Construction of Smooth Curves and Surfaces from Polyhedral Models

by

Leon A. Shirman

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Applied Mathematics

University of California
Berkeley, California

December 1990

CONSTRUCTION OF SMOOTH CURVES AND SURFACES

FROM POLYHEDRAL MODELS

Leon A. Shirman

To My Parents

Acknowledgements

I am deeply grateful to many people for their help and support during my doctoral studies and the lengthy production of this thesis. My sincere thanks to each of them.

Foremost, I would like to thank Professor Carlo Séquin, my thesis advisor, not only for his enthusiastic interest and contribution to this dissertation, but also for the encouragement and support he has given me. Through him, many opportunities were made available to me that would have otherwise been impossible.

My other committee members, Professors Brian Barsky, Beresford Parlett, and Ole Hald have also been very encouraging and helpful and I sincerely thank them for that.

I would also like to thank Berkeley graduate students in Computer Graphics, especially Henry Moreton and Seth Teller, for their useful suggestions and discussions during my studies.

Most of all, I am deeply grateful to my family and friends for their love and moral support crucial to the completion of this work.

This work was supported through a variety of industrial gifts from Siemens Corp., Silicon Graphics Inc., and Tektronix.

Table of Contents

Construction of Smooth Curves and Surfaces from Polyhedral Models

Leon A. Shirman

ABSTRACT

This work is devoted to the problem of constructing geometrically continuous interpolating curves and surfaces through a given set of points. Geometric continuity is a relaxed measure of parametric continuity and is defined in terms of continuity of tangent vector for curves and of tangent plane for surfaces. Various solutions to the above problem have been proposed. However, the resulting curves and surfaces are often of poor visual quality and contain unnecessary 'wiggles' and 'overshoots'. Our method breaks the process of curve or surface construction into several independent procedural steps and assigns available degrees of freedom based on geometric reasoning. This greatly improves the resulting visual quality of the object.

Our curve interpolation method builds G^1 continuous curves from cubic Bézier segments. A number of heuristic rules for determining vertex normals and curve control points is introduced. Furthermore, this approach is extended to produce curvature continuous (G^2) cubic splines.

This procedural approach is generalized to the construction of surfaces from a raw polyhedral model, supplied by the user. This construction proceeds in several steps: first, suitable normals at each vertex of the original model are computed; they will become the normals of the final surface at these points. Second, cubic Bézier curves are constructed in place of the edges of the original model; the final surface will interpolate these curves. Third, surface patches are defined with these curves as their boundaries; the patches will jointly form an interpolating G^1 surface. At each of the above steps, heuristic rules are used that assign reasonable values to available degrees of freedom, such as vertex normals or tangent directions of boundary curves. However, the user always has an option of overriding default assignments with prescribed values.

From the mathematical analysis of G^1 continuity conditions between Gregory and/or Bézier patches, it follows that the surface can be defined in several ways and still maintain required locality and interpolating properties. These available degrees of freedom that are intrinsically present in the system, are utilized to alter the cubic boundaries as well as the shape of the two patches near their common seam. They are made available to the user in the form of intuitively understandable *shape parameters*.

Simplicity of the surface representation is important for rendering efficiency. Therefore, a method was developed that replaces the Gregory patches used in the original implementation by polynomial quadrilateral and triangular Bézier patches.

The described method of surface construction has several important advantages. The method is *modular*, i.e. the construction process is split into several separate stages so that a change of procedural rules at one stage does not affect another. The method is also *local*, and the underlying surface representation is *simple*. The method is *flexible* because surfaces can be built for various types of data (e.g. points, points with normals, profile curves). Finally, the method gives to a user many *shape parameters* that can be used to alter the final shape; however, an *automatic* default solution can always be computed.

1

Introduction

In the early days of Computer Aided Design and Manufacture (CAD/CAM), it was common to model objects with linear segments and planar polygonal facets. However, polygonal modeling is not well suited for the representation of the intrinsically *smooth* objects, such as a car hull, a human face, or an airplane. To deal with object like these, curves and surfaces modeled by *parametric splines* have been widely utilized in the computer graphics community. Various mathematical expressions for smooth surface design have been proposed by Coons [29], Bézier [11], Gregory [61], and others [10, 43, 46]. This work describes procedural approaches to creating smooth objects and surfaces by rounding of polyhedral models and by creating smooth interpolations between given profile curves.

A widespread method for constructing smooth surfaces in CAD/CAM assembles many surface patches in such a way that neighboring patches meet smoothly. Smoothness can be defined in terms of differentiability of the surface across their common boundary. However, very often only the shape of adjoining patches, not their parametrization, is important. Then, *geometric continuity* [3, 8, 9, 32] is sufficient. A key problem in designing a smooth surface is to ensure smoothness across boundary curves.

There are two major classes of surfaces used in CAD/CAM, those that *interpolate* the given vertices or curves and those that *approximate* them. In this work, we have concentrated our attention on interpolating surfaces because they arise naturally as a result of procedural surface construction. Moreover, interpolating surfaces are practical for industrial applications since often several points, or even a whole profile, of the desired surface are known precisely.

It is typically the case that many different smooth surfaces can be constructed for a given set of interpolation data. One way to avoid this ambiguity and to develop a method that builds unique surfaces is to assign reasonable default values for the extra degrees of freedom. Alternatively, these extra degrees of freedom can be made available to the user for controlling and fine-tuning the shape of the resulting surface. Moreover, additional flexibility can be very useful for more strongly constrained input, for example, if the given boundary curves or vertices need to be interpolated with specified tangent directions.

Extra care must be taken when assigning values to the available degrees of freedom. While perfectly smooth G^1 surfaces could be constructed even for poor choices of parameters, the result

might not be pleasing to the human eye nor be satisfactory for the intended purpose. For example, undesired waves on the surface could occur. Interpolation with *pleasing splines* is an important aspect of our approach.

Local control over a surface is yet another desirable property. Often only a small part of a surface need to be changed and one would then like the rest of it to remain unaffected. Locality of control is also very important for an interactive system. Finally, simplicity of the underlying representation is clearly required for an efficient and fast display of potentially complicated curved objects.

1.1. Procedural Interpolation

Our interpolation scheme addresses all of the above issues. A user is expected to supply at least a set of points to be interpolated in $\mathbf{R}^3$. In addition, the following information may or may not be supplied:

- A *tessellation* that connects the given points into triangular or quadrilateral faces, thus defining explicitly the topology of the surface to be constructed. Each edge in the tessellation will be converted into a boundary line between two patches forming the surface.
- *Normals* to the surface at the given points. The resulting surface, in addition to interpolating the points, will be orthogonal to the above normals at each given point.
- *Curves* between the given points that the final surface will interpolate along with the given points. The final surface will be made of the patches, joining across these boundary curves. The curve network must, of course, be compatible with the set of faces and the normals.
- A set of *shape parameters* that controls the shape of the surface both locally and globally.

This additional information need not be supplied for the whole data set. For example, normals could be given only at few points.

The construction of the surface proceeds in the following steps:

1. The tessellation into triangular or quadrilateral patches is performed (if it is not already supplied by the user).
2. Normals at each given point are computed (if they are not already supplied by the user).
3. Each edge in the tessellation is replaced by a boundary curve (if the curves are not already supplied by the user). Cubic Bézier curves are used for boundary representation.
4. The patches with the boundaries defined in the previous step are constructed. Each pair of patches will be geometrically continuous across their common boundary, thus making the whole surface G^1. Bicubic quadrilateral and quartic triangular Bézier or Gregory patches are used.

5. The shape of the surface is adjusted with local and global shape parameters.

The above approach to surface construction is illustrated in Fig. 1.1.

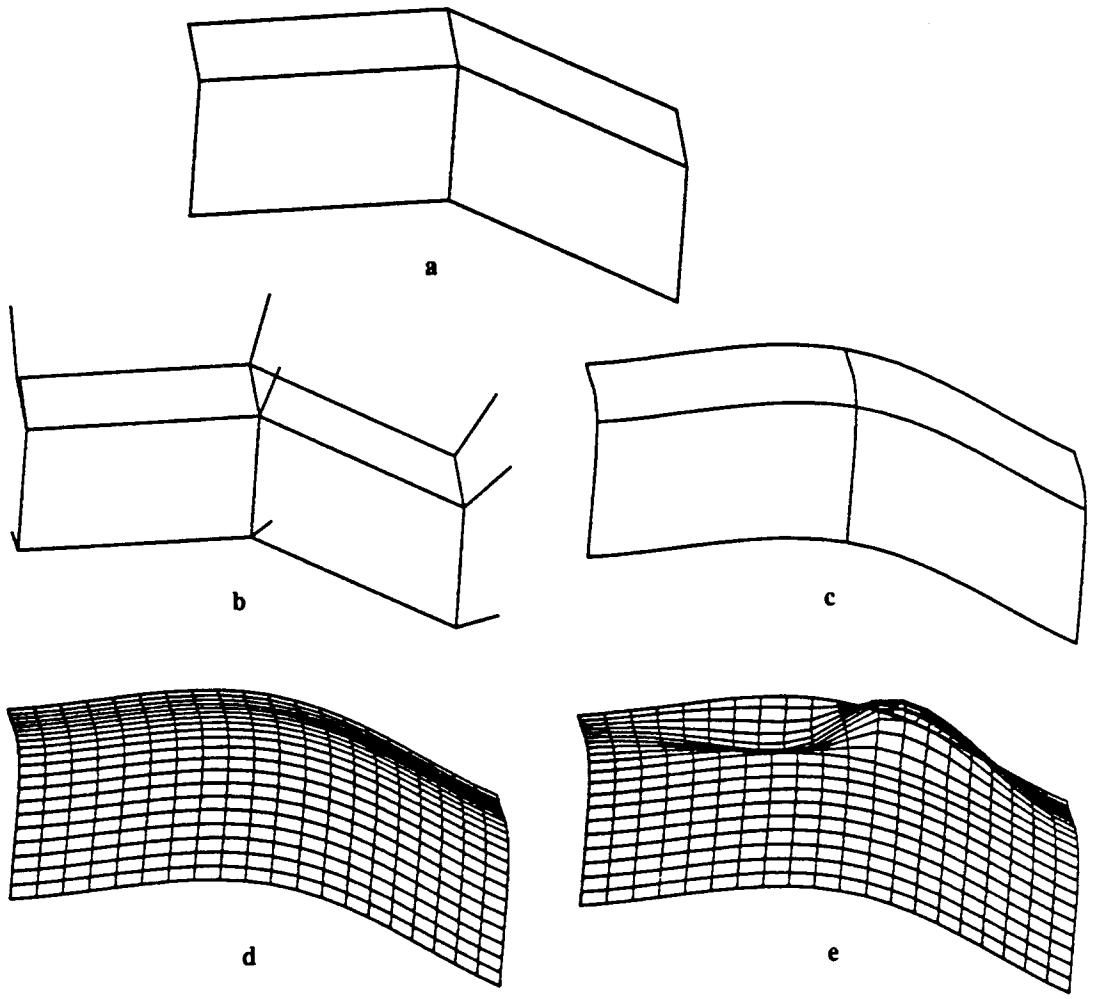


Figure 1.1: *Procedural approach of surface construction.*
a. *Original polyhedral frame.*
b. *Normals are computed at each vertex.*
c. *Each edge of the original object is replaced by a curve.*
d. *G^1 continuous patches are filled into the meshes.*
e. *Shape of the surface is adjusted.*

The procedural, step-by-step method of constructing surfaces is easily understood by the user. A *default* solution is computed by choosing reasonable values for all parameters not explicitly specified by the user, thus allowing minimal input. Then it is possible to fine-tune the solution to a user's particular needs by changing several parameters that define the surface. Due to the locality of the method and the fact that the patches are polynomial, the changes can be

performed very efficiently.

It turns out that the problem of selecting surface normals and eventually boundary curves from a set of data points is difficult in the sense that it is hard to avoid some solutions of poor quality in a resulting surface for at least some sets of data points. For strongly asymmetric sets of interpolation points, a naive method of surface construction would typically produce a surface with creases, spikes, or overshoots. These unsatisfactory results are generally characterized by large variations of curvature along the surface. To gain insight into 'smarter' ways of selecting normals and boundary curves, we first analyze the interpolation problem for curves.

In this problem, a set of points in $\mathbf{R}^2$ or $\mathbf{R}^3$ is given, perhaps with optional normals at selected points. Our procedural method of constructing interpolating G^1 curves through a given set of points produces curves of high visual quality (typically measured in terms of curvature variation). The approach is very general and flexible and easily extends to produce curvature continuous (G^2) curves. The study of interpolating curves provides the foundation for the construction of boundary curves for interpolating surfaces.

1.2. Thesis Overview

This presentation is logically divided into 4 parts.

The first part (Chapter 2) is devoted to previous relevant research in spline construction and surface interpolation. Mathematical formulations of Bézier and B-splines are given. Different surface patch types are discussed: Coons patches, Hermite patches, and triangular and quadrilateral Bézier patches. The need for Gregory patches in an interpolation problem is explained. Finally, composite surfaces and geometric continuity are discussed.

In the second part (Chapters 3 and 4), we introduce our concept of procedural interpolation for curves. In Chapter 3, we start by listing useful properties that pleasing splines should have. Then we present our procedural approach for the case of curves composed of cubic segments. This process has two stages: first, the tangent direction (or the normal) at each vertex is determined. Second, Bézier points of cubic segments are chosen on appropriate tangent lines. Various rules for choosing these normals and Bézier points are further described that guarantee pleasing shapes for the resulting curves. In Chapter 4, the procedural method is extended to produce G^2 curves. In this method, two cubic Bézier segments are used between each pair of adjacent vertices. A rule to choose default curvatures at interpolation points is also given.

The third part of this work (Chapters 5–7) is devoted to surface interpolation. In Chapter 5, we extend the procedural approach of Chapter 3 to define surface normals and Bézier points of boundary curves. Several methods are introduced and compared to each other. In Chapter 6, we address the next logical step in surface construction: filling in Gregory patches into a mesh of boundary curves. Further, we study the G^1 continuity constraints across patch boundaries. The available degrees of freedom can be made available in the form of shape parameters to change the

shape of two adjoining patches near a common seam without destroying G^1 continuity. Finally, in Chapter 7, we introduce a method to fill in several Bézier patches into a given curve mesh instead of (rational) Gregory patches. The resulting polynomial surface is guaranteed to be G^1 across both interior and exterior boundaries.

Finally, in the last part (Chapter 8) we describe UNICUBIX, an experimental modeling system which was used to test the above ideas. Concluding remarks are given in Chapter 9.

2

Overview of Surface Modeling with Patches

In this chapter, an overview of different patch types is presented. Mathematical formulations of Coons, Gordon, B-spline, Bézier, and Gregory patches are given. Finally, composite surfaces and the notion of geometric continuity are discussed.

2.1. Types of Surface Patches

2.1.1. Parametric and Algebraic Patches

The simplest and the most versatile mathematical element used to model a surface is a *patch*. There are two basic types of patches used in Computer Aided Geometric Design: parametric and implicit. A parametric patch is a curve-bounded collection of points whose coordinates are given by continuous parametric functions of the form

$$x = x(u,v), \quad y = y(u,v), \quad z = z(u,v), \tag{2.1}$$

or, in the vector form,

$$\mathbf{s} = \mathbf{s}(u,v). \tag{2.2}$$

The parametric variables u and v are constrained to lie in some interval, usually in [0,1].

An implicitly defined surface, on the other hand, is described by a single implicit equation:

$$f(x,y,z) = 0. \tag{2.3}$$

If the function $f(x,y,z)$ is polynomial, the surface is called algebraic.

In some cases, it is advantageous to use an implicit rather than a parametric representation. Unlike parametric patches, algebraic surfaces do not need the extra parameters u and v, which decreases the dimension of the space of variables. This may result in simpler expressions for geometric shapes. For example, an algebraic equation for a sphere $x^2 + y^2 + z^2 - r^2 = 0$ cannot

be expressed in a polynomial form by using a parametric representation. On the other hand, it is generally hard to define a bounded surface in algebraic terms. Moreover, the parametric form is usually more intuitive as one has full control over the parameters u and v. Their exact domain is always known; it is easy to identify boundaries and corners of the surface, etc. One can visualize a parametric patch simply as a mapping of a region in the (u,v) plane into (x,y,z) space.

Consequently, most of the research on free form surfaces has focused on the parametric form. Many elegant techniques exist for modeling with parametric patches [10, 43, 46]. Conversely, algebraic surfaces are rarely used for geometric design and only a few papers [122, 123, 124] exist on the subject. Moreover, many of the existing solid modeling systems [26, 112, 121] do not accommodate algebraic patches. This work will deal with parametric patches only.

2.1.2. Quadrilateral and Tensor Product Patches

Although theoretically a patch can have any number of boundary curves, mostly quadrilateral and triangular patches are used. Quadrilateral patches are especially popular, as they can be thought of as a mapping of a unit square in the (u, v) plane (Fig. 2.1) and therefore are conceptually simple.

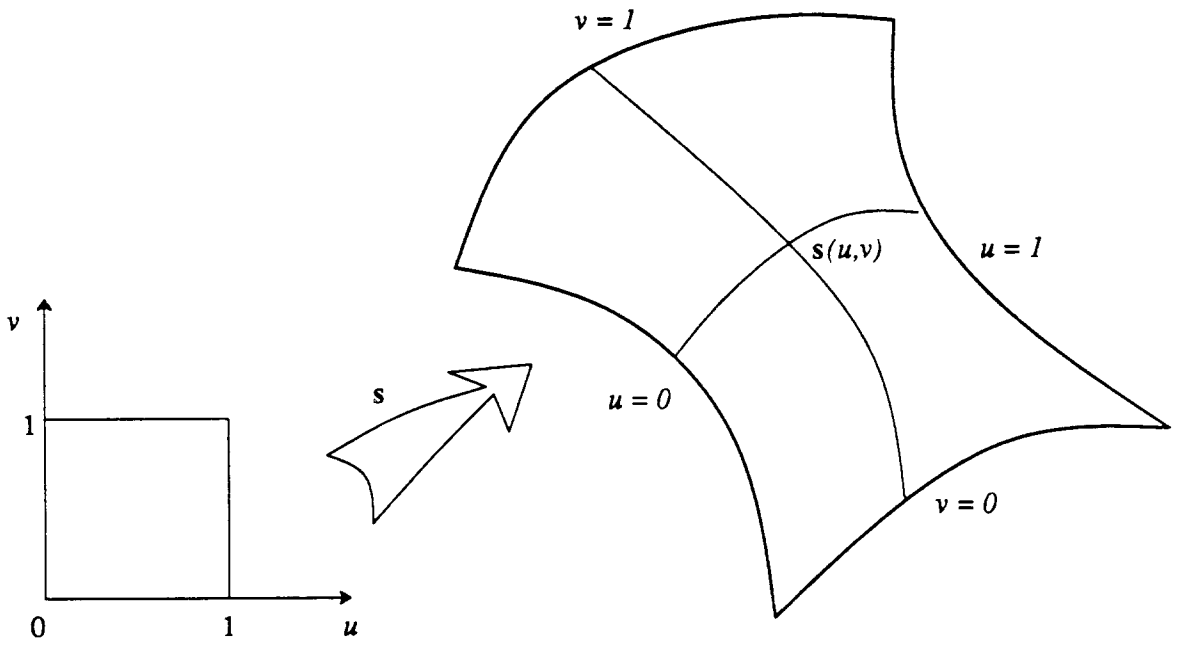


Figure 2.1: *A parametric surface patch.*

Among the quadrilateral patches, *tensor product* patches are widely used. A tensor product patch is defined by

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{V}_{ij}\, p_i(u)\, q_j(v), \quad u, v \in [0,1]. \tag{2.4}$$

Here, V_{ij} is a rectangular grid of *control vertices*, or just 3-dimensional coefficients, and $p_i(u)$ and $q_j(v)$ are scalar functions that can be used for curve construction. Thus, tensor product patches are extensions of the one-dimensional scheme for curves

$$\mathbf{c}(u) = \sum_{i=0}^{n} \mathbf{V}_i\, p_i(u) \tag{2.5}$$

to two dimensions. One of the reasons that these patches are so common is the fact that the smoothness of the surface $\mathbf{s}(u,v)$ is guaranteed by the smoothness of the curves $p_i(u)$ and $q_j(v)$ [32, 59]. Hermite, Bézier, and B-spline quadrilateral patches are examples of the tensor product patches, whereas Coons patches are not.

2.1.3. Triangular Patches and Barycentric Coordinates

Quadrilateral surface patches are well-suited for the modeling of objects that have a rectangular structure, such as a car door or a top of a human head. However, applying quadrilateral patches to other objects can result in degenerate patches. In these situations, triangular patches, which do not suffer from these degeneracies, are superior. Triangular patches are also better suited for the problem of fitting smooth surfaces to scattered data [41]. For these reasons, triangular patches have become increasingly important in Computer Aided Geometric Design, or CAGD. A lot of literature exists on the definition and use of triangular patches for surface construction, for example [2, 15, 42, 64, 76, 93, 106].

Just as a quadrilateral patch is a mapping of a unit square, a triangular patch is a mapping of an equilateral triangle with a unit side length. Triangular patches are best expressed in barycentric coordinates.

Let T be a triangle in space with vertices $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ (Fig. 2.2). A point $\mathbf{P}$ in the plane of the triangle can be uniquely expressed as

$$\mathbf{P} = u\, \mathbf{T}_1 + v\, \mathbf{T}_2 + w\, \mathbf{T}_3, \quad u + v + w = 1. \tag{2.6}$$

$\mathbf{P}$ is said to have *barycentric coordinates* (u,v,w) with respect to T. The interior of the triangle is characterized by the additional restriction $u, v, w \geq 0$.

The patch itself is then expressed as some function of the three variables u, v, and w. Since only two of them are independent, the function is still bivariate.

2.1.4. Other Patch Types

One does not have to work exclusively with quadrilateral or triangular patches. It is possible to define a patch with more than 4 boundaries by a suitable mapping of a n-gon in (u,v) plane. These non-standard patches can be useful in certain design situations. For example, a pentagonal patch will be the natural choice for designing a dome with a pentagonal floor. Several

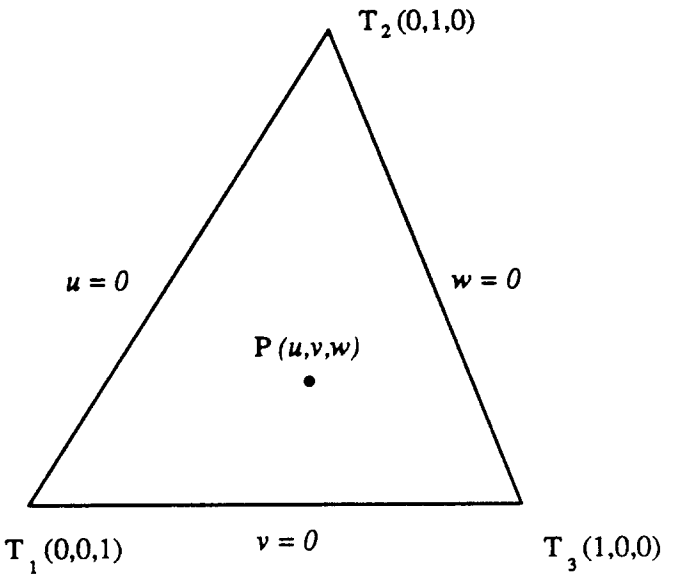


Figure 2.2: *Barycentric coordinates.*

formulations for such patches have been proposed [24, 70, 113].

However, the situations when the patches with large number of boundaries are useful are relatively rare, and the high cost of incorporating these patches into a practical design systems is usually not justified. Moreover, as shown in [114], for $n > 5$ the mathematical expressions for corresponding patches become very complicated. Recent work on S-patches, or n-sided patches well suited for inclusion into the network of Bézier curves demonstrated that they are of very high degree [85].

There are quite a few other mathematical formulations for surface patches. They include cylindrical and ruled surfaces, surfaces of revolution, etc. However, they are not relevant to this work and therefore will not be discussed here. The reader is referred to corresponding textbooks [46, 91] for a complete treatment of the subject.

