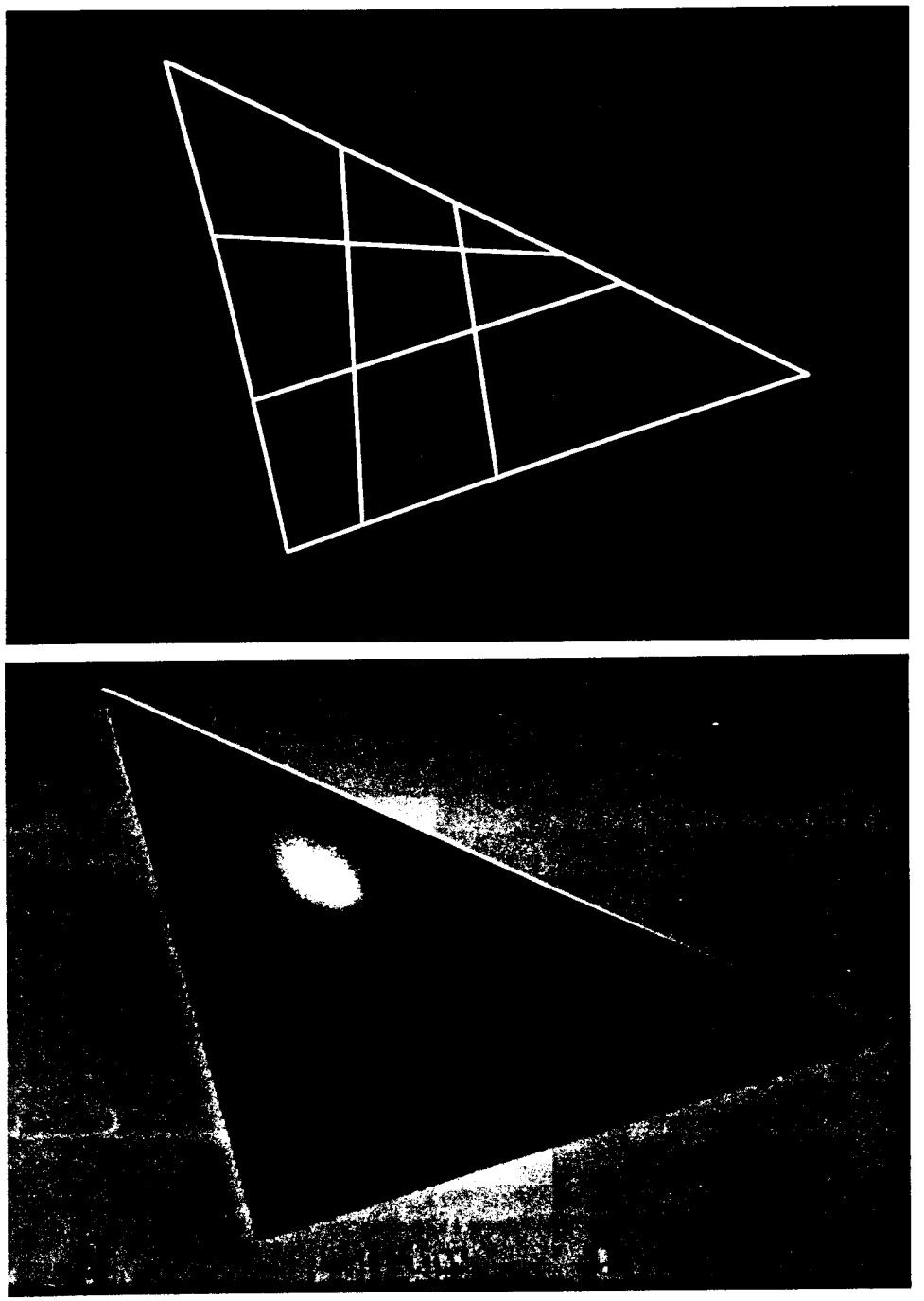


Figure 5.4.

Flat control polygons can be used to construct a twisted patch that is not flat. Both the control polygon and rendered surface corresponding to the twisted patch are shown.

extremely fast method for subdividing at the parametric midpoint, that is where we subdivide. In general, this will not produce an optimal approximation in terms of the cost of the subdivision or the number of output polygons. We know how to subdivide spline curves at arbitrary parametric values (see previous section), and are investigating ways to select better subdivision points.

Several types of stopping criteria have been used in the past. Catmull[9] subdivided until the patches were of pixel size. This is a visually based criterion but it ignores the properties of the surface that is being approximated. This method will generally subdivide unnecessarily since pixels do not measure flatness. However, there was no problem with cracks because the subdivision was carried out to the pixel level. Nydegger[19] used the difference in normals at the corners of a patch to determine when to stop. Such a criterion can produce good results, but it is a very local measure since the normals at the corner of a patch may not necessarily reflect the flatness, or lack of it, of the entire patch. Nydegger also saved a tree representing the subdivision and fixed any resulting cracks. Clark[10] based his criterion on what he called "curvature in parametric space" or bilinearity (the linearity of all of the geometric values in their parameters). The actual criterion was based on the magnitude of the parametric second derivative. The problem is that the bilinearity measure is not directly related to the geometric properties of the surface. Lack of "parametric curvature" in no way guarantees geometric flatness so that the resulting linear approximation may be poor. Clark's method did prevent cracks from appearing in the subdivision. The stopping criterion we use is similar to that of Lane and Carpenter;[17] however, they did not address the issue of preventing cracks and thus were unable to take full advantage of adaptively subdividing based on flatness. Instead, they were sometimes forced to subdivide to the pixel level, bypassing the savings of adaptive subdivision.

We have integrated stopping criteria based on flatness, directionally sensitive testing, crack prevention, and adaptive tolerances, allowing us to produce accurate approximations while avoiding unnecessary costs. We have also reduced the number of tests that are performed.

6. Crack Prevention

Ensuring continuity of the polygons that approximate a surface has difficulties not appearing with curves. If the subdivision is performed to the same level of recursion for each patch, or if the subdivision is carried out to the "pixel level" during display, then cracks do not appear between adjacent polygons. However, when the resultant polygons are larger than a pixel and the subdivision is based on a local criterion like flatness, a given patch may be subdivided more than a neighboring patch, creating the possibility of cracks (Figure 6.1). Subdividing using directional tests compounds the problem. Any algorithm performing adaptive subdivision of surfaces should deal with the crack problem.

In a sense, our adaptive subdivision algorithm is generating a quadrilateral mesh approximating a surface. The mesh points are recursively output four at a time, corresponding to the four corner points of the flat patch being approximated. Associated with the quadrilateral are four line segments corresponding to the four flat boundaries of the patch. Any mesh points that are generated later in the subdivision and that lie on such a boundary must also lie on the corresponding line segment. If they do not, cracks will appear along the boundary.

To prevent cracks, neighboring patches must determine that their *shared boundary* becomes flat at the *same* recursion level. They can then use the *same*

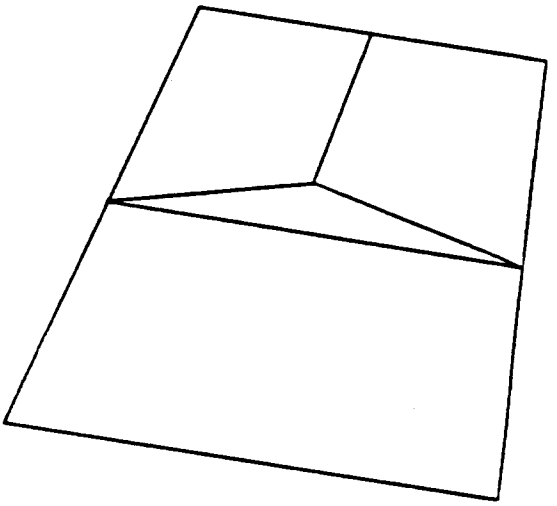
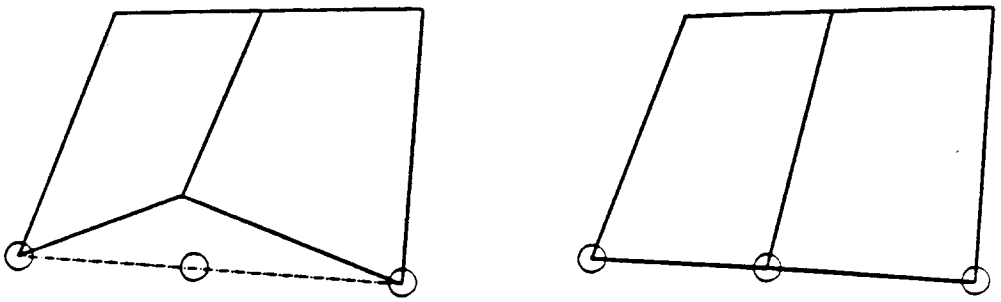


Figure 6.1.
Unequal subdivision of neighboring patches can cause cracks to occur in the polygonal approximation.

line segment to represent that boundary. This line segment is also passed down to any subpatches sharing the boundary. When such a subpatch is to be added to the mesh, the flat boundary segment is used to "correct" the approximating quadrilateral so that no cracks appear. A *boundary test* is used to provide this control. It is only performed on boundary control polygons and ensures that patches sharing a boundary will determine that boundary to be flat at the same time.

Information that is passed down about flat boundaries during the recursive subdivision must be kept with the patch. When approximating a patch, we choose its corner points to define the approximating quadrilateral, except when there is flat boundary information. In this case, we choose points that lie on the associated line segment rather than the patch corner points. Thus, the additional information *overrides* the patch corner points (Figure 6.2). We refer to flat boundary information that is passed down as the *corner information* since it is used to correct the corner points of a patch on output.

Note the important property that corner information will be set at the first detection of flatness of a boundary. Corner information, once set, cannot be changed. It can be demonstrated that this method prevents cracks. There are two cases to consider. The first is that of two surfaces who share a border. The corner information for the common boundary will be set at the same recursion level and to the same value. The neighbors will also pass down these values to their children so that neighboring descendants will have corner information appropriate to prevent cracks along common boundaries. The second case is that of neighbors that share a corner but not a boundary. If the corner lies on the interior of a flat



(a)
polygons for neighboring patches, flat border of other polygon shown by dashed line, circles represent corner information

(b)
corner information used to prevent crack

Figure 6.2.
Cracks in the polygonal approximation can be prevented by the use of corner information.

boundary segment, cracks will be prevented as before. If the corner does not lie on the interior of a flat boundary segment, the output point corresponding to the corner will be the actual surface point. Thus both patches will use the same value and there will be no crack. A rigorous proof of the crack prevention property of our algorithm can be constructed inductively from the above arguments.