2.2. Coons Patches

Coons patches [29, 30] are completely defined by their boundaries. They are easy to construct and are well-suited for interpolation of the network of curves in space. The most widely used Coons patches are bilinearly blended, which are defined solely by the boundary curves positions in space, and bicubically blended, which additionally depend on the derivatives on the boundaries. In general, Coons patches are not tensor product.

2.2.1. Bilinearly Blended Coons Patches

Given the four patch boundaries, $\mathbf{r}(u,0)$, $\mathbf{r}(0,v)$, $\mathbf{r}(u,1)$, $\mathbf{r}(1,v)$ (Fig. 2.3), we seek an expression for a patch that would interpolate these boundaries. It can be shown [43] that $\mathbf{r}(u,v)$ can be conveniently expressed as

$$\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2 - \mathbf{r}_{12}, \tag{2.7}$$

where $\mathbf{r}_1(u,v)$ is a *ruled surface* that linearly interpolates the two boundaries $\mathbf{r}(0,v)$ and $\mathbf{r}(1,v)$:

$$\mathbf{r}_1(u,v) = (1-u)\,\mathbf{r}(0,v) + u\,\mathbf{r}(1,v). \tag{2.8}$$

Analogously, $\mathbf{r}_2(u,v)$ interpolates the other two boundaries:

$$\mathbf{r}_2(u,v) = (1-v)\,\mathbf{r}(u,0) + v\,\mathbf{r}(u,1). \tag{2.9}$$

Now the sum $\mathbf{r}_1 + \mathbf{r}_2$ represents a patch each of whose boundaries is a sum of the desired curve with the linear interpolant between the endpoints of that curve. The function $\mathbf{r}_{12}(u,v)$ compensates for this excess:

$$\mathbf{r}_{12}(u,v) = (1-u)\,(1-v)\,\mathbf{r}(0,0) + u\,(1-v)\,\mathbf{r}(1,0) + (1-u)\,v\,\mathbf{r}(0,1) + u\,v\,\mathbf{r}(1,1). \tag{2.10}$$

The functions u, $1-u$, v, and $1-v$ are called *blending functions*, because they are used to blend together four separate boundary curves to give a well-defined surface.

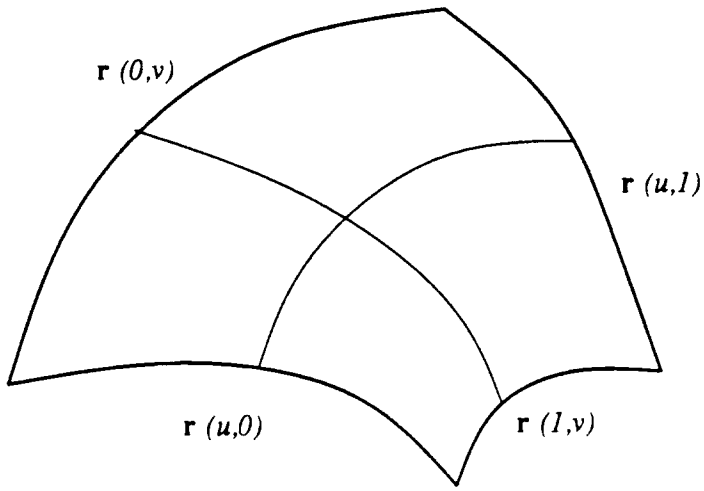


Figure 2.3: *Bilinearly blended Coons patch.*

Given a network of curves, it is possible to construct a composite surface made up of patches of this type. However, this surface will only have positional continuity across the boundaries. For derivate continuity, it is necessary to use bicubically blended Coons patches.

2.2.2. Bicubically Blended Coons Patches

For these types of patches, not only positional data at the boundaries is required, but first derivative information is necessary as well. Therefore, the given data now consists of $\mathbf{r}(u,0), \mathbf{r}(0,v), \mathbf{r}(u,1), \mathbf{r}(1,v)$ and $\mathbf{r}_v(u,0), \mathbf{r}_u(0,v), \mathbf{r}_v(u,1), \mathbf{r}_u(1,v)$. The mathematical expression for the patch remains the same as in the previous section:

$$\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2 - \mathbf{r}_{12},$$

but the auxiliary surfaces $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_{12}$ are now constructed with cubic, rather than linear, interpolation:

$$\mathbf{r}_1(u,v) = H_0^3(u)\,\mathbf{r}(0,v) + H_1^3(u)\,\mathbf{r}_u(0,v) + H_2^3(u)\,\mathbf{r}_u(1,v) + H_3^3(u)\,\mathbf{r}(1,v), \tag{2.11}$$

$$\mathbf{r}_2(u,v) = H_0^3(v)\,\mathbf{r}(u,0) + H_1^3(v)\,\mathbf{r}_v(u,0) + H_2^3(v)\,\mathbf{r}_v(u,1) + H_3^3(v)\,\mathbf{r}(u,1), \tag{2.12}$$

$$\mathbf{r}_{12}(u,v) = \left[H_0^3(u)\,H_1^3(u)\,H_2^3(u)\,H_3^3(u)\right] \times \begin{bmatrix} \mathbf{r}(0,0) & \mathbf{r}_v(0,0) & \mathbf{r}_v(0,1) & \mathbf{r}(0,1) \\ \mathbf{r}_u(0,0) & \mathbf{r}_{uv}(0,0) & \mathbf{r}_{uv}(0,1) & \mathbf{r}_u(0,1) \\ \mathbf{r}_u(1,0) & \mathbf{r}_{uv}(1,0) & \mathbf{r}_{uv}(1,1) & \mathbf{r}_u(1,1) \\ \mathbf{r}(1,0) & \mathbf{r}_v(1,0) & \mathbf{r}_v(1,1) & \mathbf{r}(1,1) \end{bmatrix} \times \begin{bmatrix} H_0^3(v) \\ H_1^3(v) \\ H_2^3(v) \\ H_3^3(v) \end{bmatrix}. \tag{2.13}$$

Here, $H_i^3(u), i = 0,1,2,3$ are the Hermite polynomials of order 3 [43]. They provide the solution to the problem of finding a cubic curve through the two points with the given tangents at these points:

$$\mathbf{p}(0) = \mathbf{p}_0,\ \mathbf{p}'(0) = \mathbf{m}_0,\ \mathbf{p}'(1) = \mathbf{m}_1,\ \mathbf{p}(1) = \mathbf{p}_1. \tag{2.14}$$

The required solution is

$$\mathbf{p}(u) = \mathbf{p}_0 H_0^3(u) + \mathbf{m}_0 H_1^3(u) + \mathbf{m}_1 H_2^3(u) + \mathbf{p}_1 H_3^3(u). \tag{2.15}$$

The *twist* vectors $\mathbf{r}_{uv}$ at the four corners of the patch either have to be supplied as extra parameters or have to be selected by one of the available methods, described in [12, 22, 43].

With bicubically blended Coons patches it is possible to fill in a given network of C^1 continuous curves with a C^1 surface. At each patch corner, the twist vector is estimated. Then, the derivatives at the boundaries are defined by the Hermite interpolation:

$$\mathbf{r}_v(u,0) = H_0^3(u)\,\mathbf{r}_v(0,0) + H_1^3(u)\,\mathbf{r}_{uv}(0,0) + H_2^3(u)\,\mathbf{r}_{uv}(1,0) + H_3^3(u)\,\mathbf{r}_v(1,0). \tag{2.16}$$

The other derivatives are determined analogously. The required surface can then be constructed according to the above formulas.

2.2.3. Gordon Surfaces

Gordon surfaces[57, 58] are a generalization of Coons patches. While a Coons patch interpolates just four boundary curves, a Gordon surface interpolates a whole network of prescribed

curves $\mathbf{r}(u,v_j)$ and $\mathbf{r}(u_i,v)$, $i = 1,\ldots,n$, $j = 1,\ldots,m$ (Fig. 2.4). The idea behind the Gordon surface is the same as for the Coons patch: first, a surface $\mathbf{r}_1(u,v)$ is constructed that interpolates all the curves in the u direction. Analogously, $\mathbf{r}_2(u,v)$ interpolates in the v direction. Finally, a necessary correction $\mathbf{r}_{12}(u,v)$ is substructed.

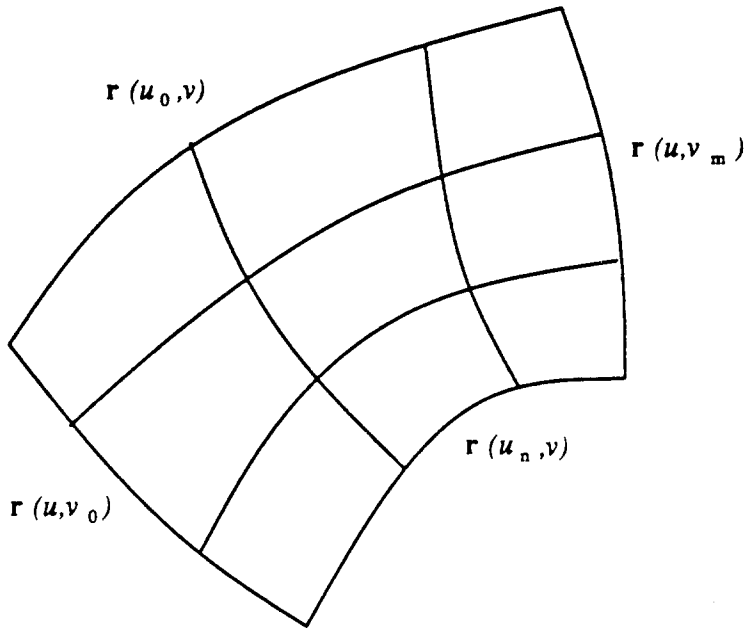


Figure 2.4: *Gordon surface.*

Lagrange polynomials [107]

$$L_i^n(u) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{n} (u - u_i)}{\prod_{\substack{j=0 \\ j\neq i}}^{n} (u_i - u_j)} \tag{2.17}$$

are very convenient for the construction of the auxiliary surfaces. The following expressions for $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_{12}$ satisfy the interpolating properties:

$$\mathbf{r}_1(u,v) = \sum_{i=0}^{n} \mathbf{r}(u_i,v) L_i^n(u), \tag{2.18}$$

$$\mathbf{r}_2(u,v) = \sum_{j=0}^{m} \mathbf{r}(u,v_j) L_j^m(v), \tag{2.19}$$

$$\mathbf{r}_{12}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{r}(u_i,v_j) L_i^n(u) L_j^m(v), \tag{2.20}$$

$$\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2 - \mathbf{r}_{12}.$$

Gordon surfaces do not have to be based on polynomial interpolation. Any univariate interpolation scheme cab be used. For example, spline-based interpolation greatly reduces the degree of the resulting surface.

2.3. Polynomial and Rational Patches

2.3.1. Cardinal Functions

Coons and Gordon surfaces are well suited for interpolating a network of given curves. However, it is often the case that only a set of points $V_i, i = 0,...,n$ that determines the shape of the surface is known. In this case, the overall surface is usually defined as

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \mathbf{V}_i p_i(u,v), \tag{2.21}$$

where $p_i(u,v)$ are the *basis* or *cardinal functions*. If, in addition, the *control points* V_i form a rectangular grid, and the cardinal functions are separable in u and v (i.e. each $p_i(u,v)$ is a product of a function of u and a function of v), then the resulting surface becomes tensor product and all its properties can be derived form the univariate case.

A curve with the n control points V_i is defined analogously to the above formula:

$$\mathbf{c}(u) = \sum_{i=0}^{n} \mathbf{V}_i p_i(u), \tag{2.22}$$

A *knot vector* $u_0 \leq \cdots \leq u_n$ is usually associated with the curve. Although the valid parameter range of $\mathbf{c}(u)$ is $[u_0,u_n]$, the cardinal functions are defined over the whole real line.

If the curve is required to *interpolate* its control points, then the cardinal functions must satisfy

$$p_i(u_j) = \delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases} \tag{2.23}$$

If this condition is not satisfied, the curve is called *approximating*.

There are many choices for the cardinal functions $p_i(u)$. The most common selection suitable for many applications is a piecewise cubic curve. This curve is composed of n cubic pieces defined on $[u_i,u_{i+1}]$, $i = 0,...,n-1$ in such a way that the prescribed continuity conditions at the knots are satisfied. Fig. 2.5a shows C^2 cardinal functions for a *natural cubic spline* [43, 107].

As one can see, the above cardinal functions are non-zero on all the intervals (u_i,u_{i+1}). Thus, a change in a position of any vertex V_i will affect the whole curve. This property is clearly undesirable for interactive design and *local*, rather than *global* methods are preferable.

Fig. 2.5b shows cardinal functions for a cubic C^1 Catmull–Rom spline [23]. The fact that the cardinal functions are non-zero only on 4 intervals means that the curve between V_i and V_{i+1}

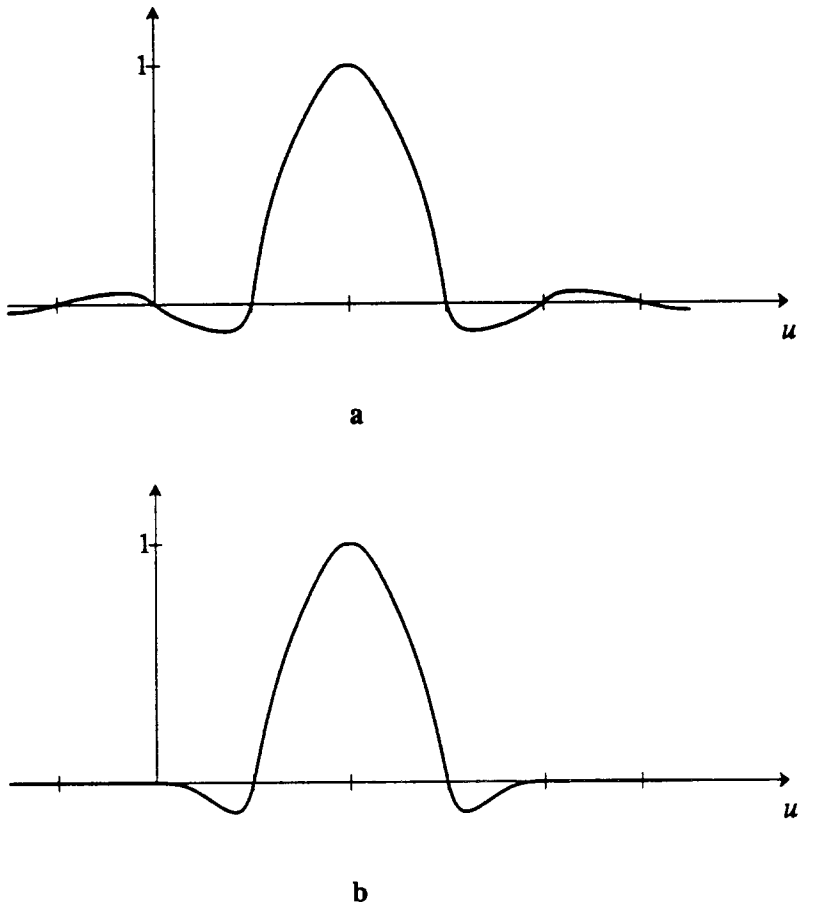


Figure 2.5: *Cardinal functions for global and local interpolation schemes.*
a. *Natural cubic spline.* **b.** *Cubic Catmull–Rom spline.*

will be affected only by V_{i-1}, V_i, V_{i+1}, and V_{i+2}. Thus, the method will be local, if the corresponding cardinal functions have *compact support.*

Finally, the tensor product surface is defined on the rectangle $[u_0,u_n] \times [v_0,v_m]$ by the products of univariate cardinal functions $p_i(u)$ and $q_j(v)$:

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{V}_{ij} p_i(u) q_j(v). \tag{2.24}$$

2.3.2. B-splines

Non-uniform rational B-splines, or NURBS, have recently gained a great popularity in computer graphics community. This can be explained by their versatility and adaptability to many CAGD applications. Bézier and Gregory patches, which are primary primitives on which this work is built, are just particular types of B-splines. Quite a few algorithms have been developed

for rendering and evaluating B-splines, and a lot of literature exists on the subject [10, 43].

Although triangular B-splines have been developed [32], B-spline patches are usually quadrilateral and of the tensor product type. In the univariate case, a rational B-spline curve of order k is given by

$$\mathbf{c}(u) = \frac{\sum_{i=0}^{m} \mathbf{V}_i N_i^k(u)}{\sum_{j=0}^{m} w_j N_j^k(u)}, \quad u \in [u_0, u_m]. \tag{2.25}$$

The $m+1$ points V_i are the control points, and the numbers w_i are called *weights*, associated with the corresponding control point. Each weight can be regarded as a fourth coordinate of the control points, so that the curve can be thought of as a perspective projection from $\mathbf{R}^4$ onto $\mathbf{R}^3$. Rational B-splines can be used to *exactly* describe conic curves and quadric surfaces [43], something that polynomial splines fail to do. For that reason, they are gaining widespread acceptance. If, for all i, $w_i = 1$, then the curve becomes polynomial, or non-rational.

The cardinal functions $N_i^k(u)$ are called *basis functions* in this context. They may be defined recursively by the Cox/deBoor relation [18]:

$$N_i^k(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_i^{k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1}^{k-1}(u), \quad k > 1, \tag{2.26}$$

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

The basis functions are defined over the *knot vector* $u_0, \ldots, u_{m+k-1}$. The continuity of the basis functions and, therefore, of the whole curve, is determined by the order of the basis functions and by the knot vector. The continuity of the basis function at a knot with multiplicity μ is $C^{k-\mu-1}$. An example of a B-spline basis function of order 4 (degree 3) on a uniform knot vector is shown in Fig. 2.6.

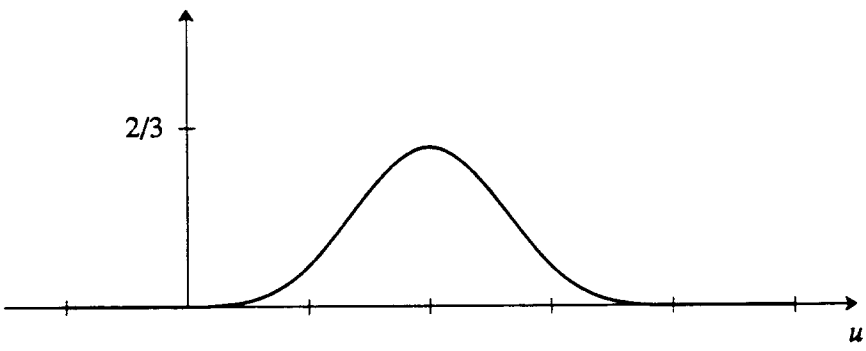


Figure 2.6: *B-spline basis function of order 4 on a uniform interval.*

B-spline basis functions have the following properties:

- They form a partition of unity, i.e.

$$\sum_{i=0}^{m} N_i^k(u) = 1; \tag{2.27}$$

- They are non-negative;
- They have local support, i.e. they are non-zero only over the subrange $[u_i, u_{i+k}]$.

These properties guarantee translation, rotation, and scaling invariance as well as the fact that curve lies within a convex hull of its control vertices and that it has local control (a perturbation of a control vertex induces only a local perturbation). Finally, B-splines are in general approximating; the condition (2.23) is clearly violated.

The (non-rational) tensor product B-spline surface of order (k, l) is given by

$$\mathbf{s}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{V}_{ij} N_i^k(u) N_j^l(v), \quad u \in [u_0, u_n], \quad v \in [v_0, v_m]. \tag{2.28}$$

An expression for the rational B-spline surface is analogous to the univariate case.

2.3.3. Bézier Patches

As stated before, Bézier splines are a special case of B-splines and therefore possess all the useful properties that B-splines have. Bézier curves of degree n can be defined as B-splines over two knots at 0 and 1 with $n+1$ multiplicity each. An equivalent, but more widespread formulation is

$$\mathbf{c}(u) = \sum_{i=0}^{n} \mathbf{V}_i B_i^n(u), \quad u \in [0,1], \tag{2.29}$$

where $B_i^n(u)$ are the Bernstein polynomials of order n:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}. \tag{2.30}$$

Thus, Bézier curves always interpolate the end vertices:

$$\mathbf{c}(0) = \mathbf{V}_0, \ \mathbf{c}(1) = \mathbf{V}_n. \tag{2.31}$$

Each Bézier curve of degree n can also be expressed in terms of Bernstein basis polynomials of degree $n+1$:

$$\sum_{i=0}^{n} \mathbf{V}_i B_i^n(u) = \sum_{i=0}^{n+1} \mathbf{W}_i B_i^{n+1}(u), \tag{2.32}$$

where the new control points $\mathbf{W}_i$ can be obtained as follows [10]:

$$\mathbf{W}_i = \frac{n+1-i}{n+1}\mathbf{V}_i + \frac{i}{n+1}\mathbf{V}_{i-1}, \quad i = 0,\ldots,n+1. \tag{2.33}$$

We assume $\mathbf{V}_{-1} = \mathbf{V}_{n+1} = 0$. This process is called *degree elevation.*

2.3.3.1. Tensor Product Bézier Patches

A tensor product Bézier patch of degree (n,m), defined over the unit square, is given by

$$\mathbf{s}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{V}_{ij}B_i^n(u)B_j^m(v). \tag{2.34}$$

Thus, the Bézier patch is completely determined by its control points $\mathbf{V}_{ij}$ (Fig. 2.7a).

A polynomial Bézier patch can also be expressed in *Hermite* form. The Hermite patch uses the position and partial derivatives at the four corners of the patch, rather than the Bézier control points. For example, a bicubic Hermite patch is characterized by positions, first derivatives, and twists at the corners:

$$\mathbf{s}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3}\mathbf{h}_{ij}H_i^3(u)H_j^3(v), \quad u,v \in [0,1], \tag{2.35}$$

where $\mathbf{h}_{ij}$ are given by the matrix from equation (2.13).

The derivative of a Bézier patch in the u direction are given by the following expression:

$$\mathbf{s}_u(u,v) = n\sum_{i=0}^{n-1}\sum_{j=0}^{m}(\mathbf{V}_{i+1,j} - \mathbf{V}_{ij})B_i^{n-1}(u)B_j^m(v). \tag{2.36}$$

Substituting $u = 0$ in the above formula gives a cross-boundary u-derivative of the patch:

$$\mathbf{s}_u(0,v) = n\sum_{j=0}^{m}(\mathbf{V}_{1,j} - \mathbf{V}_{0,j})B_j^m(v). \tag{2.37}$$

The expression for the derivative in the v direction is analogous.

2.3.3.2. Triangular Bézier Patches

Bézier patches can also be defined over the triangles. The mathematical expression is most convenient in barycentric coordinates (Section 2.1.3):

$$\mathbf{s}(u,v,w) = \sum_{\substack{i+j+k=n\\ i,j,k\geq 0}}\mathbf{V}_{ijk}B_{ijk}^n(u,v,w), \quad u,v,w\geq 0, \quad u+v+w = 1. \tag{2.38}$$

Here

$$B_{ijk}^n(u,v,w) = \frac{n!}{i!j!k!}u^i v^j w^k, \quad i+j+k = n, \quad i,j,k\geq 0. \tag{2.39}$$

are *bivariate Bernstein polynomials.* They are indeed bivariate, because one variable is dependent on the other two: $w = 1-u-v$. Fig. 2.7b shows the control points for a quartic triangular patch.