Some researchers have suggested that cracks can be alleviated by making the subdivision stopping tolerance small enough to prevent the cracks from being visible when rendered.[17] Besides being an extremely ad hoc criterion, such a method defeats part of the gains of adaptive subdivision since it may force more subdivision than is really necessary. The approach we have taken is to modify the subdivision process to prevent cracks from occurring. This approach has also been taken by Nydegger[19] and Clark.[10]

Nydegger constructed a subdivision tree representing the *entire* state of the recursive subdivision. In general, this can have a prohibitive cost. There were two methods considered for fixing cracks that had been generated. The first was to simply fill the cracks with polygons. This technique is clearly unacceptable as the crack filling polygon may cause a singularity; e.g. it may be perpendicular to its neighbors. Nydegger too discarded this technique. The second method he used was to reexamine and modify the subdivision tree by adding vertices to fill the cracks in a reasonable manner. This second method yielded much better results but at a high cost; half of the subdivision tree may have to be reexamined and modified. Generally the technique is very space inefficient with high overhead in the additional tree traversal. Nydegger estimates a 50% time increase in cost for crack

fixing.

The crack prevention method of Clark is similar to ours. He did not construct a subdivision tree but modified the subdivided surfaces to prevent cracks. He tested each boundary of the control graph and if it was flat, it was approximated by a line segment connecting the corners. Any new control vertices generated from that boundary by subdivision were forced to lie on the approximating line segment. Subdivision continued until the entire control graph was flat, in which case the four corners of the surface were output as a polygon. An important difference from our technique is that Clark forces the border control polygon corners of all subsequent offspring to actually lie on any flat boundary segments. This introduces an error in the subdivision process since the patch definition is actually changed. This creates a less accurate polygonal approximation than our approach. This error will be dependent on the tolerance used. Clark also used bilinearity as the error metric in his criterion. Bilinearity is unreliable since it does not quantify geometric flatness; very curved surfaces may satisfy the criterion and flat surfaces may not.

A valuable aspect of our method is its simplicity and generality. By using a border test and corner information, cracks can be prevented with little additional cost. This technique also supports the use of adaptive tolerances.

6.1. Adaptive Tolerance

When rendering an object, a visual, rather than geometric, tolerance is used. The farther away an object is, the greater the error in the approximation that is visually tolerable. In this case, the user specified tolerance gives the maximum error allowed in the *image* after perspective projection. Thus, the input tolerance is scaled by the minimum depth value of the object to obtain the geometric tolerance for the subdivision. This scaling can be performed once for the entire object, creating a single tolerance used for all of the patches making up the object. However, the visual tolerance can be calculated for each patch, or even at each subdivision level, providing a level of geometric detail which varies across an object. This will allow the tolerance to vary corresponding to the properties of the object in combination with the viewing parameters. When an object's visual extent is large, the geometric tolerances may vary widely across the object. Thus, it is important to adapt the tolerance to reduce the number of output polygons and the amount of work performed, while maintaining the desired precision in the approximation.

When an adaptive visual tolerance is used, additional care must be taken to prevent cracks. The only requirement is that neighboring patches must use the same tolerance on their common borders. This, in conjunction with the border test and corner information, will prevent cracks as before. In the most general case, where the tolerance is varied across subdivision levels, each patch will have five geometric tolerance parameters: one for each of the four borders and one for both the directional and global tests. The border tolerances will be based on the minimum distance from the viewer of the control vertices for each border, and the directional and global tolerance will be the minimum over the entire control graph.

There are simple approximations that can be used to reduce the cost of calculating the adaptive tolerances. The tolerance can be calculated by only considering the corner points of the patch. It can also be calculated for each patch of the object,

rather than for each subdivision level. In general, the additional costs involved in calculating the adaptive tolerances can be a very small percentage of the total subdivision cost.

6.2. Corner Information

There are many ways to create and use the corner information, as well as to pass it to offspring. The simplest one is:

1) whenever a boundary becomes flat, fix the overriding corner information values to be the boundary endpoints;

2) when a patch is subdivided, take any fixed corner information values, find the midpoint of the corresponding line segment, use this as a fixed corner value for the associated offspring, and pass along any other fixed corner values as appropriate.

This is a very simple process which corresponds, in some sense, to subdividing at the parametric midpoint. However, the actual parametric subdivision may correspond more exactly to a point not in the middle of the flat boundary segment. If more exactness is desired, then a closest point calculation should be used. In this case, when a boundary has become flat and a subdivision is performed, the subdivision point along the boundary is calculated and the point on the flat boundary segment closest to it is passed down as a fixed corner value for the offspring. This gives a closest approximation in distance.

7. Testing Methods

The method of flatness testing is critical to generating an accurate approximation while reducing the number of polygons. However, testing is costly and the number of tests performed must be reduced as much as possible. The recursion that results during subdivision creates a corresponding recursion or subdivision tree. We want to minimize the number of nodes in that tree, and at the same time keep the cost per node small. There are several ways in which the number of tests performed can be reduced. Our techniques will incorporate these factors, and thus reduce the *total* cost of performing adaptive subdivision.

The primary strategy is a simple and important one. We order the tests so that the need for performing a subdivision is determined as early as possible.

Initially, we test only the border control polygons. As soon as a border control polygon of a particular parametric direction is found not to be flat, the associated direction is subdivided.