The derivative of the triangular patch in the u direction is [41]:

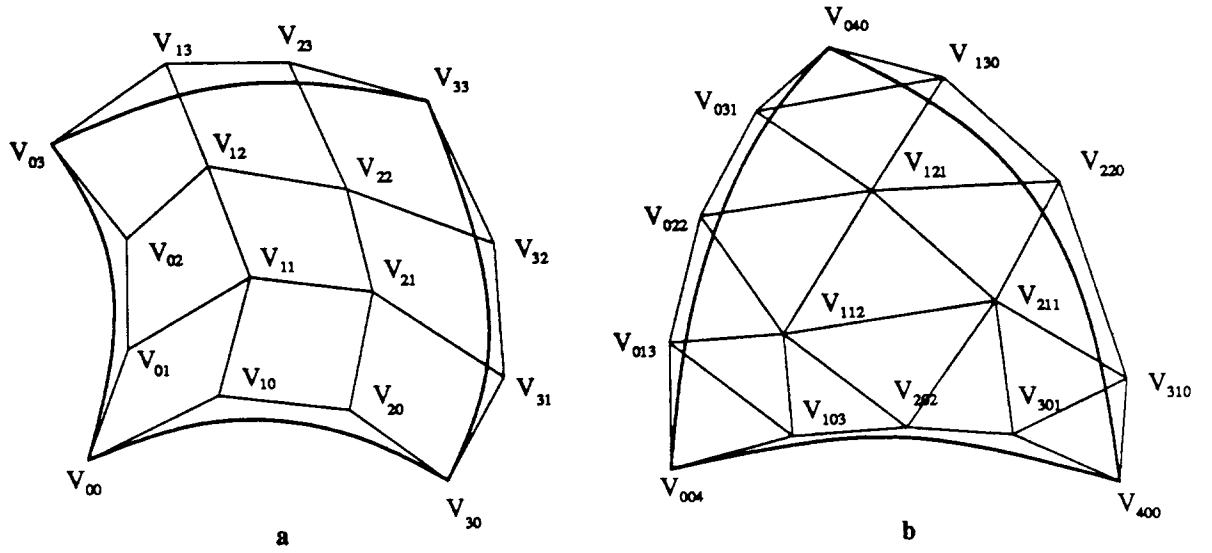


Figure 2.7: *Quadrilateral bicubic* (**a**) *and triangular quartic* (**b**) *Bézier patches.*

$$\mathbf{s}_u(u,v,w) = n \sum_{\substack{i+j+k=n-1 \\ i,j,k \geq 0}} \mathbf{V}_{ijk}^{[1]} B_{ijk}^{n-1}(u,v,w), \tag{2.40}$$

where

$$\mathbf{V}_{ijk}^{[1]} = \mathbf{V}_{i,j,k+1} - \mathbf{V}_{i,j+1,k}, \quad i+j+k = n-1. \tag{2.41}$$

The derivatives in the v and w directions can be computed analogously.

Consider now a special case when the boundary $u = 0$ of the triangular patch of degree n is of degree $n-1$. Then the border curve $\mathbf{s}(0,v,1-v)$ can be described as

$$\mathbf{s}(0,v,1-v) = \sum_{i=0}^{n-1} \mathbf{S}_i B_i^{n-1}(v). \tag{2.42}$$

In Fig. 2.8, quartic triangular Bézier patch with cubic boundaries is shown.

Rather than using derivatives in the coordinate directions, Farin [39] proposed a *radial* derivative:

$$(D\,\mathbf{s})(v) = (1-v)(\mathbf{s}_u - \mathbf{s}_v) + v(\mathbf{s}_w - \mathbf{s}_v). \tag{2.43}$$

Then the radial derivative at the border $\mathbf{s}(0,v,1-v)$ can be conveniently expressed as

$$(D\,\mathbf{s})(v) = n \sum_{i=0}^{n-1} (\mathbf{T}_i - \mathbf{S}_i) B_i^{n-1}(v), \tag{2.44}$$

where the $\mathbf{T}_i$ is the next row of the control points (Fig. 2.8). It is understood that the points T_0 and T_3 are the control points for the corresponding degree n boundary curve. This expression is very similar to the equation (2.37) for the cross-boundary derivative of a quadrilateral Bézier

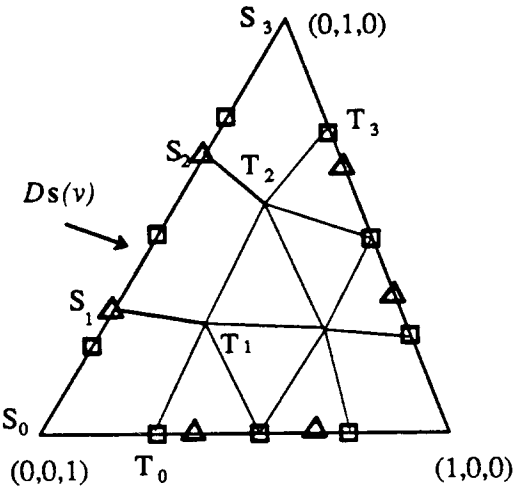


Figure 2.8: *Radial derivative for a quartic triangular patch with cubic boundaries. Cubic control points of the boundaries are marked by triangles, while degree-elevated quartic points are marked by squares.*

patch. This fact will be useful in constructing composite surfaces.

2.3.4. Gregory Patches

From equation (2.37) it follows that the cross-boundary derivative at the boundary of a Bézier patch depends on the two rows of control points: the control points on the boundary itself and the next adjoining row (Fig. 2.9). Conversely, specifying a polynomial cross-boundary derivative at the border would determine the next row of control points. Consider a bicubic Bézier patch in Fig. 2.9. If we define the cubic u-derivative at s(0,v) (obviously, it has to agree with the known derivatives $s_u(0,0)$ and $s_u(0,1)$), the control points V_{11} and V_{12} will be defined. Analogously, a v-derivative at s(u,0) will determine V_{11} and V_{21}. Thus, the point V_{11} is defined twice from the two nearest boundaries.

These definitions need not be compatible with each other, which means that the cross-boundary derivatives cannot be specified independently. An analogous *twist constraint* [61] problem arises when Coons patches are used. To remedy the situation, Gregory [62] has proposed a modification to the Coons patch by introducing *variable twists*. The resulting generalization of the Coons patch is now known as *Gregory's square*. The analogous approach has been applied to Bézier patches by Chiyokura and Kimura [27]. Although the resulting formulation is an instance of a more general Gregory's square, the term 'Gregory patch' is often used with respect to Chiyokura's generalization of Bézier patches. We will use this term throughout this work.

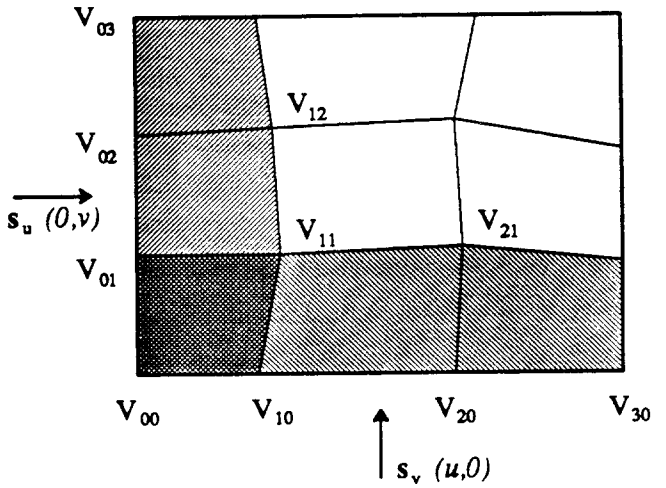


Figure 2.9: *Incompatibility of cross-boundary derivatives.*

A Gregory patch of an arbitrary degree can be defined; however, we will consider bicubic quadrilateral and quartic triangular patches only.

2.3.4.1. Quadrilateral Gregory Patches

A Gregory patch, like a Bézier patch, is defined by its *control points* $\mathbf{V}_{ij,[uv]}$, $i,j = 0,\ldots,n$. The notation $\mathbf{V}_{ij,[uv]}$ should become clear from the equations and Fig. 2.10. In the case of a quadrilateral Gregory patch (Fig. 2.10a), the equations are as follows [27]:

$$\mathbf{s}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)\, B_j^3(v)\, \mathbf{Q}_{ij}(u,v), \quad u,v \in [0,1], \tag{2.45}$$

where the effective inner control points $Q_{ij}(u,v)$, $i,j = 1,2$ are linear combinations of two control points each:

$$\mathbf{Q}_{11}(u,v) = \frac{u\,\mathbf{V}_{11,v} + v\,\mathbf{V}_{11,u}}{u+v}, \tag{2.46}$$

$$\mathbf{Q}_{12}(u,v) = \frac{u\,\mathbf{V}_{12,v} + (1-v)\mathbf{V}_{12,u}}{u+1-v}, \tag{2.47}$$

$$\mathbf{Q}_{21}(u,v) = \frac{(1-u)\mathbf{V}_{21,v} + v\,\mathbf{V}_{21,u}}{1-u+v}, \tag{2.48}$$

$$\mathbf{Q}_{22}(u,v) = \frac{(1-u)\mathbf{V}_{22,v} + (1-v)\mathbf{V}_{22,u}}{1-u+1-v}, \tag{2.49}$$

For boundary control points,

$$\mathbf{Q}_{ij}(u,v) = \mathbf{V}_{ij}. \tag{2.50}$$

For conceptual elegance, a Gregory patch can also be expressed in equivalent recursive form:

$$\mathbf{s}(u,v) = (1-u+uE)^3\,(1-v+vF)^3\,\mathbf{Q}_{00}(u,v), \tag{2.51}$$

where E and F are the *shift operators:*

$$E\,\mathbf{Q}_{ij} = \mathbf{Q}_{i+1,j}, \quad F\,\mathbf{Q}_{ij} = \mathbf{Q}_{i,j+1} \tag{2.52}$$

and $\mathbf{Q}_{ij}$ are defined as above.

A Gregory patch can be described as a Bézier patch, each of whose interior points is split in two. The influence of these two points is linearly interpolated as a surface point gains more distance from one border and approaches the other. When the eight inner control points satisfy the equations $\mathbf{V}_{ij,u} = \mathbf{V}_{ij,v}$, then the Gregory patch degenerates into a bicubic Bézier patch. Thus, a Gregory patch is indeed a generalization of a Bézier patch.

Unlike a Bézier patch, a Gregory patch is rational. In fact, it can be shown that a Gregory patch is a NURBS of degree 7.

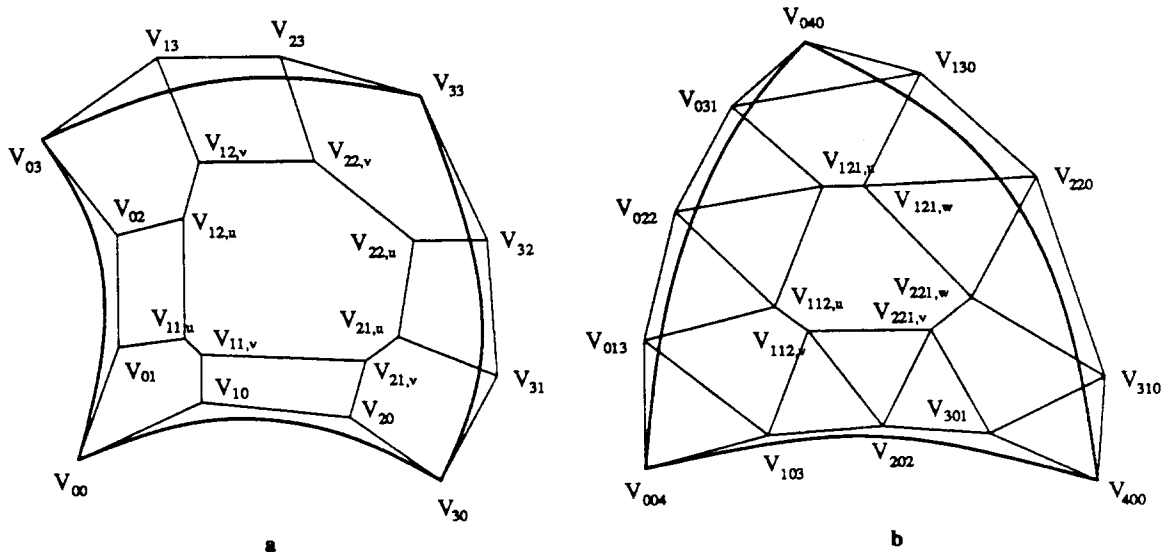


Figure 2.10: *Bicubic quadrilateral (***a***) and quartic triangular (***b***) Gregory patch.*

The derivatives of a Gregory patch will obviously be more complicated that the derivatives of a Bézier patch. However, at the borders, the derivatives look very much alike. For example, the cross-boundary u-derivative of the Gregory patch is

$$\mathbf{s}_u(0,v) = 3\,((\mathbf{V}_{10} - \mathbf{V}_{00})B_0^3(v) + (\mathbf{V}_{11,u} - \mathbf{V}_{01})B_1^3(v) + \tag{2.53}$$
$$(\mathbf{V}_{12,u} - \mathbf{V}_{02})B_2^3(v) + (\mathbf{V}_{13} - \mathbf{V}_{03})B_3^3(v)).$$

Note the similarity of this formula to eq. (2.37). This fact will enable us to stitch Bézier and

Gregory patches together easily.

2.3.4.2. Triangular Gregory Patches

Working again in barycentric coordinates, a triangular Gregory patch (Fig. 2.10b) of degree 4 is determined by the 18 control points:

$$\mathbf{s}(u,v,w) = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} B^n_{ijk}(u,v,w)\, \mathbf{Q}_{ijk}(u,v,w),\ u,v,w \geq 0,\ u+v+w = 1. \tag{2.54}$$

or

$$\mathbf{s}(u,v,w) = (uE + vF + wG)^4\, \mathbf{Q}_{000}(u,v,w),\ u,v,w \geq 0,\ u+v+w = 1. \tag{2.55}$$

where E, F, and G are again the *shift operators:*

$$E\ \mathbf{Q}_{ijk} = \mathbf{Q}_{i+1,j,k},\ F\ \mathbf{Q}_{ijk} = \mathbf{Q}_{i,j+1,k},\ G\ \mathbf{Q}_{ijk} = \mathbf{Q}_{i,j,k+1}; \tag{2.56}$$

$$\mathbf{Q}_{112}(u,v,w) = \frac{(1-u)v\,\mathbf{V}_{112,u} + u(1-v)\mathbf{V}_{112,v}}{(1-u)v + u(1-v)}, \tag{2.57}$$

$$\mathbf{Q}_{121}(u,v,w) = \frac{(1-u)w\,\mathbf{V}_{121,u} + u(1-w)\mathbf{V}_{121,w}}{(1-u)w + u(1-w)}, \tag{2.58}$$

$$\mathbf{Q}_{211}(u,v,w) = \frac{(1-v)w\,\mathbf{V}_{211,v} + v(1-w)\mathbf{V}_{211,w}}{(1-v)w + v(1-w)}, \tag{2.59}$$

$$\mathbf{Q}_{ijk}(u,v,w) = \mathbf{V}_{ijk} \quad \text{otherwise.} \tag{2.60}$$

Again, if

$$\mathbf{V}_{112,u} = \mathbf{V}_{112,v},\ \mathbf{V}_{121,u} = \mathbf{V}_{121,w},\ \mathbf{V}_{211,v} = \mathbf{V}_{211,w}, \tag{2.61}$$

the triangular Gregory patch becomes a triangular Bézier patch. Also, just as in the quadrilateral case, the derivatives of the Gregory patch at the borders (including the radial derivative) are expressed similarly to the Bézier patch derivatives.

2.4. Composite Surfaces and Geometric Continuity

As mentioned before, surface patches are the building blocks from which the whole surface is constructed. Obviously, for a continuous surface, it is necessary that the neighboring patches share the same boundary or, in other words, meet with *positional* continuity. Very often, however, the surface is required to be smooth across the patch borders. Smoothness can be defined in terms of differentiability, i.e. partial derivatives at the boundary points must be identical. Alternatively, one can often be satisfied with *geometric continuity* [8, 9], where only the shape of adjoining patches is considered, but not their parametrization.

2.4.1. Parametrically Continuous Surfaces

A method for constructing a C^1 surface from a network of C^1 curves has already been discussed in Section 2.2. Bicubically blended Coons patches describe the required surface. If, in addition, the given curves are polynomial, then the Coons patches can also be represented in Bézier form.

B-spline surfaces (without knots of too high multiplicity) are another example of parametrically continuous surfaces. The order of continuity in this case will be at least $k-\mu-1$, where k is the order of the basis functions, and μ is the highest multiplicity of a knot.

The expression for the derivatives of Bézier or Gregory patches in eq. (2.37) and (2.53) readily produces the conditions for the C^1 continuity between the two patches (Fig. 2.11). In terms of the control polyhedra of the two patches, the four pairs of polyhedron edges that meet at the boundary must be collinear and equal in magnitude.

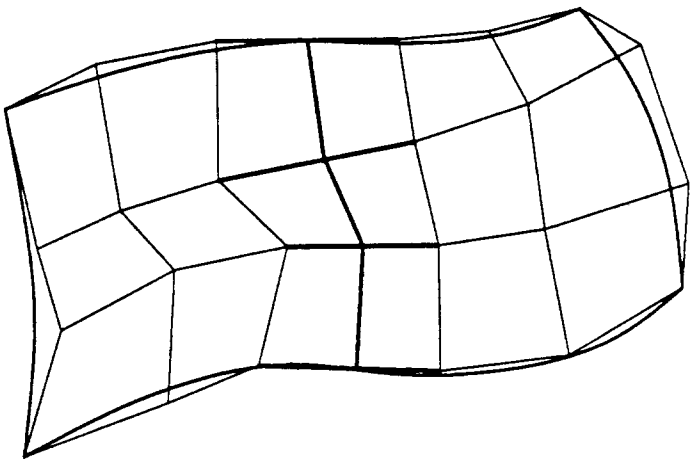


Figure 2.11: ***Parametrically continuous Bézier patches.***

In practice the restrictions imposed by parametric continuity are too severe. Indeed, it is clearly impossible to fit Bézier patches into a network of curves in space, if these curves are not C^1, but only have the same tangent plane at their intersection points. For this situation, we must be satisfied with with the more general notion of *geometric continuity*.

2.4.2. Geometric Continuity

Geometric continuity has been introduced by Barsky [3] and then elegantly generalized in [7, 32]. Geometric continuity is an extension of parametric continuity and is based on the shape of a curve or a surface, rather than on their parametrizations. It is designed to relax the unnecessary restrictions imposed by parametric continuity.

Consider Fig. 2.12. It shows a pair of planar curves $\mathbf{c}_1(u) = (u,0)$, $u \in (-\infty,0]$ and $\mathbf{c}_2(u) = (2u,4u^2)$, $u \in [0,\infty)$. These two curves meet at the parametric value of $u = 0$. Although the curves appear to meet smoothly, they are not C^1 continuous at $u = 0$ since $\mathbf{c}_1'(0) = (1,0)$ and $\mathbf{c}_2'(0) = (2,0)$. However, the second curve can be reparametrized by the substitution $2u \rightarrow u$ yielding $\mathbf{c}_2(u) = (u,u^2)$. Under this parametrization, the two curves become C^1, although their shape has not changed at all!

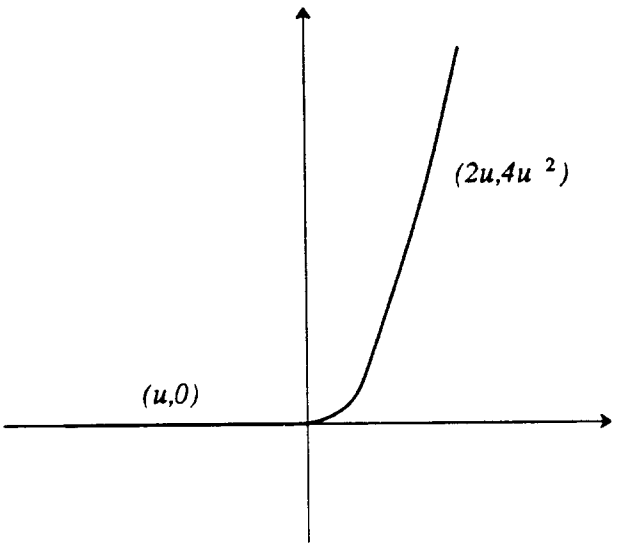


Figure 2.12: *Two geometrically, but not parametrically continuous curves.*

The notion of geometric continuity was introduced to deal with these situations. For many CAGD applications, the parametrization of a curve or a surface is not important, only their shape is. Then, geometric continuity is clearly preferable over its parametric counterpart.

2.4.2.1. Characterizations of Geometric Continuity

Three equivalent definitions of geometric continuity of order n for curves have been given by Barsky and DeRose [8, 9]. We briefly summarize these definitions.

Parametrization Independence. Let $\mathbf{p}(t)$ and $\mathbf{q}(u)$ meet at a common point. They are said to meet with G^n continuity if there exists a reparametrization of $\mathbf{q}(u)$, $u \rightarrow \tilde{u}$ such that $\mathbf{p}(t)$ and $\mathbf{q}(\tilde{u})$ are C^n at the common point.

This definition is not constructive in the sense that it does not show how to check if the two curves are G^n. This can be accomplished by using Beta-constraints.

Beta-Constraints. Let $\mathbf{p}(u)$ and $\mathbf{q}(u)$ meet at a common point:

$$\mathbf{p}(t_0) = \mathbf{q}(u_0).$$

Then they meet with G^n continuity at this point, if there exist numbers $\beta_1, \ldots, \beta_n$, such that

$$\mathbf{p}^{(i)}(t_0) = \sum_{j=1}^{i} CR_{ij}(\beta_1, \ldots, \beta_i) q^{(j)}(u_0), \quad i = 1,\ldots,n. \tag{2.62}$$

The notation CR_{ij} refers to chain rule; if $\mathbf{p}(t)$ and $\mathbf{q}(\bar{u})$ are C^n under reparametrization $u \rightarrow \bar{u}$, then

$$\beta_j = \frac{d^j \bar{u}}{du^j}(u_0), \quad j = 1,\ldots,n. \tag{2.63}$$

For example, the constraints for G^3 continuity are:

$$\mathbf{p}^{(1)}(t_0) = \beta_1 \mathbf{q}^{(1)}(u_0), \quad \beta_1 > 0 \tag{2.64}$$

$$\mathbf{p}^{(2)}(t_0) = \beta_1^2 \mathbf{q}^{(2)}(u_0) + \beta_2 \mathbf{q}^{(1)}(u_0) \tag{2.65}$$

$$\mathbf{p}^{(3)}(t_0) = \beta_1^3 \mathbf{q}^{(3)}(u_0) + 3\,\beta_1 \beta_2 \mathbf{q}^{(2)}(u_0) + \beta_3 \mathbf{q}^{(1)}(u_0) \tag{2.66}$$

This definition is very well-suited for deriving geometric continuity constraints for B-spline and Bézier curves. For example, Beta splines [3, 5] are generalizations of B-splines: continuity at the knots is geometric, rather than parametric.

Arc Length Parametrization. The last characterization of geometric continuity is based on arc length parametrization. The two curves meet with G^n continuity if and only if their arc length reparametrizations meet with C^n continuity.

This definition provides a geometric insight into the nature of geometric continuity. Indeed, two curves will be G^1 if their unit tangent vectors are continuous, because the unit tangent is the first derivative with respect to arc length. Analogously, two curves will be G^2, if they have a continuous curvature vector (the second derivative with respect to arc length).

2.4.2.2. Geometrically Continuous Surfaces

The last definition of geometric continuity is especially important as it can be easily extended to surfaces. Indeed, the two surface patches are said to meet with a G^1 continuity at their common boundary, if, for any point on the boundary, the two tangent planes of the two patches are the same [39, 68, 106, 126]. This can also be expressed as a continuity of a unit normal vector across the whole surface.

Analogously, the G^2 condition for surfaces is expressed in terms of the *osculating paraboloid*, or the continuity of the curvature vector [14, 76]. In this work, however, we will be dealing with G^1 surfaces only. Various methods of modeling with G^1 Bézier and Gregory patches will be discussed in Chapter 6.

3

Procedural Interpolation with Pleasing Splines

In this chapter, a local interpolation method for curves in $\mathbf{R}^2$ or $\mathbf{R}^3$ offering G^1 continuity is described. A curve is represented as a union of geometrically continuous cubic Bézier segments between each pair of adjacent vertices. At each interpolation point our procedure determines a tangent direction and two derivative magnitudes on either side of the vertex. The method uses an intuitive geometric, rule-based approach to find a 'good' default solution that produces pleasing-looking results even for highly irregular sets of data. Various spline properties and their relevance to our method are also discussed.