When both borders of a given direction are flat, the interior control polygons of that direction are tested. Again, as soon as an interior control polygon is found that is too curved, the corresponding direction is subdivided.

When all of the control polygons in both parametric directions are flat, we perform a global test. If the patch as a whole is not flat, we alternate subdivision directions rather than subdividing in both directions. Such an object can sometimes require subdivision in both directions, but we have chosen to reduce the number of

output polygons and alternate directions at the risk of performing more tests. Situations that would require subdivision in both directions should be unusual. To improve the selection of the parametric direction in which to subdivide, the maximum error value for each direction can be calculated and the subdivision direction selected by taking the maximum.

Once a given border control polygon is flat, the corresponding control polygon of any subdivided offspring will also be treated as flat. Flags are set to prevent retesting border control polygons that have become flat. Also, when a given parametric direction becomes flat, it is treated as flat for all offspring. Additional flags are used to prevent retesting of the associated interior control polygons. The flags are checked before performing any tests.

7.1. More Complicated Strategies to Reduce Testing Cost

The minimum depth of subdivision *required* by all of the offspring of a given border control polygon for a patch is maintained. This information is shared among neighboring patches so that some of the testing of that border does not have to be redone. If minimum subdivision depth information about a given border is known, then a patch sharing this border *must* be subdivided to the specified recursion level in the respective parametric direction. Thus, no testing on the given border need be done until the minimum subdivision level is reached. Once the minimum number of subdivisions has been carried out, testing must be done on the border, since more than the minimum number of tests may be needed. These savings only apply to a border for which the minimum subdivision depth is known; for example, the opposite border of the associated direction may have to be tested as in the usual case.

The sharing of testing information is straightforward. To prevent cracks, we must know the exact subdivision level at which a border becomes flat; thus, this information can be passed back up the subdivision tree from children to their parents. The parents can then pass the information down to other offspring who share these borders, and up to their parents and so on. The information that is passed up and down the subdivision tree is the *minimum* number of subdivision levels required by each border.

The recursive method for calculating the minimum number of subdivisions required of a border is also simple. A leaf node of the subdivision tree simply returns the number of tests each border required. At a non-leaf recursion level, the minimum for each border is found by taking the minimum of the values returned by all offspring sharing the border. These minimum values are then passed on to any new offspring sharing the border as well as returned to the parent who can then pass them on, etc. This sharing of subdivision information is accomplished without the need of a subdivision tree, thereby avoiding high storage costs.

There is another way in which testing can be reduced. A surface can either be subdivided completely in one direction then the other, or it can be subdivided in both directions simultaneously. These two choices correspond to creating a deep, narrow tree or a shallow, wide tree respectively. If subdivision is performed completely in one parametric direction and then the other, less cost will be incurred when each direction requires about the same level of subdivision, but greater worst

case cost will arise. It also favors symmetrical, equally curved shapes, which is not the expected case. If testing in both directions is performed, then less testing will be incurred when only one parametric direction requires subdivision and unequally curved objects will be favored in general. For these reasons, we choose to check both directions simultaneously, at each subdivision level, until they become flat.

7.2. A Possible Testing Alternative

We have chosen the above methods for flatness testing because they reduce testing cost and the number of output polygons.

If the input surfaces are of a particular type or certain constraints are satisfied, it may be possible to gain performance improvements with a modified testing strategy. One important case occurs when the flatness of a particular direction of a surface is directly reflected by the flatness of the two associated borders. This corresponds to surfaces which are not very bumpy or twisted. Such a testing alternative is:

1) test only the border polygons and subdivide based upon them.

2) when all of the borders are flat, convert to the global test. There are two variations at this point:
   a) subdivide in both parametric directions, or
   b) alternate parametric directions.

This method ignores the interior control polygons with respect to directional testing. If the borders of the patch directly reflect the total curvature of the patch, then this method will provide superior performance since it avoids the directional tests. However, in general, this method will incur greater testing costs because a global test is more expensive then a directional test and is being used in its place. This strategy may also produce a larger number of polygons.

8. The Algorithm

We have discussed the algorithmic aspects of the stopping criterion, crack prevention, adaptive tolerances, and testing strategies. We use *border*, *directional*, and *global* tests that are based on *flatness* to determine when to stop the subdivision. These techniques are *adaptive* to the geometric properties of the surface and the viewing parameters, and the resulting approximation is *without cracks*. This algorithm incorporates our techniques to reduce the number of tests performed and the number of output polygons. We give a brief outline of the algorithm:

> Subdivision proceeds in a depth-first manner. If a visual tolerance is used, the geometric tolerance value is varied during the subdivision. Each recursive call returns the number of subdivisions required of each border. This information is then passed down to neighbors to reduce retesting. We keep the flatness information for each border separate for this purpose. Subdivision of a patch proceeds in both parametric directions simultaneously. Once a border or parametric direction of a surface becomes flat, no testing of the associated characteristic is performed for any offspring.

Initially, only the border control polygons are tested. To prevent cracks, each border must be tested at least once to determine exactly when it becomes flat, even if testing information is shared. When both borders of a given parametric direction are flat, we begin to test the interior control polygons. As soon as we find an interior control polygon that is not flat, we subdivide in the respective direction. Once the interior control polygons in a given direction are all flat, we stop testing that direction. When both directions are flat, the global test is applied. Even if all of the control polygons satisfy the tolerance, the control graph may not be flat (Figure 5.4). When the global test is not satisfied, we subdivide along a single parameter, alternating directions as we go. We save flags indicating when borders and directions become flat to prevent retesting.