3.1. Motivation

Much work has been done in the area of interpolating splines, i.e. fitting smooth curves and surfaces through a given set of points in a plane or in space. Several textbooks deal with these issues [10, 43, 46], and hundreds of papers have been published on the subject.

From this abundance of available interpolation schemes, it is often very difficult for a designer to choose a method that will be best suited to his particular needs. In this chapter, we attempt to confront the problem of interpolation not from a mathematician's, but from a designer's point of view. We start by looking at various spline properties that are desirable in a practical modeling system. From this set of properties we formulate the mathematical definition of the spline in a procedural manner.

The procedural approach is based on geometric rather than algebraic reasoning. The curve is derived from a sequence of procedural steps [121] that may include special rules that take into account the special situation of a particular segment. This method is more flexible than a conventional algebraic approach that applies a fixed arithmetic expression to a set of interpolation vertices.

An immediate benefit of this approach is the fact that *pleasing* curves can be produced even for irregular sets of interpolation points. Figure 3.1a shows a Catmull–Rom spline [23] through an asymmetric set of points. Figure 3.1b shows our pleasing spline through the same points.

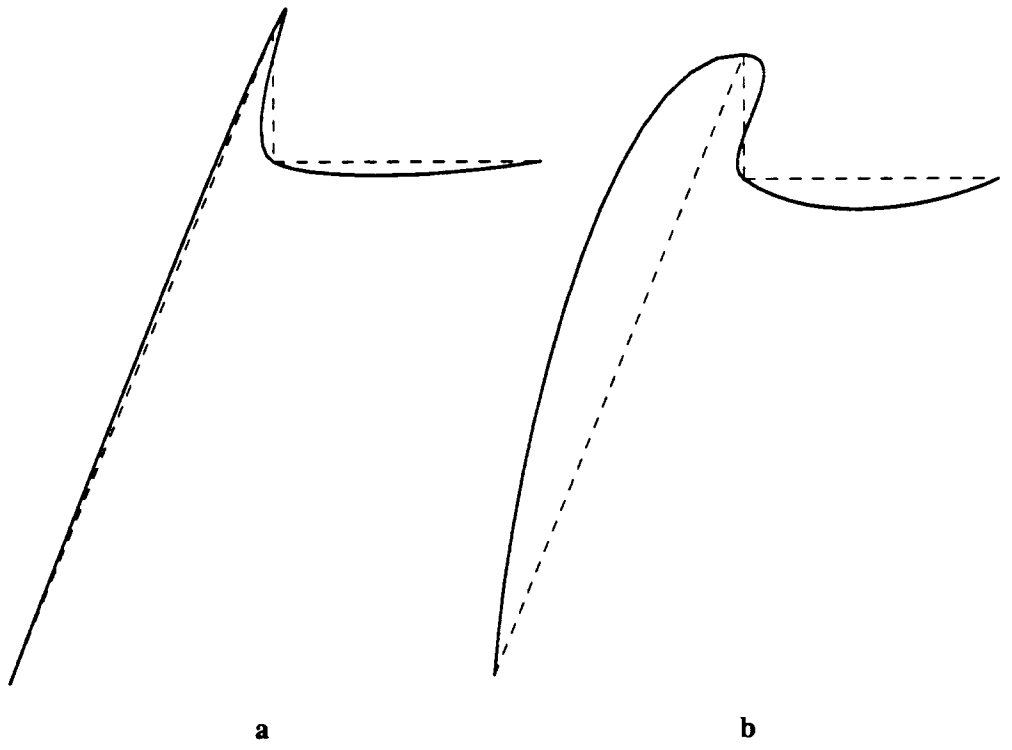


Figure 3.1: a. *Catmull–Rom spline.* **b.** *Procedural pleasing spline. Defining polygon is shown dashed.*

3.2. Useful Spline Properties

3.2.1. Geometric Properties

Since the concept of *geometric continuity* [3, 7, 32] was first introduced, geometric splines are becoming increasingly popular. They are very well suited to computer aided geometric design because only their shape and not their parametrization is important. In other words, only geometric or *visual* continuity [40] (usually of first or second order) is required at the joints of a

spline. This keeps the spline visually smooth and at the same time provides extra degrees of freedom since geometric continuity is less restrictive than its parametric counterpart. Modeling with geometric splines is well described in the literature [6, 17, 68].

Another important geometric property that is clearly necessary in a modeling system is *coordinate system independency*. Most of the interpolating and approximating splines have this property. All such splines are also *invariant under rotation, translation, and uniform scaling*. In fact, many interpolation schemes treat the coordinates of the curve completely independently of each other, which leads to invariance under differential scaling (different scaling ratios for different coordinate axes) as well. As we will see later, this property is actually *undesirable* for pleasing splines.

3.2.2. Local Control

One of the pioneering approaches to the interpolation problem was to represent the whole curve with a single polynomial. For example, Lagrange interpolation [107] could be used resulting in a polynomial of relatively high degree. Alternatively, one could build a continuous spline with piecewise polynomial pieces of some fixed order. A *natural cubic spline* minimizes the square of its second derivative componentwise [1, 43]. Alternatively, *physical* splines minimize some physical property of the spline, such as bending energy or curvature [53, 80]. All these methods have the serious disadvantage that they are *global*. In other words, a change in a position of any vertex typically results in a change in the whole curve. Usually, a system of linear equations has to be solved anew each time a change occurs, which leads to inefficient implementation.

However, often only a small part of a curve needs to be changed and one would then like the bulk of it to remain unaffected. Thus, the property of *local control* is crucial for a modeling system. Indeed, most of the recent research has concentrated on *local splines*. The ability to *fine-tune* the curve near a certain point is often associated with local control. As mentioned above, geometric continuity is less restrictive than parametric continuity. Therefore, it is possible to use *shape parameters* for additional control of the curve. The use of shape parameters in Barsky's Beta-splines [3, 4, 5, 54, 102] has been widely publicized. Moreover, weights in rational curves can also be used as shape parameters [6, 63].

It is desirable that these shape parameters have a clear geometrical interpretation. If this is the case, a designer will be able to correlate the change in the shape of the curve with the change of a corresponding parameter in an *intuitive manner*. For example, a tangent direction or a normal to a curve at a certain point will be a good shape parameter. It is also important that the shape parameters have good *default values* that would make the curve look pleasing to a human eye. Finding the default values will be a crucial part of our spline construction process.

3.2.3. Simplicity of Representation

Simplicity of representation is obviously important for *efficiency* and for realtime interactive display. Various mathematical formulations for splines have been proposed. These formulations can be classified into polynomial, rational, and others.

Rational splines have recently gained a lot of attention due to the fact that they can represent conic curves exactly. Much has been published on modeling with rational curves, starting from simple schemes using just circular arcs and linear segments [72, 104] or rational conics [36, 60, 103, 108] to higher degree rational curves [6, 16, 50, 74, 75, 105]. It is also possible to define a spline that cannot be represented as a ratio of two polynomials. Many tension-controlled splines [28, 118] are of this form.

However, piecewise polynomial splines usually are appropriate for many applications. Although quadrics have been used for interpolation [20, 95], it is generally agreed that the piecewise cubic form is the simplest yet versatile enough spline representation suited for most applications. The review [13] has a good survey on the use of cubics in CAGD. In our method, we will use Bézier cubic segments.

3.2.4. Stability and Consistency

Simplicity is often associated with *numerical stability*. By stability we mean that the curve must depend on the set of interpolating vertices in a continuous manner. In other words, a small change in a vertex position should result in a small change in the curve.

The concept of consistency is closely related to stability. It has been discussed recently by Ohlin [94]. Loosely speaking, the introduction of an additional interpolating point near the curve should result in a small change in the curve. In the limit, if the new vertex lies exactly on the curve, the new resulting curve should be the same as the original one. For example, a piecewise linear 'spline' is consistent. Certain minimal energy splines are also consistent.

This obviously attractive property is hard to attain with practical splines because one changes an n point interpolation into a $n + 1$ point problem. Ohlin has solved this problem by introducing a non-polynomial spline that minimizes a certain function of curvature parametrized by arclength. However, the resulting spline is also global, and therefore does not satisfy at least two of the properties listed above. Section 3.6.3 will further discuss the difficulties of modeling with consistent splines.

3.2.5. Shape Preservation

It is usually desirable for the shape of the spline to resemble in some sense the shape of the polygon of interpolation points. For example, if the polygon is symmetric with respect to some axis, one would also like the curve to be symmetric. Two basic forms of shape preservation have been studied in the literature: monotonicity preservation and convexity preservation.

Monotonicity preservation only makes sense if the data arises from a function. In this case, usually the x component of the vertices is strictly increasing, and so one can regard the y component of a curve as a function of x. A spline is then called monotonicity preserving, if its behavior on a certain interval is dictated by the corresponding sequence of the y coordinates of the vertices. For example, the spline should increase, if the y coordinates of the vertices increase. A precise definition of this property as well as more discussion on monotonity preserving can be found in the literature [52,97,117]. However, as we will see in Section 3.6.1, this property implies a dependence on a coordinate system.

A spline is called *convexity preserving* if it is convex on an interval where the corresponding sequence of vertices is convex [21,37,55,89]. In this case, the number of inflections in the spline will match the number of inflections in the interpolation set. This property is important for a practical design system and can be viewed as a requirement for good visual quality of the spline as discussed below.

3.2.6. Visual Quality

Visual quality, or *pleasingness*, is fundamentally different from the above properties in the sense that it cannot be described in precise mathematical terms. Conceptually, a 'pleasing spline' is a spline that looks good to the human eye. This 'definition' is, of course, very subjective as it is impossible to construct a curve to everyone's liking. However, this property is very important to designers, as they would like to get pleasing forms even for irregular or strongly asymmetric sets of interpolation vertices.

Some researchers have linked pleasingness to the *fairness* of curves as defined by naval architects or car hull designers [44,48,71]. In this context fairness seems to be related to the continuity of the second and even third derivatives and thus require G^2 or even G^3 continuity. We give up the strict requirements of the G^2 continuity to handle cases with strictly linear curve segments, but concentrate instead on avoiding large, asymmetric *overshoots* or *inappropriate bulges* in the curve (see Fig. 3.1). Admittedly, the method to be described produces what *we* consider good default solutions to many situations. The user, however, will have the freedom to change one or several of the values that determine curve behavior.

3.3. Determination of Local Shape Parameters

3.3.1. Geometric Parameters

As mentioned above, we have chosen Bézier cubics to represent our splines because of their simplicity, wide acceptance, and ease of local control. A cubic Bézier segment between two given endpoints will be completely defined if a *derivative* is chosen at each point. This process is often decomposed into two steps: first, a *tangent direction* is selected at each endpoint, and then the inner two control points of the Bézier curve are placed on these two tangent lines at a suitable

distance from the endpoints to give the wanted amount of 'bulge' to the curve segment (Fig. 3.2). G^1 continuity is ensured as long as the Bézier points on either side of an original vertex lie on the same tangent line. There are, therefore, three geometric parameters, associated with each given point that completely determine the curve:

- A *normal* direction. The curve tangent at the vertex P_i will be perpendicular to this normal $\mathbf{n}_i$.
- Two *velocities*. The velocities at either side of the vertex determine the magnitude of the two derivatives or, in the other words, the location of the two nearest Bézier points. For a C^1 continuous curve, these two velocities must be equal; for G^1 continuity, their collinearity is sufficient. The velocity magnitudes v_{ip} and v_{is} determine the position of the predecessor C_{ip} and the successor C_{is} Bézier control points of resulting curves:

$$\mathbf{C}_{ip} = \mathbf{P}_i + \frac{1}{3}\,\mathbf{v}_{ip}, \quad \mathbf{C}_{is} = \mathbf{P}_i + \frac{1}{3}\,\mathbf{v}_{is}. \tag{3.1}$$

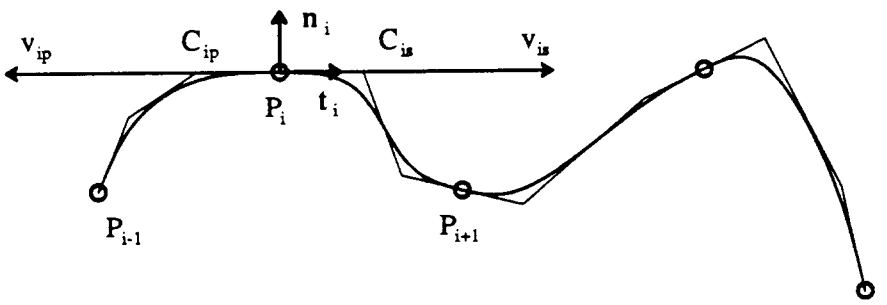


Figure 3.2: *A chain of Bézier segments.*

The construction of the interpolating curves proceeds as follows:

1. Two neighboring vertices are identified; there will be an interpolating curve segment that has these neighbors as its endpoints. This step is usually performed by the user, either by supplying a *defining polygon* $P_0, \ldots, P_i, \ldots, P_k$ or an ordered list of points.
2. A suitable tangent direction or a normal is selected at each vertex.
3. Suitable velocities are chosen on either side of the vertex, or, equivalently, the Bézier points are placed along the tangent line of each vertex.

A G^1 curve can be constructed for an arbitrary choice of a normal and of two velocities at any given point. Indeed, the fact that these shape parameters can be modified to fine-tune the shape of the spline is widely known [31,47,79,88]. However, virtually no literature exists on how to choose these shape parameters to guarantee good visual results. The next two sections will concentrate on the best default values for normals and velocities. We will first investigate methods

that depend only on a vertex P_i and its two nearest neighbors.

3.3.2. Determination of the Normal

We start by looking at several simple methods of normal determination. Then we discuss their advantages and shortcomings. Finally, we analyze various extensions and combinations of these methods with the goal of improving the behavior of the normal.

3.3.2.1. Primitive Methods

Catmull–Rom (Directly Weighted Normals) Method. Catmull and Rom [23] define the tangent direction of the curve at the point P_i to be the direction of the chord between the two nearest neighbor vertices:

$$\mathbf{t}_C = \mathbf{P}_{i+1} - \mathbf{P}_{i-1}. \tag{3.2}$$

It can be readily verified that the normal direction $\mathbf{n}_C$ can also be expressed as a direct weighted average of the two unit normals $\hat{\mathbf{n}}_b$ and $\hat{\mathbf{n}}_b$ (shown dotted in Fig. 3.3) of the underlying edges of the defining polygon, with the weight factors equal to the length of these edges:

$$\mathbf{n}_C = \mathbf{n}_{DN} = p\ \hat{\mathbf{n}}_p + s\ \hat{\mathbf{n}}_s. \tag{3.3}$$

This method may lead to a lopsided bulge if the two sides of the defining polygon form an acute angle and are different in length (Fig. 3.3a).

Inversely Weighted Edges. It is possible to achieve similar results by using the weighted average of the polygon edges themselves, rather than the normals. Since Catmull–Rom turns the resulting normal away from the shorter side, we choose an indirect weighting:

$$\mathbf{n}_{IE} = s\ \hat{\mathbf{p}} + p\ \hat{\mathbf{s}}. \tag{3.4}$$

Here $\hat{\mathbf{p}}$ and $\hat{\mathbf{s}}$ are unit vectors in the directions of the edges of the defining polygon (shown dashed). However, this method has the problem that when the two edges are collinear, $\mathbf{n}_{IE}$ will also be collinear with them. Fig. 3.3b illustrates this on a polygon with an obtuse angle.

Inversely Weighted Normals. One could argue that two closely spaced points on a control polygon convey strong information about the tangent direction near that location and that the normal direction should thus be more strongly influenced by the *shorter* side of the two sides. This can be easily achieved by inversely weighting the unit normals of the two sides (Fig. 3.3c):

$$\mathbf{n}_{IN} = s\ \hat{\mathbf{n}}_p + p\ \hat{\mathbf{n}}_s. \tag{3.5}$$

Directly Weighted Edges. Again, similar result can be achieved by using the directly weighted average of the polygon edges (Fig. 3.3d):

$$\mathbf{n}_{DE} = p\ \hat{\mathbf{p}} + s\ \hat{\mathbf{s}}. \tag{3.6}$$

Similarly to the Inversely Weighted Edges, this method fails for the case of collinear edges.

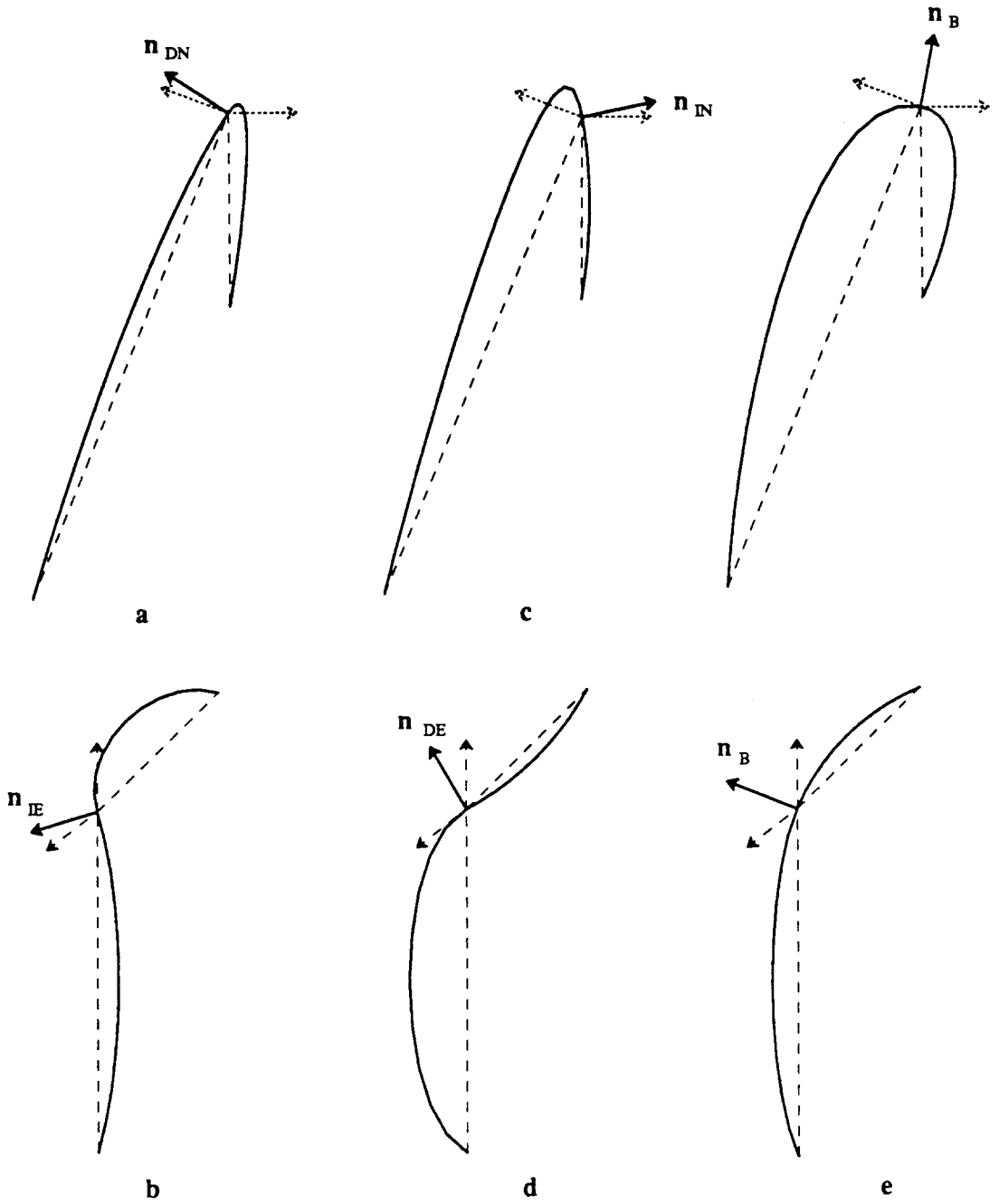


Figure 3.3: *Primitive methods for normal determination.*
a. *Direct Normals.* **b.** *Inverse Edges.*
c. *Inverse Normals.* **d.** *Direct Edges.* **e.** *Angle Bisector.*
The edge normals are shown dotted, and the extensions of polygon sides are dashed.

Angle Bisector. A good compromise between the two sets of the approaches above is to make the normal independent of the length of the underlying side, i.e. to set it to the direction of the angle bisector at the vertex (Fig. 3.3e):

$$\mathbf{n}_B = \hat{\mathbf{n}}_p + \hat{\mathbf{n}}_s = \frac{\mathbf{n}_{DN} + \mathbf{n}_{IN}}{p + s} = \frac{\mathbf{n}_{DE} + \mathbf{n}_{IE}}{p + s}. \tag{3.7}$$

Other Methods. Other methods of choosing the normal direction are also possible. For example, one can define the normal in the direction between the midpoint of the chord between the two neighbors and the current vertex. Similarly to the methods based on edge averaging (see above), this method works reasonably well for acute angles, but fails completely if the two edges are collinear.

Another approach to normal selection may utilize a circle that passes through the current point and its two neighbors. The vertex normal is then defined to be the normal to the circle at the interpolation point. The resulting normal in this case is close to $\mathbf{n}_{IN}$ because of strong influence of closely spaced points, but it is computationally more expensive.

3.3.2.2. Pleasing Normals

As we see, the Angle Bisector method is an average of two opposite weighting procedures. It produces pleasing curves for practically all test cases. Therefore, the Angle Bisector normal is a good and robust default choice for the normals at all interpolation points.

In Fig. 3.4, the deviation of various normal methods from the Bisector normal is graphed. The horizontal axis reflects the magnitude of the angle $P_{i-1}P_iP_{i+1}$, while the vertical axis corresponds to the angle between the Bisector normal and other normals. The four graphs form a very symmetric figure; indeed, the normals $\mathbf{n}_{DN}$ and $\mathbf{n}_{IN}$ as well as $\mathbf{n}_{IE}$ and $\mathbf{n}_{DE}$ are symmetric with respect to the horizontal axis. Also, $\mathbf{n}_{DN}$ and $\mathbf{n}_{IE}$ as well as $\mathbf{n}_{IN}$ and $\mathbf{n}_{DE}$ are symmetric with respect to the vertical axis through an angle of 90° at P_i.

The actual functional dependence of the deviation of a particular normal method from the Bisector normal can be easily derived from the definitions of normal methods (3.3–3.7). For example, the angle δ between $\mathbf{n}_B$ and $\mathbf{n}_{DN}$ can be expressed as:

$$\delta = \arccos\left(\frac{\mathbf{n}_{DN} \cdot \mathbf{n}_B}{|\mathbf{n}_{DN}|\,|\mathbf{n}_B|}\right) = \arccos\left(\frac{(p+s)\sqrt{1+\hat{\mathbf{n}}_p \cdot \hat{\mathbf{n}}_s}}{\sqrt{2}\sqrt{p^2+s^2+2\,p\,s\,\hat{\mathbf{n}}_p \cdot \hat{\mathbf{n}}_s}}\right). \tag{3.8}$$

The angle between $\mathbf{n}_B$ and the other normals can be easily deduced from symmetry considerations.