Our subdivision stopping criterion has three parts: a border test to determine if a border of a patch is flat, a directional criterion that determines if a patch is flat in a particular parametric direction, and a global criterion that determines if the patch as a whole is flat. We use flatness-based criteria, but the algorithm can actually be used with general criteria and patch formulations similar in spirit to Lane and Carpenter.[16] We only require that neighboring patches calculate the same border criterion value on their common border. We also force the method to be able to directly generate the corner points approximating the surface. Bézier surfaces are an important example of a formulation that satisfies these constraints

The tests that we use are applied to control polygons and control graphs. The border test and directional test that we have described are really one and the same; that is, the border test simply tests the border control polygon and the directional test applies the same test to the interior control polygons in a particular parametric direction. When we spoke of the border or directional test, we were referring to the use of the control polygon flatness test in two different situations. In general, this need not be the case; that is, the two tests may be different. It is also possible to allow distinct tolerances for the different tests. This is what is done to provide an adaptive tolerance for rendering.

Given another parametric formulation with border, directional, and global stopping criteria, and a subdivision method that could directly generate the corner values, we can easily integrate our testing reduction, crack prevention, and adaptive tolerance techniques.

9. Cost

There are some definite costs attributable to adaptive subdivision and crack prevention. We will examine the space and time costs separately. We do this in the framework of subdividing bicubic spline surfaces using all of our techniques. We will also try to make some comparisons with a subdivision algorithm that only performs global testing. Such an algorithm subdivides in both parametric directions while a surface is not flat, and does not prevent cracks, use adaptive tolerances or reduce testing. This will serve to indicate the savings over methods not containing these techniques. Refer to the previous sections for an outline of our algorithm.

9.1. Space

Our analysis is based on a unit of storage corresponding to a single floating point value.

To reduce the number of tests performed, there is additional space required. The algorithm must remember when a border or interior control polygon becomes flat to prevent retesting it. The depth of testing required by each border is kept to be returned to the parent for sharing purposes. Shared testing information that has been passed down must also be retained. Finally, a flag is kept to denote which parameter to subdivide along when the border and interior control polygons are flat but the patch as a whole is not. We list this additional information with the number of such values in brackets:

1) flag denoting whether or not border control polygons are flat [4]

2) flag denoting whether or not directions are flat [2]

3) depth of testing performed for each border [4]

4) minimum subdivision level for each border passed from neighbors [4]

5) current parametric direction in which to subdivide when doing global subdivision [1].

This can be thought of as simply being part of the description of a patch. A bicubic spline patch in three dimensions requires 16 numbers for each of its dimensions incurring a total of 48 units of storage. The additional storage we require is the sum of the bracketed values above and is equal to 15 units. This represents about a 30% increase in storage. We have counted all of the additional information as equivalent but all of the information is integer rather than real data and can generally be stored in smaller units. As the order of the patches increases, the percentage of additional storage required drops.

The only other additional space is that for the corner information. The amount of storage used depends entirely on the surface characteristics. If, at the start, a border is flat in a given parametric direction, corner information can be set immediately at the beginning of the subdivision recursion. However, the best case is that no corner information is calculated. This will occur if a patch becomes flat "all of a sudden" that is, all of the borders and the entire patch become flat at the same recursion level. This would require almost no additional space. For example, assuming that the corner information is represented by a pointer which can be *NIL*, then in the best case we will use an "address" for each node of the subdivision tree. If we use the same assumptions as above, we require one additional unit of storage versus 48 units, making up about a 2% increase in space usage for corner information.

The worst case space usage occurs when every patch has corner information for all of its boundaries at all subdivision levels. This is not an average case; it means that every patch has flat borders and is twisted about two diagonally opposite corners and/or is puffy in the center like a down blanket. There are four

corner points to be determined, and each one requires three values for x, y, and z; thus, in the worst case 12 units of storage are used for the corner information. Using the previous assumptions, this represents a worst case increase in space usage of 25% for corner information.

Thus, assuming worst case conditions for corner information, we incur a total storage increase of about 31% + 25% = 56% per node of the subdivision tree. An important point to remember is that we *do not require the entire subdivision tree* to produce test savings or to prevent cracks. The space for both the control graph and the corner information can be freed immediately upon reaching the last subdivision level. At any instant, only storage for a single path from the root of the subdivision tree to a leaf is required. In fact, a method that keeps the entire tree, like Nydegger's, would require *exponentially* greater space. Thus the amount of space being used at one time is a small fraction of the space representing the entire subdivision tree. When performed in a depth first manner, space usage due to recursion is not likely to ever be a problem. However, the algorithm does not prevent one from saving the recursion tree and incurring the storage cost if desired.

9.2. Time

The worst case asymptotic order of the adaptive subdivision algorithm is unchanged from a purely global subdivision criterion. This worst case behavior occurs with a surface that must be subdivided equally in both directions. Since performing the border and directional tests may be more costly than a single global test (see Appendix 1), there may be additional constant cost from directional testing. However, the best case performance of the adaptive method, when a surface needs to be subdivided in only one parametric direction, will produce the *square root of the cost* of the purely global subdivision. Generally, surfaces will be unequally curved so that this is an extremely important factor for an efficient subdivision algorithm. The order of our algorithm is clearly unaffected by the crack prevention technique as there is simply a small additional constant cost at each subdivision for handling corner information.