The graphs in Fig. 3.4 correspond to the case when the ratio of the two sides s and p is 10. The larger this ratio is, the further the graphs will deviate from the horizontal axis for intermediate values of the polygon angle, but they will never extend beyond the straight lines through their endpoints, as approximated by the curve for $r = 100$. The intersection point of the deviations of $\mathbf{n}_{DN}$ and $\mathbf{n}_{IE}$ (and $\mathbf{n}_{IN}$ and $\mathbf{n}_{DE}$) will always correspond to the angle of 90° at P_i; the deviation of

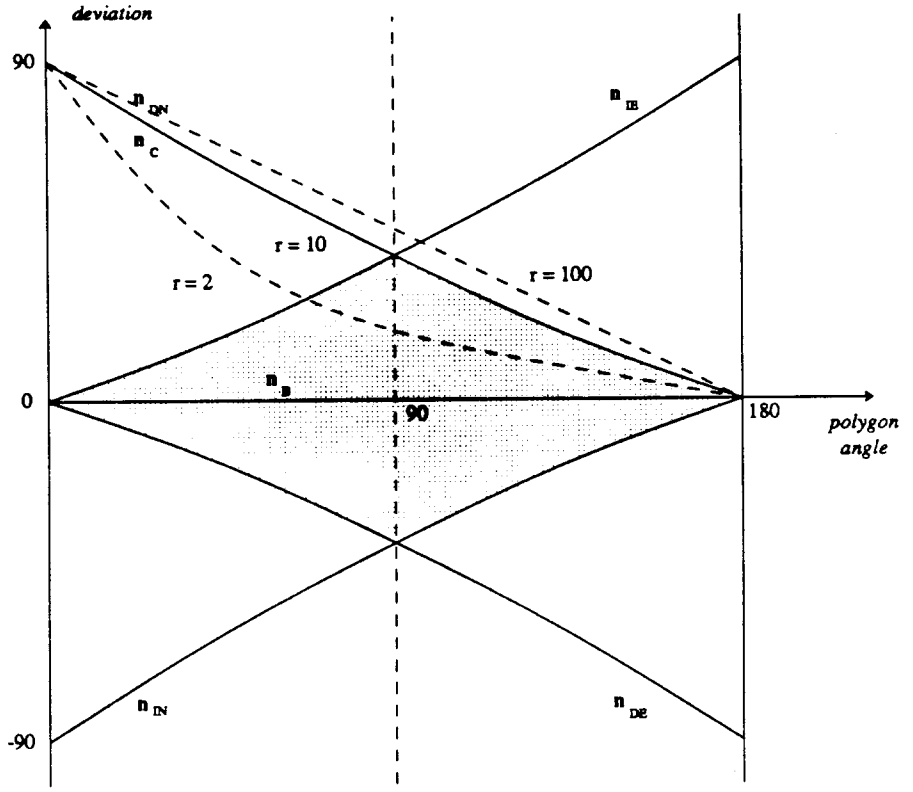


Figure 3.4: *Deviation of the Bisector normal from other normals.*
The deviation of Catmull–Rom normal is shown for various ratios r *of the lengths of adjoining sides. The shaded area corresponds to a pleasing choice of the normal for* r = 10.

all the normals from the bisector in this case will never exceed 45°.

The shaded area on both sides of the horizontal axis corresponds to a *pleasing* range of normals at any interpolation point. Indeed, in this case, the normal will always lie between the normals of the two adjoining sides as well as between the extended edges of the defining polygon. Intuitively, normals from outside of the pleasing range would tilt 'too much' to one of the polygon sides. This explains, for example, why the Catmull–Rom normal is not adequate for acute angles.

Fig. 3.5 illustrates the usage of various normal methods on two sets of interpolating points.

3.3.2.3. Control of the Normal by the Shape Parameter

Within the shaded area of Fig. 3.4 it is difficult to argue that one choice gives overall better results than another; this is largely a matter of taste. We thus capture the *deviation* of the normal from its default Bisector position by the shape parameter D. Changing D from 1 to -1 would change the normal from the top to the bottom of the pleasing region in Fig. 3.4. A default value

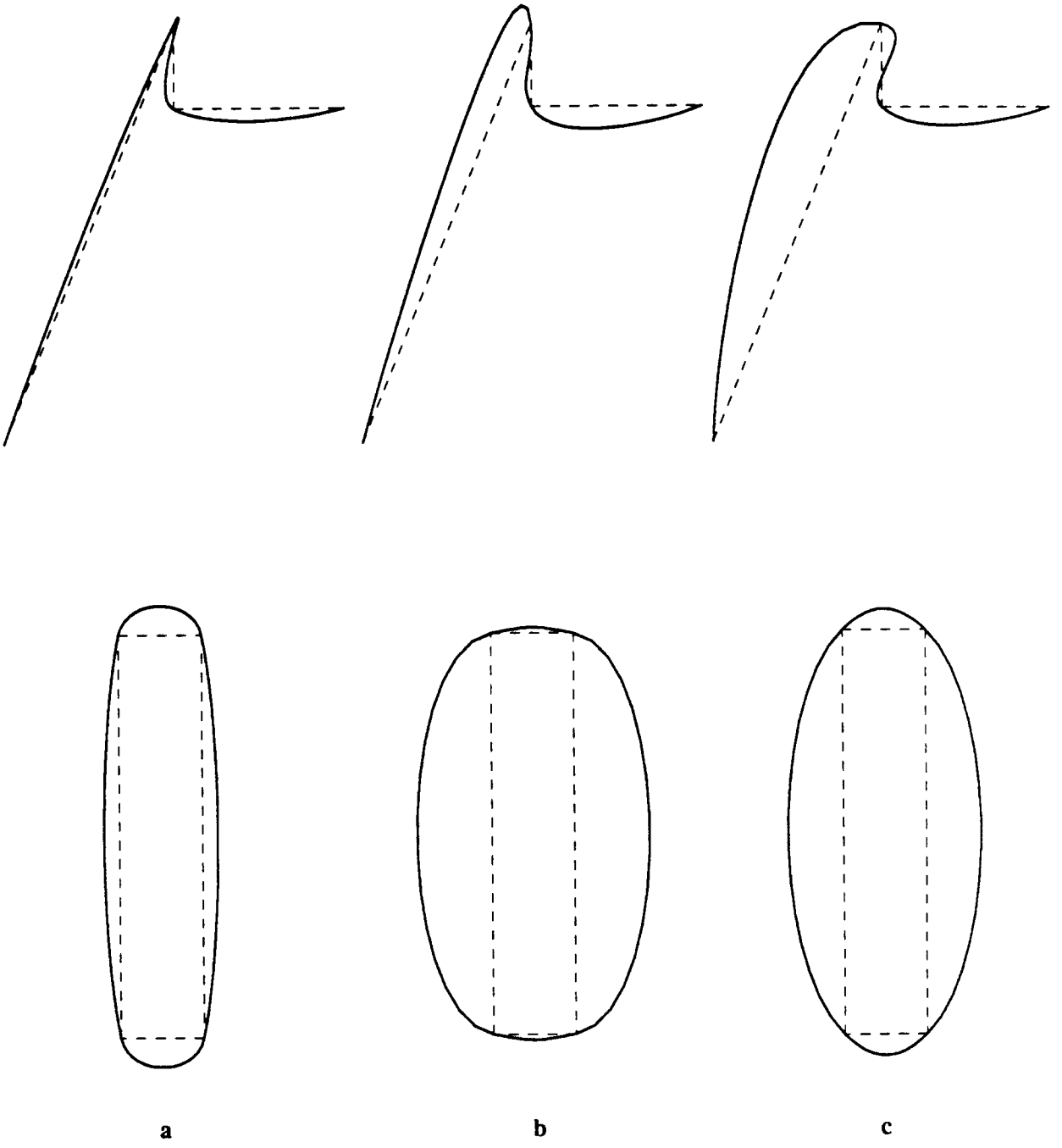


Figure 3.5: *Comparison of various normal methods.*
a. *Direct Normals.* **b.** *Inverse Normals.* **c.** *Bisector.*

of $D = 0$ will simply result in the Bisector normal.

For a nonzero deviation parameter, the final normal will be computed as a weighted average of $\mathbf{n}_{IE}$ and $\mathbf{n}_{DE}$ for acute angles, and as a weighted average of $\mathbf{n}_{DN}$ and $\mathbf{n}_{IN}$ for obtuse angles:

$$\mathbf{n} = \begin{cases} (\mathbf{n}_{IE} + \mathbf{n}_{DE}) + D\ (\mathbf{n}_{IE} - \mathbf{n}_{DE}) & \text{if angle at } P_i \le 90° \\ (\mathbf{n}_{DN} + \mathbf{n}_{IN}) + D\ (\mathbf{n}_{DN} - \mathbf{n}_{IN}) & \text{if angle at } P_i \ge 90° \end{cases} \tag{3.9}$$

This global parameter permits one to tune the system's behavior to the taste of the designer or to particular needs of an application. After default normals at all vertices have been determined, a user will have the freedom to modify the normal direction individually at selected points. Moreover, *global* changes in default normal construction can be provided, e.g. tilting *all* the normals towards the shorter polygon side.

3.3.3. Velocity Computation

3.3.3.1. Primitive Methods

Once a normal, or, equivalently, a tangent direction at each vertex has been determined, the task remains to choose two suitable velocities at each given point. As in the case of the normals, we first evaluate the performance of several elementary methods. For the illustrations in this Section we have assumed that the normals have been defined with the Angle Bisector method.

Catmull–Rom Method. Since Catmull–Rom splines are C^1 continuous, the velocities on either side of the vertex are the same. In this approach, they are defined to be one half of the length of the chord between the previous and the next vertex:

$$v_{p,C} = v_{s,C} = c, \tag{3.10}$$

where $2c$ is the length of the chord between P_{i-1} and P_{i+1} (Fig. 3.6a).

This method has the disadvantage that, if the ratio of two adjoining sides is large, the curve may 'overshoot' or bulge too much on the side of the shorter polygon edge. For example, oblong shapes or even curves with self-intersections are produced for rectangles with large aspect ratios.

Edge Length Method. In this method, the velocity on either side is just made equal to the edge length. This corresponds to placing the Bézier control point 1/3 of the edge length away from the vertex on the established tangent line:

$$v_{p,L} = p, \quad v_{s,L} = s. \tag{3.11}$$

However, for very acute angles between the two sides, this method creates strongly bulging curves (Fig. 3.6b). Furthermore, there is a large discontinuity in velocity and there is no blending effect from one side to the other.

Projection Methods. This problem can be avoided if, instead of the side length, the lengths of the *orthogonal projection* p' and s' of the sides onto the tangent direction are used:

$$v_{p,P} = p', \quad v_{s,P} = s'. \tag{3.12}$$

It is also possible to use the length s'' or p'' between the current vertex and the intersection point of the tangent direction and the perpendicular to the corresponding side (Fig. 3.7):

$$v_{p,P} = p'', \quad v_{s,P} = s''. \tag{3.13}$$

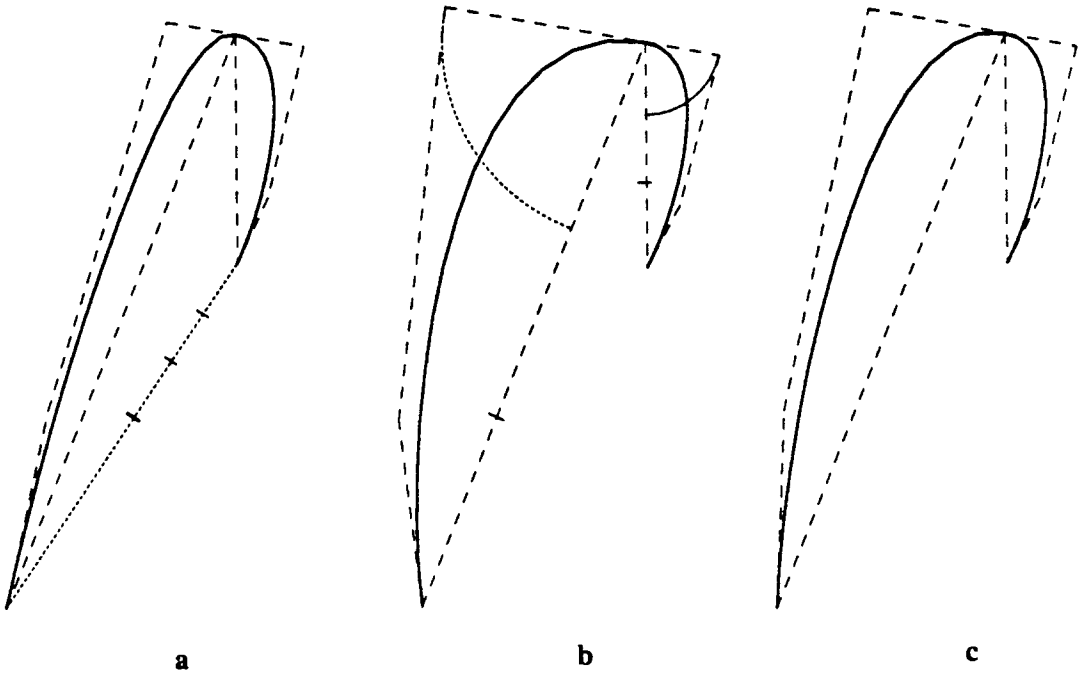


Figure 3.6: *Primitive methods for velocities.*
a. *Catmull–Rom.* **b.** *Edge Length.* **c.** *Average of Catmull—Rom and Edge Length.*

The last two methods may result in extremal velocity values: if the tangent direction is perpendicular to the edge of the defining polygon, the first method would produce unacceptable zero velocity, while the second would produce an equally unacceptable 'infinite' velocity.

Average Method. A good compromise can be reached if the average between Catmull–Rom and Edge Length methods for the velocity is used (Fig. 3.7c):

$$v_{p,CL} = \frac{1}{2}(p + c), \quad v_{s,CL} = \frac{1}{2}(s + c). \tag{3.14}$$

Certainly, velocities at the interpolation points can be defined in many different ways. We now attempt to classify the simplest methods and evaluate them for visual quality of the resulting curves.

3.3.3.2. A Catalog of Velocity Methods

Consider Fig. 3.7. In this figure, two sides p and s of the defining polygon adjacent to the vertex P_i are shown. The velocity v_p always depends on p; however, it may or may not depend on s. Analogously, v_s always depends on s, but it may not depend on p. Thus, all the velocity methods can be classified into two groups: ones that look only at the underlying side of the defining polygon, and the ones that look at both adjacent sides.

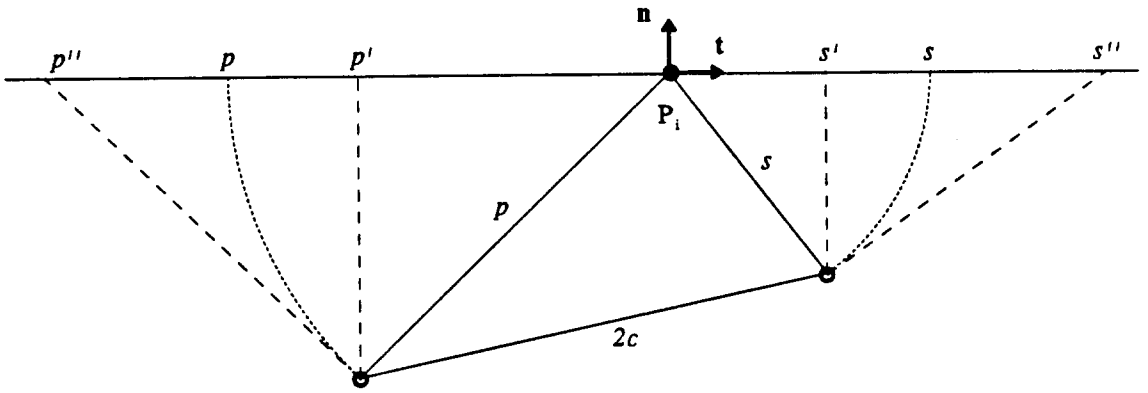


Figure 3.7: *Constructions for velocity computations.*

One-sided Velocities. The velocities in this group depend just on a single underlying side of the defining polygon. We will consider only three simple methods of this kind that yield reasonable results:

1. **Projection Method:**

$$v_{p,P} = p', \quad v_{s,P} = s'. \tag{3.15}$$

2. **Edge Length Method:**

$$v_{p,L} = p, \quad v_{s,L} = s. \tag{3.16}$$

3. **Average of Projection and Edge Length Methods:**

$$v_{p,PL} = \frac{1}{2}(p + p'), \quad v_{s,PL} = \frac{1}{2}(s + s'). \tag{3.17}$$

The fact that the adjoining side is ignored in the computation may lead to a large discrepancy in the bulges of the curve on the two sides, if the two adjoining polygon sides vary greatly in length. This fact can be characterized quantitatively in terms of a large jump of curvature value at the corresponding vertex.

Two-sided Velocities. The above problem can be avoided by eliminating the discontinuity in the curve derivative and making the curve *parametrically* (C^1) continuous. This corresponds to forcing the two velocities to be equal independent of the ratio of the underlying sides. Catmull–Rom splines are a classical example of this approach; however, other similar methods are possible:

1. **Length Average.** One could define the velocity just by averaging the underlying sides:

$$v_{p,LA} = v_{s,LA} = \frac{1}{2}(p + s). \tag{3.18}$$

2. **Projection Average.** Alternatively, the average of the projections of the sides onto the tangent direction can be used:

$$v_{p,PA} = v_{s,PA} = \frac{1}{2}(p' + s'). \tag{3.19}$$

3. **Catmull–Rom Method.** As we have already seen, the velocity on either side is defined by one half of the length of the chord between the previous and the next vertex:

$$v_{p,C} = v_{s,C} = \frac{1}{2}c. \tag{3.20}$$

Again, for large ratios of the lengths of adjoining sides, the C^1 methods have a common drawback that may be loosely described as a 'sharp turn' of the curve on the shorter side due to too large a velocity there. In mathematical terms this means that the curvature of the spline in the middle of the corresponding segment has a large absolute magnitude.

Reduction of the C^1 Component. For large aspect ratios of the two adjoining polygon sides, the Catmull–Rom velocity v_C and even the average velocity v_{CL} can exceed the length of the shorter side. This may result in loops in the curve. The original Catmull–Rom method usually does not have this problem because the tangent direction always lies closer to the longer polygon side, but it creates large asymmetric overshoots (Section 3.3.2.1).

If, however, the user is free to change the tangent direction, additional care must be taken to restrict the C^1 component of a particular velocity method. We limit it to the length s'' between the vertex P_i and the intersection point of the tangent direction through P_i and the perpendicular to the shorter side s (Fig. 3.7):

$$v_C \le \min(v_{C,orig}, s'', p'') \tag{3.21}$$

and analogously for v_{LA} and v_{PA}. Thus, the distance between a vertex and the projection of an interior Bézier point onto the corresponding side will never exceed one third of the side length.

Combination of the Two Groups of Approaches. It seems natural to combine the two different groups of methods so that the disadvantages of a particular approach are minimized. We have constructed averages of each method from one group with each method from the other group, giving a total of 9 combination methods. We have tested all the 15 (9 combination and 6 primary) methods on a large set of defining polygons. Besides subjectively judging the aesthetic appearance of the resulting curves, we compared maximum curvature jumps at the interpolation points, the sum of these jumps for all the vertices, the difference of maximum and minimum curvature values for the whole polygon (curvature variation), and, finally, the so-called *energy integral* $\int \kappa^2 ds$ [51]. All the evaluations have been carried out for the default vertex normal based on the angle bisector. Fig. 3.8 illustrates the dependence of interpolating curves on selected velocity methods; Fig. 3.9 shows corresponding curvature plots. Finally, Tables 3.1abc compare all the velocity methods.

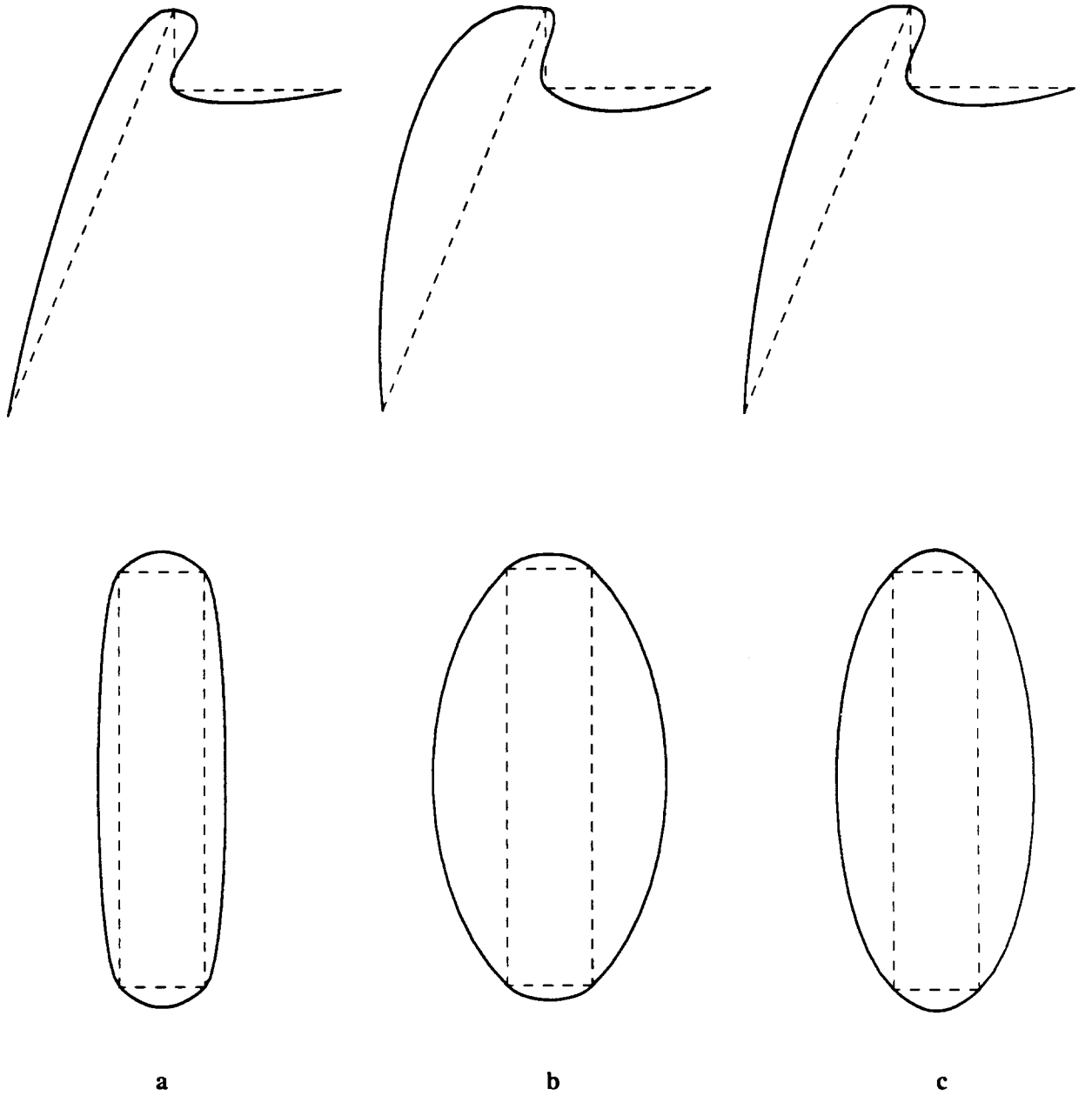


Figure 3.8: *Comparison of various velocity methods.*
a. *Catmull–Rom.* **b.** *Edge Length.* **c.** *Average of Catmull–Rom and Edge Length.*