9.2.1. Testing

In our algorithm, the subdivision criterion must be locally tested. Since there are eight control polygons defining a patch (four in each direction), the control polygon tests may have to be applied eight times, rather than a single global test for the whole patch. However, since we have structured the flatness testing so that our algorithm remembers when a border control polygon is flat, we no longer test that border for all of its offspring. Similarly, we prevent retesting of flat parametric directions. The testing cost may actually be substantially reduced by the use of local tests since the cost of the flatness test for a single control polygon is less than that of the global flatness test. For instance, if one border control polygon is initially flat while the other only becomes flat after many levels of subdivision, we prevent retesting the first polygon for all subdivisions levels except the first.

When a surface is curved in only one direction, best case behavior is exhibited by our algorithm. With only global testing, we would be subdividing unnecessarily

in a flat direction. Thus, we would be increasing the number of tests performed as well as the number of subdivisions. In this case, directionally sensitive testing is an overwhelming improvement. If the object requires n levels of subdivision, our adaptive method would produce a 2-ary tree of depth n while the global method would produce a 4-ary tree of the same depth. Thus, the number of nodes in the subdivision tree is 2^n for our method versus $4^n = \left[2^n\right]^2$ for the global one. Global testing will incur the *square* of our cost and will generate an equally greater amount of data than directional testing does.

When a surface is equally curved in both directions, at each level of the subdivision there will be four offspring. This produces the worst case behavior of our algorithm when compared with a purely global method. Testing both parametric directions will generally be equivalent to a single global test since rarely will all eight control polygons be checked. Thus, the real increase in cost results from the global flatness test performed at the deepest recursion level. We must apply a final global test since the error over the entire patch can be the sum of the directional errors. However, since the leaves of the recursion tree may represent about 75% of the nodes of the tree (since we have a 4-ary tree) and we are doubling their cost by performing both the directional and global tests, this corresponds to a 75% increase in the number of tests performed.

Even though in the worst case we can get a 75% increase in the number of tests performed over a purely global method, we expect the average case behavior to be substantially better. In the best case, we perform the square root of the number of tests performed by a global method. The reduction in the number of tests performed in the average case is important.

Directionally sensitive subdivision is a crucial part of data reduction. If directionally sensitive subdivision is to be performed at all, parametric polygons must be tested for flatness and thus control polygon tests cannot be avoided. Since cylindrical shapes are common and they produce best case results, the use of this technique can have tremendous benefits, overriding any additional worst case testing costs.

Another important method for reducing execution cost and data is the adaptive visual tolerance. The time cost of calculating the tolerance can be very small and may have a large effect. For instance, if the object's visible regions vary in distance from the viewer by a factor of two, then the geometric tolerance may vary by that same factor. This holds in general for any variation in depth across an object. It is clear that the amount of subdivision, and hence the size of the approximation produced, are directly related to the tolerance. The gains in performance and data reduction are very data and view dependent and we will not analyze them.

Testing cost can also be reduced by sharing the testing results among neighbors. This can be done with very little storage overhead. When the minimum subdivision level of a border has been reported, subdivision of the border can be performed to the given level without the need of testing. A patch can have four offspring with four common borders between them. The testing information of these four borders can be shared. Once the minimum number of tests has been performed on a border, testing must be performed, since an offspring's border can

require more than the minimum number of subdivisions. The border test must be performed for each leaf of the subdivision tree. Thus, we may only have to perform the final test that verifies the flatness of a shared border. This can result in a 25% reduction in the number of tests for a shared border in a 4-ary tree, and a 50% reduction in a 2-ary tree.

Another technique that we use is to perform subdivision in breadth-first rather than depth-first manner. By considering different subdivision depths for each direction and examining the subdivision trees that result, the relationship between subdividing in breadth-first versus subdividing depth-first can be determined. Preventing delays in subdivision, by testing both parametric directions at once, can reduce the number tests by 25% for a surface curved in only one direction. However, in the worst case situation of the equally curved surface, it will actually increase the number of tests by about 18%. On average we expect performance improvements. Heuristics can be used to improve performance when a surface is equally curved in both parametric directions.

9.2.2. Corner Information

The cost of using the corner information is again variable. The best case is that there is no corner information and therefore no cost; that is, the patch became flat all at once. The worst case is that corner information is specified for all boundaries and that we are performing closest point calculations, rather than midpoint calculations, to provide the least amount of error in the final approximation.

The passing down of corner information is simple; it is just a copy operation. There will be a maximum of four points in the corner information. If a new flat boundary segment is added, the current corner information must be checked since any previous corner information has priority. Therefore, in the worst case there will be the four copy operations and four closest point calculations at each subdivision level. These four intersection calculations cost about half as much as the flatness test of a control polygon.

By examining the worst case subdivision tree and assuming minimal testing costs to maximize the effect of crack prevention, we find that crack prevention could add about 15% in time costs.

The average case cost for crack prevention should be very small. If we simply interpolated midway between corner points rather than performing closest point calculations, this would reduce the time cost to about a 2% increase for crack prevention, due to the simplification of the calculations. Although it adds distortion, in practice this tradeoff is very useful.