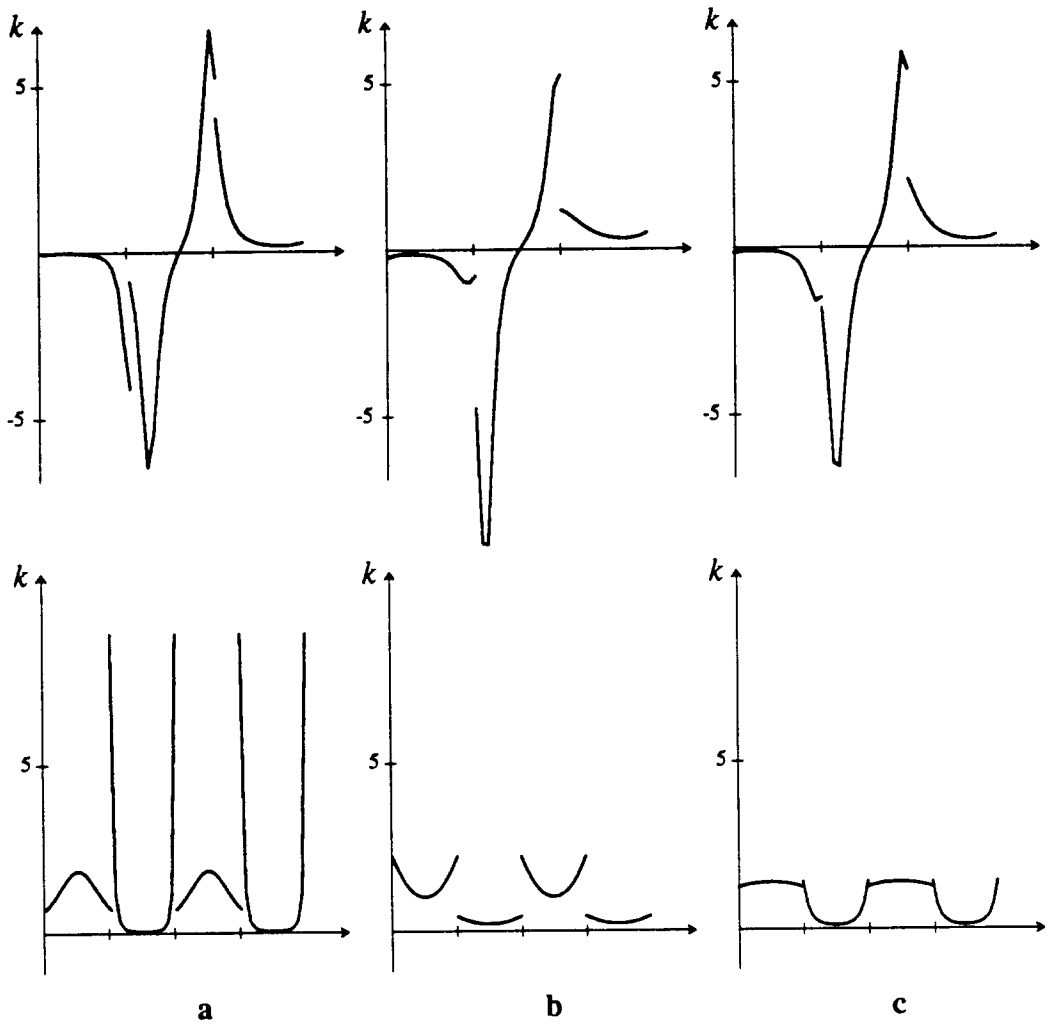


Figure 3.9: *Curvature plots of the curves from the previous Figure.*

Velocity Method	Maximum Curvature Jump	Sum of Curvature Jumps	Curvature Variation	Energy Integral
p	4.30	8.20	15.40	17.81
p'	115.22	121.40	154.78	301.66
$\frac{1}{2}(p+p')$	11.34	16.42	21.13	25.47
c	5.19	5.56	13.03	28.17
$\frac{1}{2}(p+s)$	2.50	3.58	13.94	24.17
$\frac{1}{2}(p'+s')$	74.37	75.70	93.36	483.42
$\frac{1}{2}(p+c)$	3.22	3.42	12.20	18.12
$\frac{1}{2}(p+\frac{1}{2}(p+s))$	2.81	3.34	12.93	18.23
$\frac{1}{2}(p+\frac{1}{2}(p'+s'))$	4.17	6.73	17.21	23.48
$\frac{1}{2}(p'+c)$	8.37	11.42	18.61	39.97
$\frac{1}{2}(p'+\frac{1}{2}(p+s))$	5.26	7.91	12.49	26.94
$\frac{1}{2}(p'+\frac{1}{2}(p'+s'))$	16.40	18.49	54.40	221.84
$\frac{1}{2}(\frac{1}{2}(p+p')+c)$	3.19	4.85	12.47	21.51
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p+s))$	2.77	4.38	12.60	19.72
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p'+s'))$	5.17	7.58	19.19	37.07

Table 3.1a: *Curvature comparisons for the top polygon from Fig. 3.8.*

From the above evaluation, we tend to conclude that the average of Catmull–Rom and Edge Length methods v_{CL} produces good results for all test polygons. Although in some cases other methods produced better results in terms of minimizing curvature jumps or variation, v_{CL} was generally very close to those 'best' results. We thus have accepted v_{CL} as a default choice for the velocities at the interpolation points.

Velocity Method	Maximum Curvature Jump	Sum of Curvature Jumps	Curvature Variation	Energy Integral
p	1.84	7.38	2.09	5.89
p'	4.64	18.56	5.67	10.42
$\frac{1}{2}(p+p')$	2.86	11.43	3.40	7.25
c	8.65	34.63	9.35	29.94
$\frac{1}{2}(p+s)$	8.65	34.63	9.35	29.94
$\frac{1}{2}(p'+s')$	3.84	15.39	5.16	29.94
$\frac{1}{2}(p+c)$	0.17	0.70	1.36	6.60
$\frac{1}{2}(p+\frac{1}{2}(p+s))$	0.17	0.70	1.36	6.60
$\frac{1}{2}(p+\frac{1}{2}(p'+s'))$	0.17	0.70	1.36	6.60
$\frac{1}{2}(p'+c)$	0.77	3.09	2.63	9.07
$\frac{1}{2}(p'+\frac{1}{2}(p+s))$	0.77	3.09	2.63	9.07
$\frac{1}{2}(p'+\frac{1}{2}(p'+s'))$	0.77	3.09	2.63	9.07
$\frac{1}{2}(\frac{1}{2}(p+p')+c)$	0.38	1.54	1.87	7.48
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p+s))$	0.38	1.54	1.87	7.48
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p'+s'))$	0.38	1.54	1.87	7.48

Table 3.1b: *Curvature comparisons for the bottom polygon from Fig. 3.9.*

Note that we are not trying to find a method that would minimize some curvature functional. Doing so would involve a global minimization problem that cannot be solved locally. Rather, we are trying to find a very simple approach that always produces reasonable results and can be used as a good starting default choice for possible subsequent fine-tuning or fairing.

We also have carried out the comparisons of various velocities methods for the two extreme choices of the normal direction, corresponding to the top and the bottom of the 'diamond' shape

Velocity Method	Maximum Curvature Jump	Sum of Curvature Jumps	Curvature Variation	Energy Integral
p	2.59	22.21	7.64	31.03
p'	18.24	51.49	47.35	121.46
$\frac{1}{2}(p+p')$	4.88	27.33	11.37	38.09
c	9.64	55.61	52.82	134.98
$\frac{1}{2}(p+s)$	9.52	49.68	18.09	51.48
$\frac{1}{2}(p'+s')$	21.17	51.30	46.97	164.04
$\frac{1}{2}(p+c)$	2.54	16.20	8.70	36.30
$\frac{1}{2}(p+\frac{1}{2}(p+s))$	3.28	18.08	6.46	32.43
$\frac{1}{2}(p+\frac{1}{2}(p'+s'))$	2.25	16.37	8.76	37.30
$\frac{1}{2}(p'+c)$	4.35	20.08	43.01	112.28
$\frac{1}{2}(p'+\frac{1}{2}(p+s))$	2.89	20.65	8.80	39.87
$\frac{1}{2}(p'+\frac{1}{2}(p'+s'))$	4.63	19.77	44.14	124.82
$\frac{1}{2}(\frac{1}{2}(p+p')+c)$	2.23	16.94	15.23	46.01
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p+s))$	2.97	18.44	7.40	34.79
$\frac{1}{2}(\frac{1}{2}(p+p')+\frac{1}{2}(p'+s'))$	2.10	15.41	15.42	48.67

Table 3.1c: *Curvature comparisons for a polygon from Fig. 3.10.*

in Fig. 3.4. For the top of the diamond, i.e. when the global deviation parameter $D = 1$ (Section 3.3.2.3), v_{CL} again produced best results in terms of curvature minimization. For the bottom of the diamond ($D = -1$), however, the best results were produced by the Edge Length method (v_L).

This result is not surprising. Indeed, when normals at interpolation points tilt toward shorter sides, more bulgy curves would tend to minimize curvature variation. For example, for a rectangle with a large aspect ratio, a near-circular curve would be 'ideal' in terms of minimizing

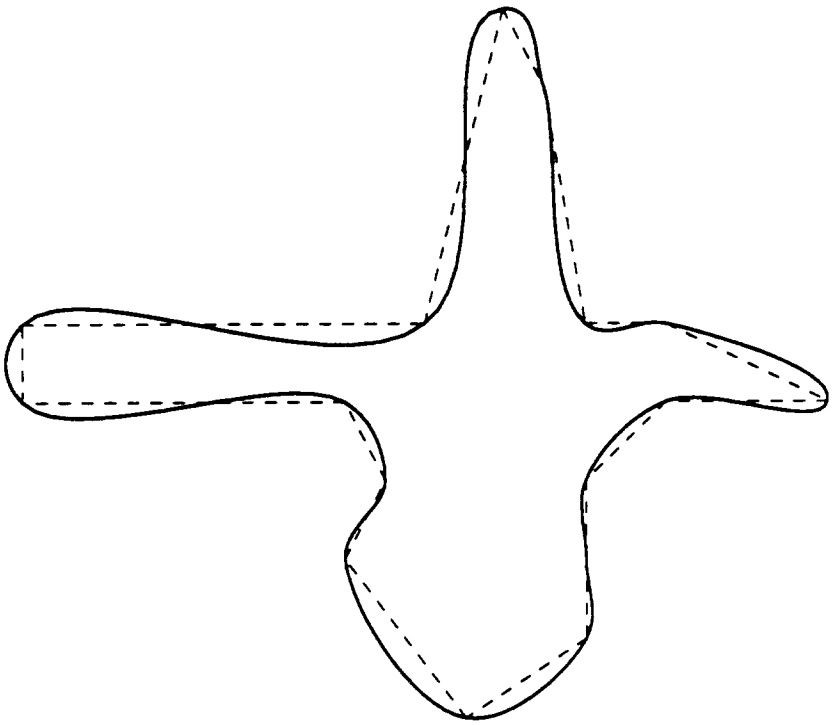


Figure 3.10: *Interpolation of a complex polygon. Default normals and velocities are used.*

curvature variation. The application of the default velocity v_{CL} in this situation results in a tighter curve (Fig. 3.5b, bottom).

In a situation when the tangent directions are prescribed beforehand, it becomes increasingly difficult to define what the 'best' curve should be. A near-circular curve in the above example is probably not what most designers would have in mind. Therefore, we feel that the default velocity v_{CL} should be independent of the normal selection. In practice, the choice of a particular velocity method would clearly be application-dependent.

3.3.3.3. Control of Velocities with Shape Parameters

After default velocities at all the vertices have been determined, it will be possible for the user to modify them locally at individual points. However, the whole process of velocity determination can be affected in a global manner. We associate two global parameters with velocity computation. The first parameter, *bulge*, is a multiplying factor on the calculated velocities. A bigger factor will produce more bulgy, round shapes, while a small value will lead to shorter segments that connect the adjacent vertices more directly. Since the distance from the projection of the interior Bézier point onto the polygon edge to the nearest interpolating point never exceeds one third of the edge length (see previous Section), it is 'safe' to use bulge value up to 1.5. A larger bulge may produce loops in the curve. The default value for bulge is 1.

Another global shape parameter determines the degree of *continuity* of the curve. Changing this parameter from 0 to 1 would generally decrease the ratio of the two velocities at any interpolation point, thus making the curve closer to being C^1. A default value for the (parametric) continuity measure is 1/2, which means that the arithmetic average of Catmull–Rom and Edge Length velocities is used.

To summarize, the velocity on either side of any interpolation point is computed according to the following formula:

$$v = B\ (C\ v_C + (1 - C)\ v_L), \tag{3.22}$$

where B is the bulge parameter, and C is the measure of continuity.

3.4. Pleasing Splines

3.4.1. Procedural Approach

The final default formulations for normals and velocities as described in previous sections produce very good results for most cases. However, the normal and the two velocities at any interpolation point depend only on the 2 neighbor vertices. In some situations, this information is not sufficient to construct a 'pleasing looking' spline or to best implement the 'intent' of the designer.

Traditionally, one would solve this problem by developing a more complex mathematical formulation involving 4 or more nearest neighbors. However, this solution is not efficient because looking further than a nearest neighbor is usually necessary only in special situations that occur relatively rarely. Therefore, we introduced a set of *patterns* of the defining polygon that triggers the application of certain *rules*. If a constellation of interpolation points does not match any of the patterns, the simple 2-neighbor scheme described above is applied. This approach is somewhat similar to a computer program with if-then-else statements where different procedures (rules) are called if certain condition (patterns) are satisfied.

Since the property of stability as described in Section 3.2.3 needs to be maintained, the rules cannot be just 'special cases'. A *threshold* is usually associated with each pattern which determines 'how close' the actual defining polygon is to the pattern. The actual normals and the velocities then continuously vary from the simple 2-neighbor scheme used up to the threshold value to the special rule in case of an exact match.

In the next sections, we describe such special rules for the collinear points and for disproportionate segments in the defining polygon as well as for the end vertices of an open curve.

3.4.2. Collinear Segments

The ability to incorporate linear segments is very important in order to capture the intent of the designer and often also to enhance the visual quality of the spline. Consider Fig. 3.11, where 2 adjacent segments of the topology polygon are collinear. It seems reasonable to infer that the designer wanted a linear curve between P_{i-2} and P_i. Using a 2-neighbor scheme is inadequate in this case, since it produces a curve with an extra inflection point (shown dotted).

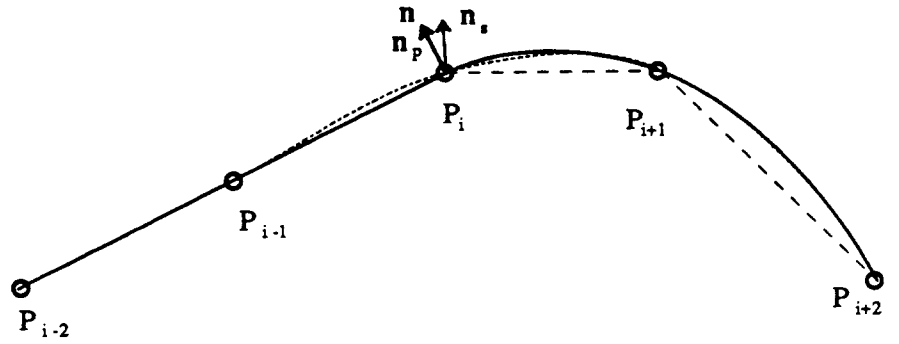


Figure 3.11: *The rule of collinear segments.*
The default curve is shown dotted, and the procedural solution solid.

A non-polynomial spline based on arclength parametrization that can incorporate linear segments has been proposed in [96]. Usually, however, the fact that the tangent at P_i should be collinear with the side $P_{i-1}P_i$ is associated with *convexity preservation*. An algorithm proposed by McLaughlin [89] generates convexity preserving curves comprising parabolic and linear segments. An approach by Goodman and Unsworth [56] computes the tangent direction at P_i as a weighted average of directions $P_{i-1}P_i$ and P_iP_{i+1} in which the weights are some functions of cross products of the defining polygon sides through the points $P_{i-2},\ldots,P_{i+2}$. A similar approach by Renner and Pochop [110] starts with the Catmull–Rom tangent and 'generalizes' it so that convexity is preserved.

However, in all cases these solutions require 4 nearest neighbors to generate a normal at a vertex. Our procedural method makes a simple test that determines if the 2 corresponding sides of the defining polygon are nearly collinear. If this is not the case, the simple 2-neighbor scheme is applied. In this way, if two segments are collinear, the desired behavior is guaranteed by the application of the special rule; otherwise it is produced by the default method.

We assume that the normal at the vertex P_i can always be expressed as a linear combination of the edge normals $\mathbf{n}_p$ and $\mathbf{n}_s$:

$$\mathbf{n}_D = c_p\ \hat{\mathbf{n}}_p + c_s\ \hat{\mathbf{n}}_s . \tag{3.23}$$

The subscript D stands for the default scheme from Section 3.3.2. The coefficients c_p and c_s are easily computed as the coordinates of $\mathbf{n}_D$ in $\hat{\mathbf{n}}_p$, $\hat{\mathbf{n}}_s$ basis. Note that this implies that the normal

between 2 collinear segments is a priori defined.

We express the corrected normal **n** as follows:

$$\mathbf{n} = k'_p \, c_p \, \hat{\mathbf{n}}_p + k'_s \, c_s \, \hat{\mathbf{n}}_s, \tag{3.24}$$

where $k'_p = 0$ if the angle $P_i P_{i+1} P_{i+2} = 180°$ and $k'_s = 0$ if the angle $P_{i-2} P_{i-1} P_i = 180°$.

A weighting function that measures an angle is required. It is natural to define it in terms of the cosine of the angle between adjoining sides (which is equivalent to the scalar product of the unit vectors in the directions of the corresponding polygon edges):

$$k_p' = \min(20\,(1 + \cos(P_i P_{i+1} P_{i+2})), 1), \quad k_s' = \min(20\,(1 + \cos(P_{i-2} P_{i-1} P_i)), 1). \tag{3.25}$$

Essentially, only if the cosine of the corresponding angle lies between -1 and -0.95, or within the threshold of 2.5% of its range, the special rule is applied. The closer the angle is to 180°, the smaller the corresponding coefficient, and the closer the normal is to the normal of the other side. In order to maintain the stability property, the normal has to depend continuously on the angles and P_{i-1} and P_{i+1}. This can be easily achieved by making coefficients k_p and k_s sum up to 1:

$$k_p = \frac{1}{2}\,(1 + k_p' - k_s'), \quad k_s = \frac{1}{2}\,(1 + k_s' - k_p'). \tag{3.26}$$

Thus, the normal is well-defined even if both angles $P_{i-2} P_{i-1} P_i$ and $P_i P_{i+1} P_{i+2}$ are equal to 180°. Fig. 3.12 illustrates the application of the Collinear Segments rule to a polygonal frame with 4 pairs of collinear segments.

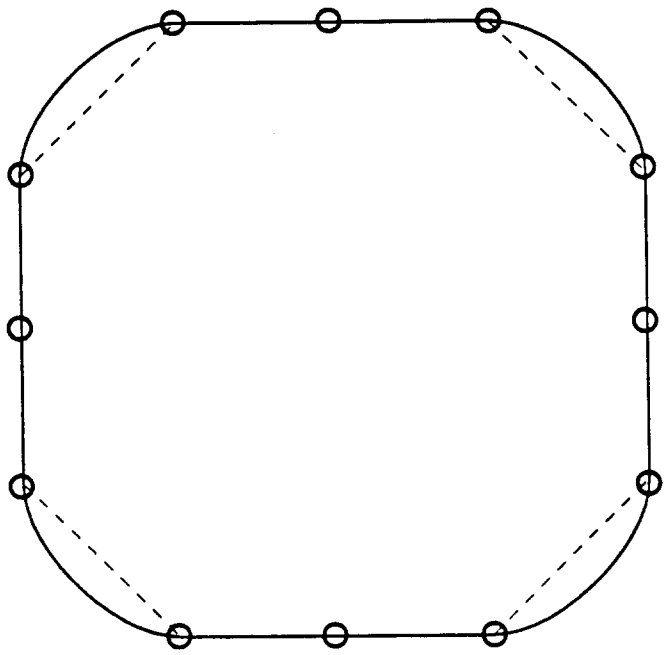


Figure 3.12: *An application of the Collinear Segments rule.*

The formulation (3.23) implies that the normal at the vertex between the 2 collinear sides *must* be orthogonal to these sides. This is due to an implicit assumption that the curve does not cross the defining polygon at interpolation points, but rather between the vertices (see, for example, Fig. 3.10). A resulting limitation of the method is illustrated by Fig. 3.13. For a 'wave' pattern of interpolation points, extra inflections in the spline are introduced because the normals at middle vertices have to be perpendicular to the polygon sides. The inflections can be eliminated by tilting these normals away from their fixed positions. However, at present, we have not yet found an easy way to detect a similar situation and adjust the normals automatically.

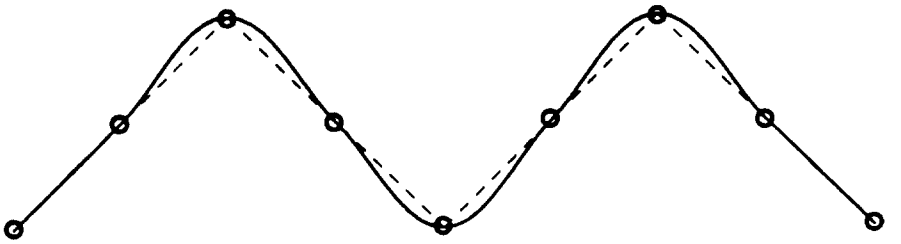


Figure 3.13: *A spline with extra inflection points for a 'wave' pattern of interpolation points.*

3.4.3. Convexity Preservation

The Collinear Segments rule is closely related to the more general problem of convexity preservation [66, 90]. Basically, a curve is called convexity preserving, if the following holds. If a sequence of vertices $P_{i-1}, P_i, P_{i+1}, P_{i+2}$ is positively (negatively) convex, then the curve must be positively (negatively) convex between P_i and P_{i+1}. If, on the other hand, P_{i-1}, P_i, P_{i+1} is positively (negatively) convex, and P_i, P_{i+1}, P_{i+2} is negatively (positively) convex, then the curve must have a single inflection point between P_i and P_{i+1}. Essentially, the curve should have only the inflections imposed by the data and no extra 'kinks'. The formal definition of convexity preservation can be found in [55].

This definition becomes very restrictive if a pair of defining polygon segments are collinear, so that a particular sequence of points can be regarded as convex or as having an inflection point at the same time. The curve is thereby forced to be linear between a corresponding pair of vertices. This situation has already been addressed in the previous section. In fact, we regard convexity preservation as a natural extension of this rule.

The general formulation (3.24) for the normal direction can still be used:

$$\mathbf{n}_D = k_p\, c_p\, \hat{\mathbf{n}}_p + k_s\, c_s\, \hat{\mathbf{n}}_s .$$

However, for best results, all 4 nearest neighbors must be considered in the computation. This can be achieved by letting coefficients k_p and k_s range from 0 to 1 as the corresponding angles $P_i P_{i+1} P_2$ and $P_{i-2} P_{i-1} P_i$ (see Fig. 3.11) range from 180° to 0.

$$k_p' = \frac{1}{2}(1 + \cos(P_iP_{i+1}P_{i+2})), \quad k_s' = \frac{1}{2}(1 + \cos(P_{i-2}P_{i-1}P_i)). \tag{3.27}$$

The reasonable behavior of the spline will still be assured by the coefficients c_p and c_s that come from the default normal scheme.

However, just modifying the normal direction will not prevent the curve from having extra inflections if the selected velocities are too large. This can typically occur if both angles $P_{i-2}P_{i-1}P_i$ and $P_iP_{i+1}P_{i+2}$ are close to 180° (Fig. 3.14). An extra check on velocities is therefore required.

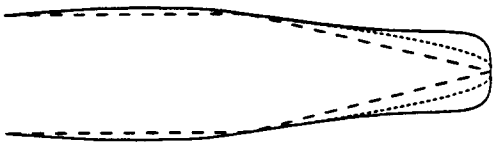


Figure 3.14: *A curve with extra inflections (solid) and convexity preserving (dotted).*

A cubic Bézier segment will be convex between P_i and P_{i+1} if its interior control points lie between the corresponding interpolation point and the intersection I of tangent directions at P_i and P_{i+1} (Fig. 3.15). If the default method would place a Bézier point outside this range, we just move it exactly to I.

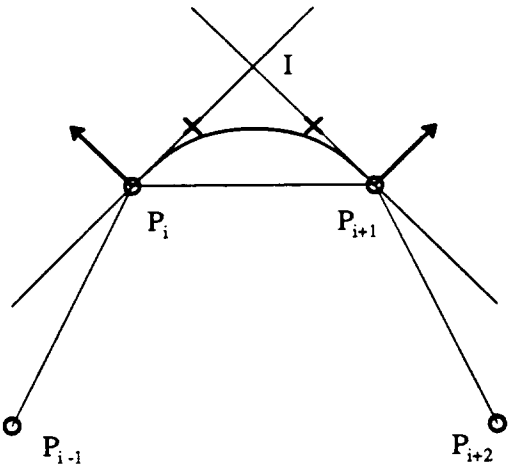


Figure 3.15: *The interior Bézier points must lie on line segments* P_iI and IP_{i+1} *for segment convexity.*

This is similar to the approach taken in [55], where the single control point of a quadratic Bézier point is always placed at I. In fact, one could always determine velocities by placing

corresponding control points somewhere on P_iI. Usually, however, the interior Bézier points as determined by default velocities from Section 3.3.3, will already lie between the vertices and the intersection point of tangent directions, and actual velocity reduction will need to be performed quite rarely. Reducing the default velocities if necessary is more preferable than just placing Bézier points between P_i and I because the latter approach does not take into consideration the constellation of neighbor points. Moreover, this approach cannot be used at all in some situations, e.g. if the defining polygon has an inflection.

No G^1 convexity preserving spline can be constructed if $P_{i-2}P_{i-1}$ is collinear with $P_{i-1}P_i$ and P_iP_{i+1} is collinear with $P_{i+1}P_{i+2}$ because of zero velocities at P_i. An application can deal in two ways with this situation: first, no action can be taken and G^1 discontinuity at P_i be assumed intended. Second, a separate rule for this situation may be introduced that does not reduce velocities further than some fraction of its original length. In this case, G^1 continuity will be preserved at the cost of introducing two extra inflections in the curve.

3.4.4. Disproportionate Sides

If the ratio of lengths of the two adjoining defining polygon sides is very large, a curve may 'negotiate a tight turn'. Suppose that the length of the side P_iP_{i+1} is much smaller then the lengths of $P_{i-1}P_i$ and $P_{i+1}P_{i+2}$ (Fig. 3.16). Then the normals at P_i and P_{i+1} will be significantly different from each other, which can lead to an apparent visual discontinuity in the tangent.

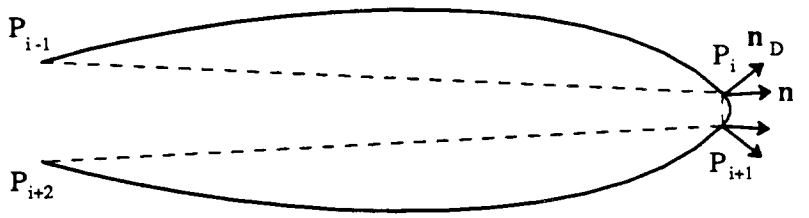


Figure 3.16: *The case of disproportionate polygon sides.*

In this case the normals at P_i and P_{i+1} should 'tilt' to the shorter side of the defining polygon. This can be easily achieved by increasing the appropriate coefficient of the default normal formulation (3.23) in the $\hat{\mathbf{n}}_p$, $\hat{\mathbf{n}}_s$ basis:

$$\mathbf{n} = c_p\ \hat{\mathbf{n}}_p + \frac{r}{R}\ c_s\ \hat{\mathbf{n}}_s . \tag{3.28}$$

Here r is the ratio of larger and smaller sides, and R is some threshold; the rule is applied only in the case of $r > R$. We set $R = 10$. Similar results can be achieved by selecting the default normal method to be closer to the bottom of the 'diamond' in Fig. 3.4, or, equivalently, by choosing the deviation parameter D (Section 3.3.2.3) to be closer to -1.

This rule is a good example of the versatility of the procedural approach and its suitability to various applications. Indeed, some modeling systems may require the ability to have sharp corners, or discontinuities in tangent direction. This can be achieved by 'doubling' the vertices where the discontinuity is desired. In this case, the designer will just disable the application of the Disproportionate Sides rule.

3.4.5. Rules for the Endpoints

As described in Sections 3.3.2 and 3.3.3, normals and velocities at a certain point always depend on the two nearest vertices. If a special rule applies, four neighbor points are needed. A closed curve always has the required number of neighbors on either side; for an open curve, however, special constructions are necessary for the first and the last vertices. In addition, if four neighbors are required for the computation, special constructions will also be needed for P_1 and P_{n-1}.

Catmull–Rom and B-splines trivially avoid this problem by constructing a spline only on those intervals where all the required vertices are present. For example, for a sequence of points $P_0, \ldots, P_n$, a cubic curve would start at P_1 and end at P_{n-1}. This is clearly unacceptable for pleasing splines; one naturally expects a spline to pass through *all* the interpolation points.

In the next sections, we describe possible rules for normals and velocities at P_0 and P_1; the rules for P_n and P_{n-1} are analogous.

3.4.5.1. Normal Construction at P_0

Perpendicular Method. The easiest way to define a normal at P_0 is just to make it orthogonal to the first segment P_0P_1 (Fig. 3.17a):

$$\mathbf{t}_P = \hat{\mathbf{a}}. \tag{3.29}$$

Here again $\hat{\mathbf{a}}$ is a unit vector in the direction of $\mathbf{a}$. In this case, an inflection point will normally appear in the first curve segment.

The following three methods attempt to determine the tangent direction at P_0 by the geometry of the triangle $P_0P_1P_2$. In these approaches, the tangent direction is some reflection of the direction of the chord P_0P_2 through P_0, so that the inflection in the first curve segment is avoided.

Reflection Method. In this method, the tangent direction at P_0 is given by the reflection of the chord P_0P_2 with respect to P_0P_1 (Fig. 3.17b):

$$\mathbf{t}_R = 2\,(\mathbf{c} \cdot \hat{\mathbf{a}})\,\hat{\mathbf{a}} - \mathbf{c}. \tag{3.30}$$

Adjacent Side Vector Difference. A similar result can be achieved at a smaller computational cost, if a difference of $\mathbf{a}$ and $\mathbf{b}$ is used (Fig. 3.17c):

$$\mathbf{t}_D = \mathbf{a} - \mathbf{b}. \tag{3.31}$$

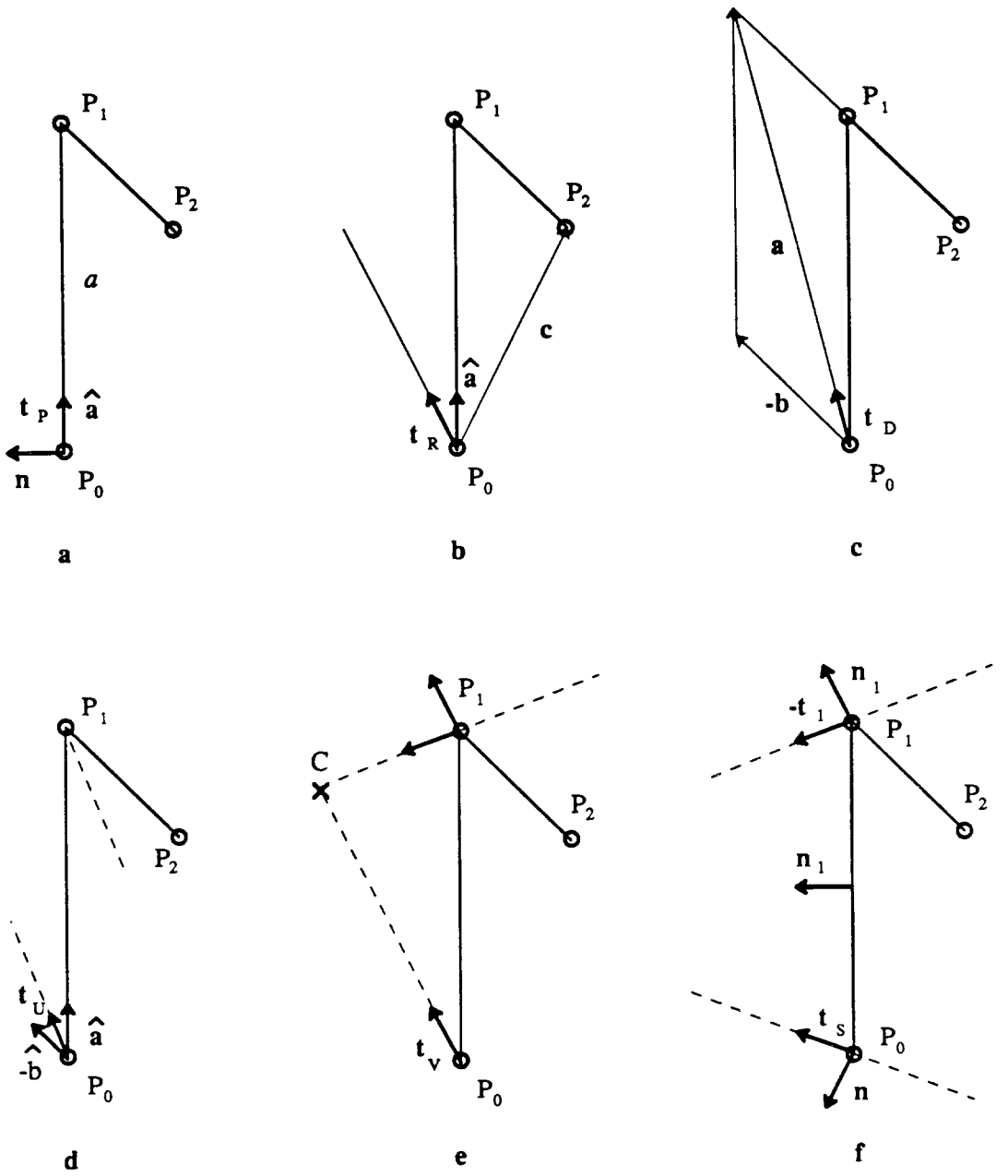


Figure 3.17: *Normal determination at* P_0.
a. *Side Perpendicular.* **b.** *Reflection.* **c.** *Side Vector Difference.*
d. *Unit Side Vector Difference.* **e.** *Adjacent Velocity.* **f.** *Symmetric.*

Unit Side Vector Difference. Alternatively, the tangent can be formulated in terms of unit vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$:

$$\mathbf{t}_U = \hat{\mathbf{a}} - \hat{\mathbf{b}}. \tag{3.32}$$

In this way, the tangent at P_0 depends only on the angle at P_1, i.e. it is parallel to the angle bisector at P_1 (Fig. 3.17d).

The last three methods have disadvantages similar to those in the Catmull–Rom method. That is, large overshoots can result if the geometry of the triangle $P_0P_1P_2$ is strongly asymmetric. This is due to the fact that the above schemes do not take into consideration the normal at P_1, computed with the default method from Section 3.3.2. Using this information can make the shape of the curve near P_0 more pleasing and predictable.

P_1 Velocity Method. In this method, the tangent direction is determined by a certain point C on the tangent line at P_1 (Fig. 3.17e):

$$\mathbf{t}_V = \mathbf{C} - \mathbf{P}_0, \tag{3.33}$$

where

$$\mathbf{C} = \mathbf{P}_1 + k\ \mathbf{v}_1. \tag{3.34}$$

Here $\mathbf{v}_1$ is the velocity at P_1 towards P_0. The coefficient k determines how the curve bulges near P_0. For $k = \frac{1}{3}$, the two interior control points and P_0 will be collinear, and curvature at P_0 will be equal to 0. Good visual results can be attained for the value $k = \frac{1}{2}$.

Symmetric Method. The first spline segment can be made symmetric with respect to the perpendicular axis through the midpoint of P_0 and P_1 if the angles between P_0P_1 and the tangent directions at these points are made equal (Fig. 3.17f):

$$\mathbf{t}_S = 2\,(-\mathbf{t}_1 \cdot \mathbf{n}_1)\,\mathbf{n}_1 + \mathbf{t}_1. \tag{3.35}$$

Here $\mathbf{t}_1$ is the tangent direction at P_1. If this rule is used, a nice symmetric shape will be produced if a defining polygon is a part of a regular n-gon.

The best visual results are produced with the last two methods that look at the normal at P_1. Both of them are implemented in our system; the actual choice of the method is usually application dependent.

3.4.5.2. Velocity Determination at P_0

Since there is no 'adjoining' side at P_0, one could just use the methods that do not depend on it. Edge Length and Projection methods as well as their average from Section 3.3.3 can all be utilized. The Average method produces the best results, but requires somewhat more computation.

If the normal at P_0 is computed with the Symmetric method (3.35) as described above, then the velocity should also be symmetric:

$$v_{0,s} = v_{1,p}. \tag{3.36}$$

In this case no extra computation is required for the velocities at the endpoints.

Our implementation supports Average and Symmetric methods for the velocities at the end vertices; again, the actual choice of the approach depends on the needs of the application.

3.4.5.3. Normals and Velocities at P_1

The normal at P_1 (and P_{n-1}) needs special attention only when a special rule is invoked that utilizes four neighbor points, e.g. Collinear Segments. In this case, we set the coefficient k_s' of (3.25) that depends on the angle at P_0 to be identically 1. Thus, the normal at P_1 will be orthogonal to the second defining polygon segment if it is collinear with the third segment.

The Disproportionate Sides rule, on the other hand, does not require any special treatment for normals and velocities at P_1. The purpose of this rule is to prevent visual discontinuity; however, it cannot occur at the beginning or at the end of the curve. Likewise, the velocities at P_1 can be computed in a standard manner as they never depend on more than two nearest neighbors.

3.5. Global Shape Parameters

Some of the procedures discussed above contain parameters that have been set to some predetermined default values. The deviation of the default normal from the bisector is one example of such a parameter. Its value will affect the general behavior of all tangents at the interpolation points. Bulge, or a multiplication factor for the velocities is another example. The larger this factor is, the more round and bulgy shapes will be produced; smaller values will result in shorter segments that connect the two vertices more directly. Finally, the thresholds for Collinear Segments and Disproportionate Sides rules can also be utilized to control the global shape of the curve in special situations.

The fact that the *normal deviation*, as well as *bulge* and *continuity measure* are very useful for interactive shape control has been previously acknowledged. Many tension-controlled splines have been described in the literature [31, 75, 92]. The continuity measure, which has a similar effect to Barsky's $\beta 1$ parameter, is also mentioned in [79]. Analogously, normal deviation is similar to the *bias* of the spline [4, 47].

The fact that the procedural method can easily accommodate all these useful shape controls illustrates its flexibility. Our approach establishes good default values for global shape parameters. Typically, a whole *range* of possible values for a particular parameter will result in a 'reasonable' behavior of the curve. It is possible (but usually does not make sense) to go beyond some of these ranges and, for example, to make the ratio of the two velocities at a certain vertex larger than the ratio of the underlying sides. However, forcing this condition would take the continuity measure parameter out of its range. Analogously, it is not desirable to tilt the normal too much from its 'pleasing region' in Fig. 3.4, or to make the bulge parameter too large (> 1.5) or too small (< 0.5).

Clearly, global shape parameters can be utilized to affect the overall shape of the curve in a uniform manner. These values are therefore a powerful addition to the local parameters (normals, velocities) and can be viewed as shape parameters for traditional splines that have been carried over to the procedural method.

3.6. General Curve Properties and Pleasing Splines

The procedural method of constructing interpolating curves, described above, does not necessarily guarantee that the various spline properties discussed in Section 3.1 will be satisfied. In this section, we look in more detail at some of these properties, evaluate to what degree our procedural splines possess these properties, and argue tha some of them may be inconsistent with the requirements of pleasing splines.

3.6.1. Monotonicity Preservation

Monotonicity preservation is often considered together with convexity preservation as prerequisites for constructing shape-preserving splines. However, monotonicity preservation has a more limited application as it concerns only the curves that interpolate functional data points $P_i(x_i, y_i)$, where the sequence x_i is strictly increasing. If the sequence $y_{i-1}, y_i, y_{i+1}, y_{i+2}$ is increasing (decreasing), then the curve is also increasing (decreasing) between P_i and P_{i+1}. Furthermore, if $y_{i-1} \le y_i \ge y_{i+1}$ $(y_{i-1} \ge y_i \le y_{i+1})$ then the curve has a single local maximum (minimum) between P_{i-1} and P_{i+1}.

The above definition is asymmetric in the sense that the x and y coordinates are treated differently. This results in the fact that monotonicity preserving splines are not rotation invariant as the following example illustrates.

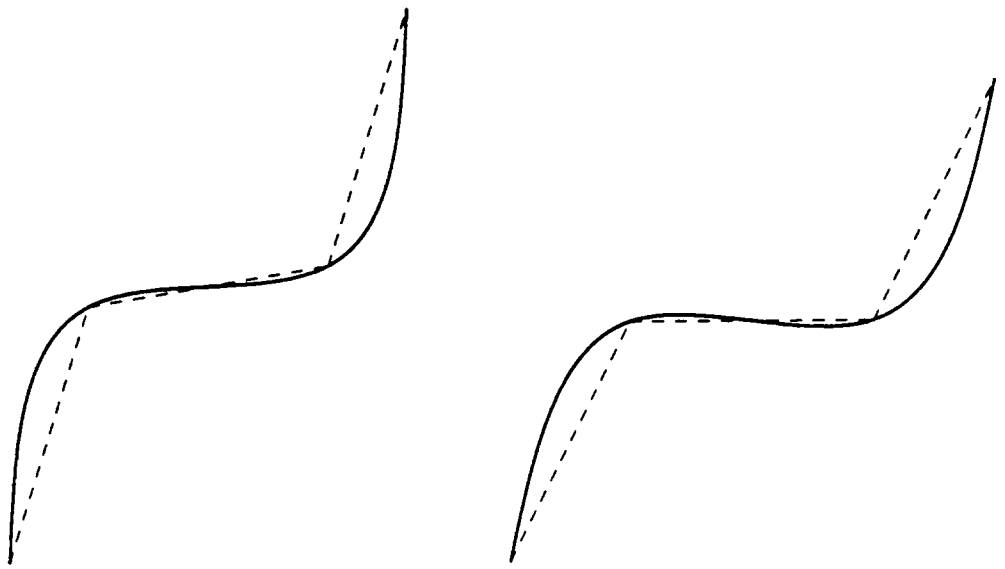
Figure 3.18 shows that the property of monotonicity preservation can be easily lost after a rotation of the curve. Therefore, the property of monotonicity preservation is incompatible with pleasing splines since the latter should be rotation invariant.

3.6.2. Differential Scaling Invariance

Invariance under differential scaling, i.e. different scaling ratios in x and y, is another common property of traditional splines. It states that the scaled image of a curve must be identical to the curve constructed from the scaled defining polygon. Pleasing splines, however, should *not* be invariant under differential scaling. In Fig. 3.19, the solid curve **b**, which was obtained by scaling the curve **a** seems to be too oblong for most people's taste.

3.6.3. Consistency

The consistency property [94] (Section 3.2.4), on the other hand, would be very useful and natural. However, it is very hard to attain as the following argument shows.



a b

Figure 3.18: a. *A monotonicity preserving spline.*
b. *The same rotated spline is no longer monotonicity preserving.*

We will assume that the spline is at least G^1, so that it has a continuous normal at each point. Since the spline is local, we will also assume that the normal $\mathbf{n}_i$ at point P_i depends only on P_{i-1}, P_i, and P_{i+1} (the case when $\mathbf{n}_i$ depends on more vertices is handled analogously). Thus, a perturbation of the vertices P_{i-1}, P_i, P_{i+1} may result in a perturbation in the normal at P_i.

Consider Fig. 3.20. The normal at P_2 of the interpolating spline through $P_0, \ldots, P_4$ depends on P_1, P_2, and P_3:

$$\mathbf{n}(P_2) = \mathbf{n}(P_1, P_2, P_3). \tag{3.37}$$

Now suppose that a new point D has been added on a curve segment between P_1 and P_2. Clearly, the position of D depends on the normals at P_1 and P_2. These normals, in turn, depend on the points P_0, P_1, P_2, and P_1, P_2, P_3, respectively. Therefore, D is a function of four points:

$$\mathbf{D} = \mathbf{d}(P_0, P_1, P_2, P_3). \tag{3.38}$$

Now, the expression for the normal at P_2, computed with the expanded vertex set, $P_0, P_1, D, P_2, P_3, P_4$, is

$$\mathbf{n}(P_2) = \mathbf{n}(D, P_2, P_3). \tag{3.39}$$

However, since D also depends on P_0,

$$\mathbf{n}_{new}(P_2) = \mathbf{n}_{new}(P_0, P_1, P_2, P_3). \tag{3.40}$$

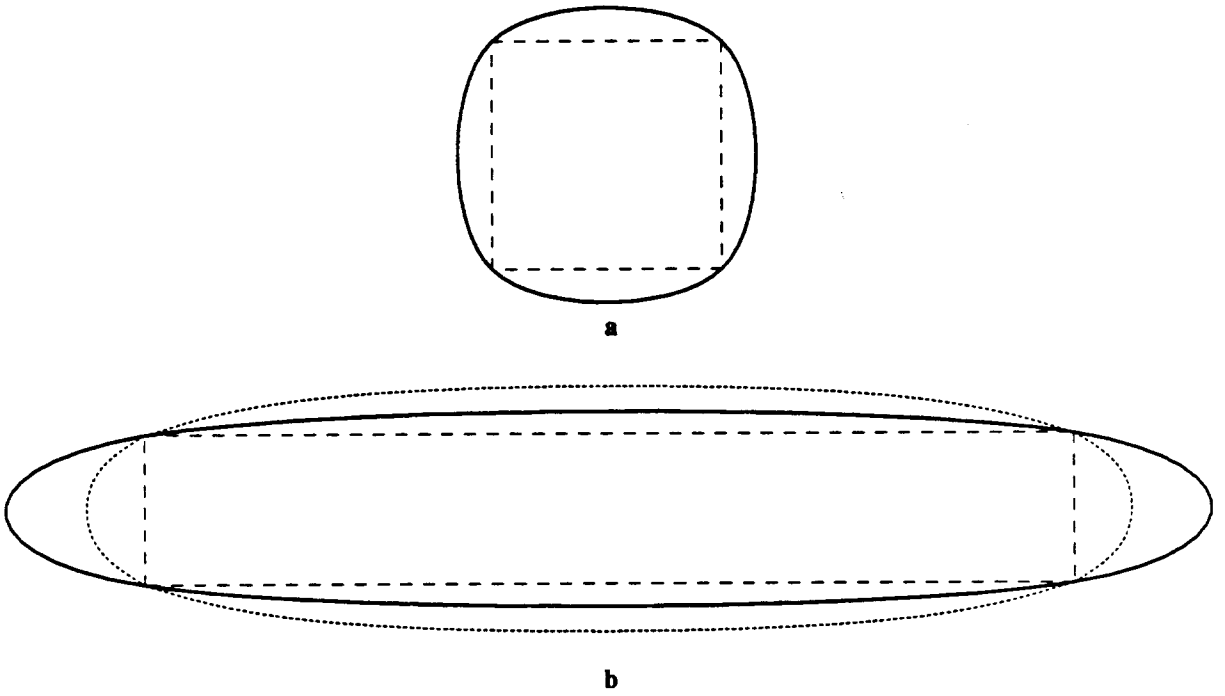


Figure 3.19: *A scaled version (***b***, solid) of the original (***a***) spline. A pleasing spline is shown dotted.*

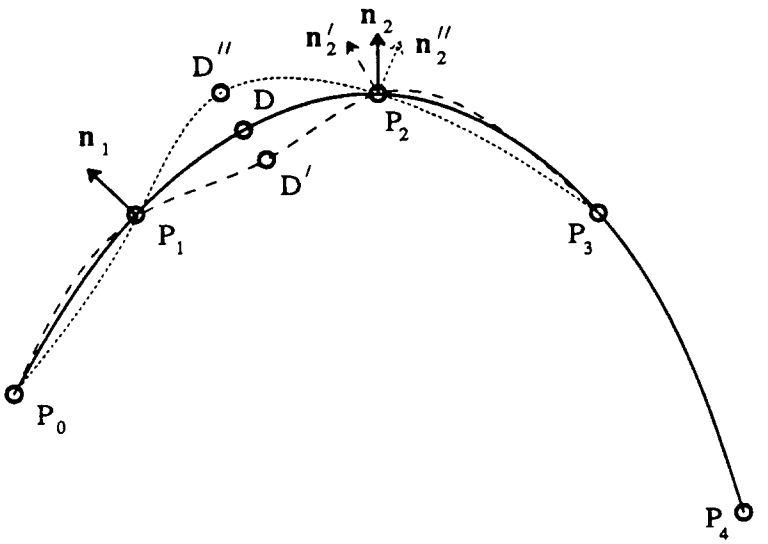


Figure 3.20: *Local spline cannot be consistent.*

The consistency property requires

$$\mathbf{n}(P_2) = \mathbf{n}_{new}(P_2). \tag{3.41}$$

However, this is false because the original $\mathbf{n}(P_2)$ does not depend on P_0, and we arrive at the contradiction. This also becomes evident by looking at Fig. 3.20: All positions shown (D, D', D'') of the predecessor of P_2 would have to produce the same normal at P_2.