One way to reduce the cost of the closest point calculations is to only do the midpoint calculations once and share them with the bordering neighbor. However, this adds new overheads in space and time because it requires saving the subdivision tree (like Nydegger) and updating its contents to ensure that the information is passed to the appropriate neighbor. The costs are high and outweigh the benefits.

10. Integration with Hidden Surface Removal Algorithms

One of the main reasons for doing adaptive surface subdivision is to enable polygonally-based hidden surface removal algorithms to render Beta-spline surfaces with various shading techniques. An adaptive visual tolerance is one particular method of reducing the number of output polygons with respect to the rendering process. An approach that we have taken is to separate the task of data generation from hidden surface algorithms as much as possible for the sake of simplicity. This does not mean that the hidden surface algorithm will not communicate with the data generation mechanisms in sophisticated ways. Hidden surface removal algorithms can call such polygon generators for data in ways that will allow for efficiencies based on visibility constraints. They can also ask the subdivision system to maintain the information over frames to allow for minimal updates.

The types of extensions we are investigating to provide greater efficiencies for hidden surface removal are:

1) the specification of a polygonal window in image space within which subdivision is to take place; this would be the case where certain parts of the surface were obscured

2) subdivision in scanline order

3) saving the subdivision in a minimal way, useful in frame-to-frame calculations so that operations like transformations can be applied to the previous subdivision rather than requiring resubdivision

Our method has been integrated into a Beta-spline surface subdivision system which is being extended to include such facilities and make it an efficient hidden surface utility.

The subdivision system can produce a variety of information. Besides nonplanar quadrilaterals, it can output triangulated versions of the quadrilaterals (for simple rendering), normal vectors (for lighting), parametric values (for texture mapping), and even fractal values (for perturbing the resulting object). To prevent visual asymmetries for large tolerances, we often create four, rather than two, triangles from the quadrilaterals approximating a surface. The average of the four points of the quadrilateral is used as the common vertex of the four triangles.

The actual implementation of our Beta-spline surface subdivision system is a pipeline of processes. The Beta-spline surfaces are designed on an interactive vector refresh system. The surfaces are then sent to a process that converts them to an equivalent Bézier representation for given values of bias (β_1) and tension (β_2). The Bézier representation is then passed to the subdivider which outputs polygons with auxiliary information as requested. The polygons are then given to a rendering system with the final image appearing on a raster display (Figure 10.1).

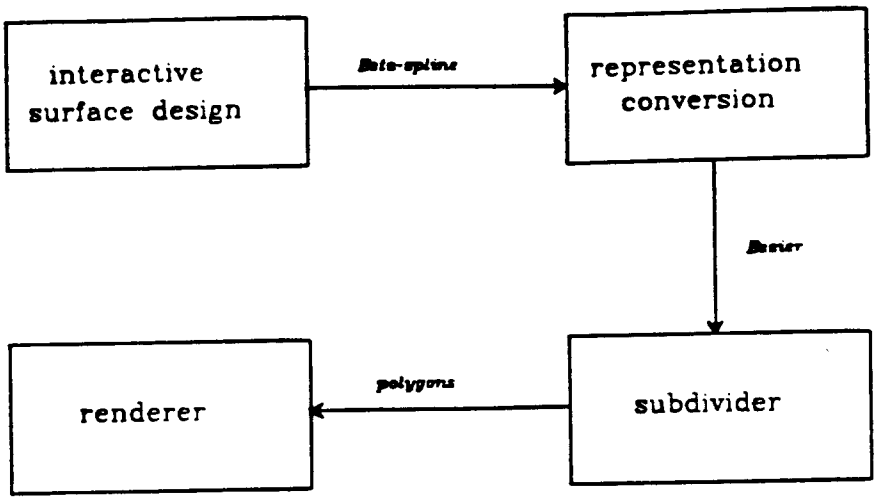


Figure 10.1.
Block diagram showing the components of the Beta-spline surface subdivision system.

11. Conclusions

There are several differences between this work and previous work: we are subdividing spline surfaces and performing directionally sensitive subdivision while reducing testing, preventing cracks and using adaptive tolerances.

The ability to subdivide splines enables their straightforward use in computer-aided geometric design and image synthesis systems which require the abilities to intersect objects and render them with polygonally based methods.

The properties of testing metrics were examined and various flatness criteria justified. We have also considered the reduction of testing costs. This is an important issue because testing costs are generally as great as, or even greater than, the subdivision itself.

The method of using a visually adaptive subdivision tolerance is an important way for reducing the number of output polygons and controlling the level of detail of the rendered object.

The crack prevention techniques are simple to implement and do not allow cracks to be generated by the subdivision. Thus, retention or reexamination of the subdivision tree is not required. They also prevent cracks from appearing in the approximation without creating unnecessary distortion.

We have presented a simple time/space efficient method for adaptive subdivision of spline surfaces without cracks. This provides natural looking polygonal approximations that can be generated with little additional space and time overhead. It can also be used in conjunction with hidden surface removal algorithms for

more powerful control of the subdivision process. The method has been implemented and is busily generating polygonal approximations (Figure 11.1).

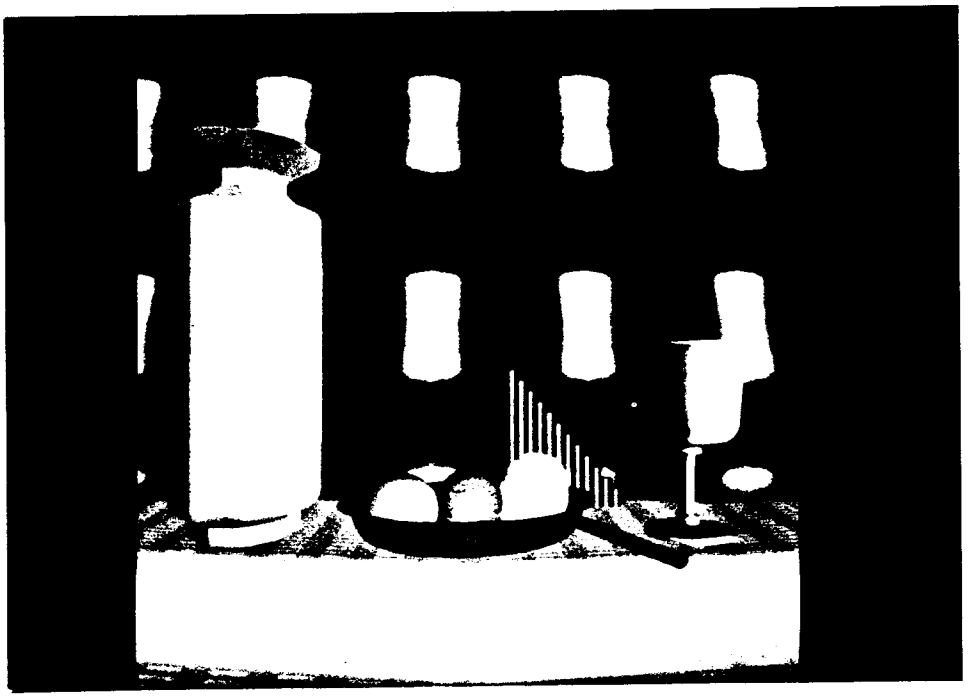


Figure 11.1.
An image created using the Beta-spline surface subdivision system described in the paper.

References

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

[2] Brian A. Barsky, *Arbitrary Subdivision of Bézier Curves,* Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA., October, 1985.

[3] Brian A. Barsky, "The Beta-spline: A Curve and Surface Representation for Computer Graphics and Computer Aided Geometric Design," in *(book title unknown)*, ed. Earnshaw, Rae A. and Rogers, David F., Springer-Verlag, Heidelberg, 1987. To appear in book from the International Summer Institute, June 1986, Stirling, Scotland.

[4] Brian A. Barsky, *Computer Graphics and Geometric Modelling Using Beta-splines,* Springer-Verlag. To appear.

[5] Brian A. Barsky, *Algorithms for the Evaluation and Perturbation of Beta-splines.* Submitted for publication.

[6] Brian A. Barsky and John C. Beatty, *Varying the Betas in Beta-splines,*

Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA., December, 1982. Also Tech. Report No. CS-82-49, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[7] Brian A. Barsky and John C. Beatty, "Local Control of Bias and Tension in Beta-splines," *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 109-134, April, 1983. Also published in *SIGGRAPH '83 Conference Proceedings* (Vol. 17, No. 3), ACM, Detroit, 25-29 July, 1983, pp. 193-218.

[8] Richard Bogen, Jeffrey Golden, Michael Genesereth, and Alexander Doohovskoy, *MACSYMA Reference Manual*, M.I.T., Cambridge, Massachusetts, December, 1977. Version nine.

[9] Edwin E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, University of Utah, Salt Lake City, Utah, December, 1974. Also Tech. Report No. UTEC-CSc-74-133, Department of Computer Science, University of Utah.

[10] James H. Clark, "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces," in *SIGGRAPH '79 Conference Proceedings*, vol. 13, August, 1979. Supplement to proceedings.

[11] S. Conte and C. de Boor, *Elementary Numerical Analysis, 3rd Edition*, McGraw-Hill Book Company, New York, 1980.

[12] Tony D. DeRose, *Arbitrary Subdivision of Blended Splines*. In preparation.

[13] Richard J. Fateman, *Addendum to the MACSYMA Reference Manual for the VAX*, Computer Science Division, University of California, Berkeley, 1982.

[14] Ronald N. Goldman, *Markov Chains and Computer Aided Geometric Design*. Submitted for publication.

[15] Ronald N. Goldman and D. C. Heath, *Linear Subdivision is Strictly a Polynomial Phenomenon*. Submitted for publication.

[16] 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, pp. 290-297, November, 1979.

[17] Jeffrey M. Lane, Loren C. Carpenter, J. Turner Whitted, and James F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communications of the ACM*, vol. 23, no. 1, pp. 23-34, January, 1980.

[18] 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*, vol. PAMI-2, no. 1, pp. 35-46, January, 1980.

[19] Robert W. Nydegger, *A Data Minimization Algorithm of Analytical Models for Computer Graphics*, University of Utah, Salt Lake City, Utah, 1972.

[20] Richard F. Riesenfeld, Elaine Cohen, Russell D. Fish, Spencer W. Thomas, Elizabeth S. Cobb, Brian A. Barsky, Dino L. Schweitzer, and Jeffrey M. Lane, "Using the Oslo Algorithm as a Basis for CAD/CAM Geometric Modelling," in *Proceedings of the Second Annual NCGA National Conference*, pp. 345-356, National Computer Graphics Association, Inc., Baltimore, 14-18 June, 1981.