Thus, one is forced to choose between consistency and being able to modify the curve in an interactive, local manner. Moreover, consistent splines have a rather complicated mathematical formulation [94]. Therefore, the consistency property, although desirable, is not practical in an engineering system.

3.6.4. Relationship of Various Spline Properties to Pleasing Splines

Table 3.2 summarizes various spline properties and indicates whether they are supported in our implementation of G^1 pleasing splines.

Property	Supported	Comments
Rotation, Translation Invariance	Yes	
Uniform Scaling Invariance	Yes	
Differential Scaling Invariance	No	Not a pleasing property
Local Control	Yes	
Local Shape Parameters	Yes	Normal, two velocities at each vertex
Global Shape Parameters	Yes	Weighting factors for normals, velocities; thresholds
Simplicity	Yes	Cubic Bézier
Stability	Yes	
Consistency	No	Too hard to attain; not compatible with locality and simplicity
Convexity Preservation	On demand	Special rule
Monotonicity Preservation	No	Not compatible with rotation invariance

Table 3.2: *Various spline properties.*

3.7. Summary

We have presented a local G^1 interpolation method. A curve is represented as a union of cubic Bézier segments, constructed between each pair of adjacent vertices.

The Bézier formulation makes it possible to naturally define the curve in terms of some shape parameters: the *normal* and the two *velocities* at each interpolation point. Unlike traditional splines, our procedure goes through a sequence of procedural steps, or rules, to determine the best possible values for these shape parameters. The resulting spline is guaranteed to be of good visual quality even for highly asymmetric defining polygons.

The procedural approach also makes our splines very versatile and easily adjustable to the application's particular needs. Besides modifying local shape parameters, the user can affect the curve in a global manner either by selecting appropriate global shape parameters, or by choosing a set of rules that apply to certain 'extreme' situations.

Furthermore, the procedural method can be easily generalized to construct curvature continuous, or G^2, curves (Chapter 4). Moreover, our approach can be used in $\mathbf{R}^3$ as well as in $\mathbf{R}^2$: for space curves, the normals and velocities are determined in the plane of the three neighbor vertices. Finally, an extension to interpolating surfaces is certainly possible. In Chapter 5, we will use the rules for normals and velocities as an inspiration to define similar geometric parameters for Bézier and Gregory patches.

4

Curvature Continuous Cubic Splines

In this chapter, we describe a natural extension of G^1 pleasing splines from the previous Chapter to curvature continuous curves in $\mathbf{R}^2$. We present a G^2 scheme with a free choice of a tangent direction, two derivatives, and a curvature at each interpolation point. In this scheme, two cubic segments are constructed for each adjacent pair of vertices. Pleasing behavior is assured by using the G^1 procedure to choose directions and derivatives and by a judicious choice of curvatures. We further discuss the possibility of extending the method to $\mathbf{R}^3$ and conclude with a comparison of our scheme with various other local schemes for curve construction.

4.1. Introduction

In the previous chapter, we have described a method to construct pleasing G^1 continuous interpolating splines with local and global shape parameters. However, it is often very important to have not only tangent direction continuity, but continuity of curvature (G^2) as well. We would like to build a natural G^2 extension of our method and at the same time maintain pleasing behavior, simple Bézier representation, and freedom of selecting various shape parameters.

Many techniques of constructing curvature continuous splines have been proposed. Minimization of some physical property usually results in a global non-polynomial solution, sometimes even with some shape parameters [80, 118]. After Nielson [92] has proposed a global piecewise polynomial solution to the interpolation problem, *local* polynomial and rational schemes were introduced. Earlier solutions [23, 75] did not have any shape parameters. In a recent development, DeRose and Barsky [34] have combined beta parameters, usually used for approximating curves [4, 5], with interpolating Catmull–Rom splines. The result is a local interpolating quintic spline with beta shape parameters. A similar method with quintic Bézier segments is described in

[69].

Indeed, quintic Bézier curves are well-suited for G^2 local interpolation because the first and the last inner control points can be used to define derivatives, while the inner pair of control points defines (indirectly) the curvatures at the vertices. For cubic Bézier curves, however, the curvature at the end of the segment depends on both interior control points. Therefore, it is impossible to achieve G^2 continuity without sacrificing the freedom of choosing the Bézier points. An interesting compromise, proposed by de Boor et al [19], allows free choice of the tangent directions and of the curvatures at the interpolation points. The velocities at each curve segment are then determined from a system of two quadratic equations resulting from satisfying certain curvature constraints at the vertices. However, the solution to the above system does not always exist and the method fails to produce G^2 curves for some configurations of interpolation points. An introduction of a *breakpoint* in some segments overcomes this difficulty [56] at the price of representing the curve with *two* cubics instead of one between *some* pairs of adjacent vertices.

It turns out that by introducing breakpoints between *every* pair of adjacent vertices, one can achieve complete freedom of choosing tangent direction, derivatives, and curvature at every vertex. The default values for the tangent and the derivatives can be taken from our G^1 method, which guarantees that the shape of G^2 curve will be similar to the corresponding G^1 curve.

In the context of this Chapter, it is more convenient to deal with the directions, or the tangents of the curve at interpolation points, rather than with the normals. Also, here we assume that the 'velocity' magnitude is the distance from a vertex to the corresponding Bézier point, rather than the derivative value, which is equal to 3 times this distance for cubics.

4.2. Preliminaries

Consider a cubic Bézier segment $\mathbf{c}(u)$. Let $\mathbf{t}_0$ and $\mathbf{t}_1$ be the unit tangent vectors at endpoints, and v_0 and v_1 be the distances from the endpoints to the nearest Bézier point. The curvature at the point corresponding to parametric value u is given by

$$\kappa(u) = \frac{\mathbf{c}'(u) \times \mathbf{c}''(u)}{|\mathbf{c}'(u)|^3} \tag{4.1}$$

where

$$\mathbf{a} \times \mathbf{b} = a_x b_y - a_y b_x \tag{4.2}$$

is a cross product in 2D. Since for the internal control points

$$\mathbf{C}_1 = \mathbf{C}_0 + v_0 \mathbf{t}_0,$$

$$\mathbf{C}_2 = \mathbf{C}_3 - v_1 \mathbf{t}_1,$$

it is easily verified that the curvatures κ_0 and κ_1 at the corresponding endpoints C_0 and C_3 can be expressed as

$$\kappa_0 = \frac{2\,t_0 \times (C_2 - C_1)}{3\,v_0^2}, \qquad \kappa_1 = \frac{2\,t_1 \times (C_1 - C_2)}{3\,v_1^2}. \tag{4.3}$$

Assuming t_0 and t_1 are given, and the curvatures κ_0 and κ_1 are chosen, then equations (4.3) can be regarded as a system for the 2 unknowns v_0 and v_1. Thus, for the given tangent directions, a G^2 continuous curve can be constructed by assigning a curvature value at each joint and solving a corresponding system of equations for each Bézier segment [19]. Note that this implies that we have given up control over the velocities v_0 and v_1.

The system (4.3) is quadratic in terms of v_0 and v_1. For certain configurations of vertices and directions, no positive solution is possible. Suppose that a linear segment joins with a non-linear segment (Fig. 4.1). Then the curvature at P_1 has to be 0, because an adjoining line segment P_0P_1 has zero curvature everywhere. This means that both Bézier points of the segment P_1P_2 have to lie on the line between P_1 and P_2, which is inconsistent with the tangent direction at P_2.

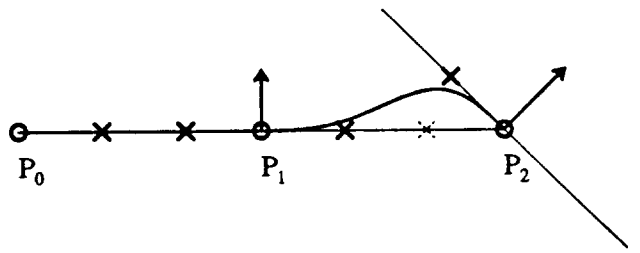


Figure 4.1: *A linear segment joining non-linear segment.*

However, by inserting a new vertex, or a *breakpoint*, into the defining polygon between P_1 and P_2, it is possible to preserve curvature continuity. Goodman et al [56] use this technique to deal with curves having linear segments. Their solution uses a pair of curve segments between those adjacent vertices for which a breakpoint insertion is necessary to maintain G^2 continuity (usually this happens when a linear segment joins a non-linear curve segment). We extend this approach by using a pair of cubic segments between *every* pair of interpolation points. As we will see in this Chapter, the following advantages are then realized:

- The curve is guaranteed to be G^2;
- The representation is still in terms of cubic Bézier segments;
- Since linear segments don't require special treatment, the stability property can be easily achieved;

- As in the G^1 method from Chapter 3, the normals and the velocities can still be chosen freely;
- In addition, one can choose curvatures at every interpolation point.

Finally, this approach is a natural extension of the G^1 procedural splines. The rules and default values for the normals and for the velocities of the G^1 method can be applied without any modification.

4.3. Breakpoint Insertion

Consider the cubic Bézier segment in Fig. 4.2. Assume that the curvatures κ_0 and κ_1 at the endpoints C_0 and C_3 have somehow been chosen (see Section 4.6). We would like to substitute the original cubic with control points at C_i, $i = 0,1,2,3$ by the two cubics, with control points S_i and T_i, $i = 0,1,2,3$ in such a way that the curvatures at $S_0 = C_0$ and $T_3 = C_3$ are as prescribed, and the derivatives at these points match the derivatives of the original cubic. Moreover, we need to guarantee curvature continuity at $S_3 = T_0$.

It is convenient to assume that the parameter in the new cubics varies twice as fast as in the original cubic. This assumption is similar to the one in the process of midpoint subdivision of Bézier segments [10]. Then the points S_1 and T_2 are readily defined:

$$\mathbf{S}_1 = \frac{1}{2}(\mathbf{C}_0 + \mathbf{C}_1), \tag{4.4}$$

$$\mathbf{T}_2 = \frac{1}{2}(\mathbf{C}_2 + \mathbf{C}_3). \tag{4.5}$$

Now we need to choose S_2 and T_1 in such a way that the curvatures at the endpoints match κ_0 and κ_1 respectively. This choice, however, is not unique. Indeed, it follows immediately from the definition of curvatures (4.3) that the locus place of the points S_2 yielding a fixed curvature κ_0 at C_0 is a line parallel to the direction $\mathbf{t}_0$ at C_0 at a distance of $\frac{3}{2}\kappa_0 v_0^2$. Analogously, the geometric place of the points T_1 with the fixed curvature κ_1 at C_3 is a line parallel to $\mathbf{t}_1$ with the offset of $\frac{3}{2}\kappa_1 v_1^2$ (Fig. 4.2). Note that that the actual position of S_2 and T_1 on *equal curvature lines* can be defined in terms of Barsky's $\beta 2$ parameter [5].

Once the positions of S_2 and T_1 have been selected (see the next Section), $S_3 = T_0$ needs to be determined. Obviously, for G^1 continuity between the two subsegments, this point has to be on the line between S_2 and T_1. As indicated in [40] for more general situations, the G^2 continuity condition at $S_3 = T_0$ can be derived from matching the curvatures at the two sides

$$\frac{2\,\mathbf{b} \times (\mathbf{S}_1 - \mathbf{S}_2)}{3\,v^2} = \frac{2\,\mathbf{b} \times (\mathbf{T}_2 - \mathbf{T}_1)}{3\,(l - v)^2}. \tag{4.6}$$

Here $\mathbf{b}$ is the direction of S_2T_1, v is the unknown distance from S_2 to S_3, and l is the length of

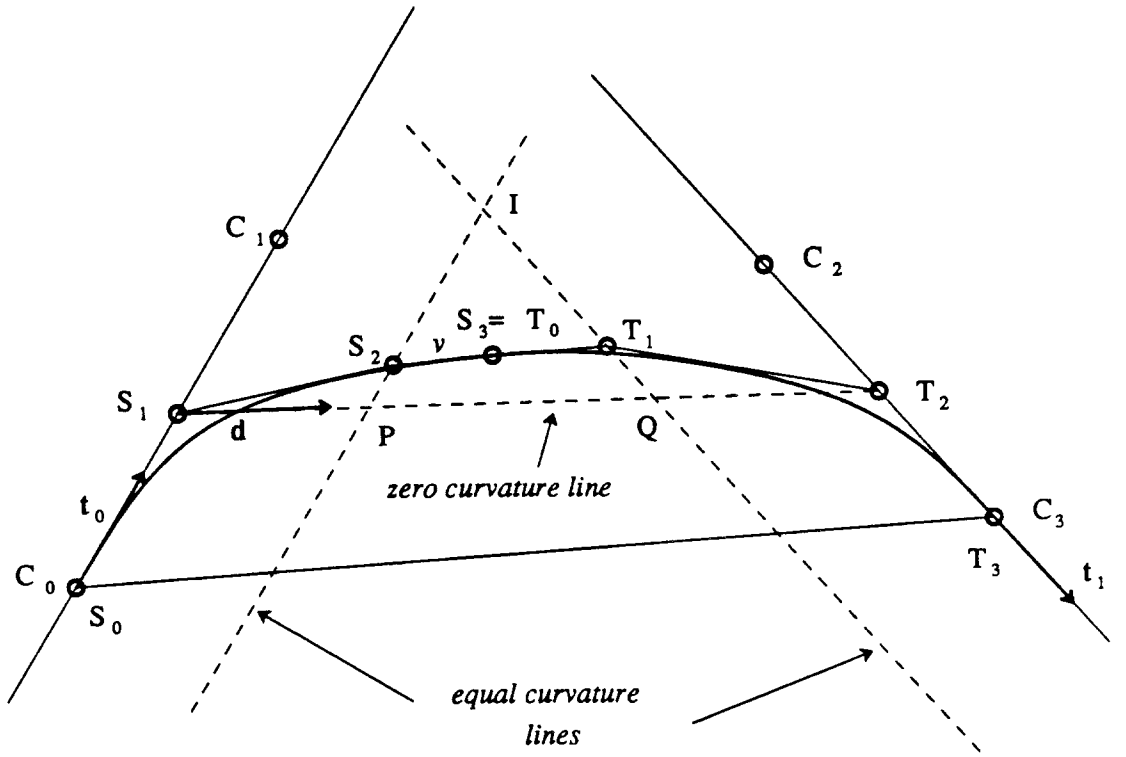


Figure 4.2: *Breakpoint insertion in a cubic Bézier segment.*

S_2T_1. Letting

$$c_1 = \frac{2}{3}\,\mathbf{b}\times(\mathbf{S}_1-\mathbf{S}_2),\quad c_2 = \frac{2}{3}\,\mathbf{b}\times(\mathbf{T}_2-\mathbf{T}_1) \tag{4.7}$$

we obtain a quadratic equation for v:

$$(c_2-c_1)v^2 + 2lc_1v - c_1l^2 = 0. \tag{4.8}$$

The solution to this equation is:

$$v = \frac{\pm\sqrt{c_1c_2}-c_1}{c_2-c_1}\,l. \tag{4.9}$$

If $c_1 = c_2$, then

$$v = \frac{1}{2}\,l. \tag{4.10}$$

A positive solution in the range $[0,l]$ always exists if the sequence $S_1S_2T_1T_2$ is convex. A necessary (but not sufficient) condition for that is that c_1 and c_2 have the same sign, or, equivalently, that both S_2 and T_1 lie on the same side of the S_1T_2 line.

A very simple sufficient condition for solvability of (4.8) can be obtained by forcing S_2T_1 to be parallel to S_1T_2. This choice simplifies the solution even further: in this case, the cross products in the numerators of the equation (4.6) will be just the offsets of the points S_2 and T_1 from the line S_1T_2, and so $c_1 = c_2$. Then,

$$\mathbf{S}_3 = \mathbf{T}_0 = \frac{1}{2}(\mathbf{S}_2 + \mathbf{T}_1). \tag{4.11}$$

Note that this position of the breakpoint makes the two subsegments also *parametrically* continuous (C^1).

The two cubic G^2 subsegments will then be completely defined by the choice of the points S_2 and T_1. We now look at various methods of choosing these points.

4.4. Choice of the Points $\mathbf{S}_2$ and $\mathbf{T}_1$

4.4.1. Algebraic Methods

C^2 continuity. As shown above, if the line S_2T_1 is chosen parallel to S_1T_2, then the resulting cubic segments will be C^1 continuous at the breakpoint. It seems natural to extend this formulation and determine the distance between the two parallel lines so that the curve is made C^2 at S_3.

Expressing the second derivatives on the two sides of S_3 in terms of the corresponding Bézier points and making them equal results in the condition

$$2(\mathbf{T}_1 - \mathbf{S}_2) = \mathbf{T}_2 - \mathbf{S}_1. \tag{4.12}$$

However, this solution is not acceptable as extra inflections or even loops may be forced if the intersection point of the equal curvature lines lies near the S_1T_2 line.

Curvature Fairing. In recent publications, several researchers argued that the C^2 curves can be 'faired' by smoothing the derivative of the curvature [78, 86]. Farin et al [44, 45] formulate this condition in terms of C^3 continuity. In the context of our construction, however, we will have to define fairness in terms of the derivative of curvature magnitude (C^3 cannot be forced because, in general, the curve will not even be C^2).

Looking again at Fig. 4.2, we can see that placing all interior control points of the two segments on S_1T_2 line will force zero curvature at the breakpoint. For this reason, we call S_1T_2 the *zero curvature line*. The graph of the curvature magnitude in this case is shown dotted in Fig. 4.3.

In the other extreme case, when S_2, S_3, and T_1 are all placed at the intersection point I of the equal curvature lines, the curve will have 'infinite' curvature there. The corresponding plot is shown dashed.

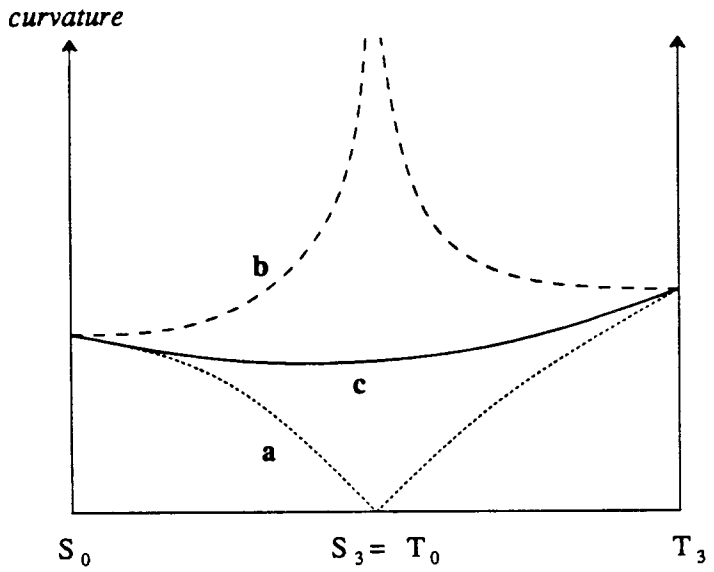


Figure 4.3: *Curvature plots for various offsets of the breakpoint from the zero curvature line.*
a. *Zero offset.* **b.** *Equal curvature lines intersection offset*
c. *An offset, corresponding to curvature derivative continuity.*

It is clear that there exists a distance from the breakpoint to the equal curvature line that corresponds to the continuity of curvature derivative (corresponding plot shown solid). However, finding this distance involves a solution of a polynomial equation of a high degree. Also, it is not clear that the fairing of the spline at the breakpoint will necessarily result in a better-looking curve. Finally, the curvature derivatives at the interpolation points will still remain discontinuous anyway, so no improvement in overall smoothness of the curve will be gained.

Assignment of Curvature at the Breakpoint. From the equation (4.6), the curvature value κ at the breakpoint, the offset h of the line S_2T_1 from S_1T_2, and the distance v from S_2 to S_3 are related as follows:

$$\kappa = \frac{2h}{3v^2}. \tag{4.13}$$

Since the relationship between h and v can be deduced from the similarity of the triangles S_2T_1I and PQI (Fig. 4.2), h can be found from the above equation, if the curvature κ at $S_3 = T_0$ is known. From the discussion above it follows that this equation will always have a solution for any value of κ.

However, the task of selecting a proper curvature at the breakpoint is a very difficult one. Simple averaging methods of curvature selection are inadequate. For example, a curve segment with zero curvatures and non-parallel tangents at the endpoints should clearly have non-zero curvature in the middle (Fig. 3.12). Moreover, inappropriate curvature at S_3 can easily result in asymmetric shapes, extra inflections, or even loops in the resulting cubic subsegments.

Minimization of Curvature Variation. Alternatively, one could define the breakpoint in such a way that the variation of the curvature on both subsegments be minimized. In loose terms, this would keep the curvature as constant as possible. However, minimization of the corresponding 'energy integral' $\int \kappa^2 \, ds$ [51] again results in a very complex formulation.

For these reasons, we use a procedural method to find the best offset of S_2T_1 from the zero curvature line. This approach is a natural extension of the methods for finding normals and derivatives for the G^1 spline (see Chapter 3).

4.4.2. Procedural Choice of S_2 and T_1

Since we regard G^2 splines as natural extensions of G^1 splines, we would like to keep them similar to each other. In particular, if the G^1 spline happens to be also G^2 continuous (as it would be if the set of interpolation points formed a regular n-gon), we would expect that our G^2 construction *preserve* the already curvature continuous curve. This means that each pair of subsegments of the G^2 spline must be a *subdivision* of the original segment of the G^1 spline.

Consider the original cubic with the control points C_i. It has some curvature κ_0^{orig} at C_0 and κ_1^{orig} at C_3. If these curvatures coincide with the prescribed curvatures, i.e. $\kappa_0^{orig} = \kappa_0$ and $\kappa_1^{orig} = \kappa_1$, we would like to get the original cubic back. In other words, in this case, the two segments constructed should be a subdivision of the original curve. This useful property can be utilized to determine the position of S_2 and T_1.

Depending on whether the original cubic curve does or does not have any inflection points, we apply one of the two procedures below for finding the necessary control points.

Case 1. The original cubic with control points C_i does not have inflections (Fig. 4.4). Assume that the equal curvature lines intersect the zero curvature line at P and Q. Let P^M and Q^M be the intersection points of S_1T_2 with the equal curvature lines, corresponding to the curvatures at C_0 and C_3 of the original single segment Bézier cubic. Also, let S_2^M and T_1^M be the inner Bézier points of the midpoint subdivision of the original cubic:

$$\mathbf{S}_2^M = \frac{1}{4}(\mathbf{C}_0 + 2\,\mathbf{C}_1 + \mathbf{C}_2), \tag{4.14}$$

$$\mathbf{T}_1^M = \frac{1}{4}(\mathbf{C}_1 + 2\,\mathbf{C}_2 + \mathbf{C}_3). \tag{4.15}$$

The curvature κ_0^{orig} can be expressed as follows:

$$\kappa_0^{orig} = \frac{2\,(\mathbf{S}_2^M - \mathbf{S}_1) \times \mathbf{t}_0}{3\,v_0^2} = \frac{2\,(\mathbf{P}^M - \mathbf{S}_1) \times \mathbf{t}_0}{3\,v_0^2}. \tag{4.16}$$

It follows that the position of P^M can be expressed as:

$$\mathbf{P}^M = \mathbf{S}_1 + \frac{3\,\kappa_0^{orig}\,v_0^2}{2\,\mathbf{t}_0 \times \mathbf{d}}\,\mathbf{d}, \tag{4.17}$$

where **d** is the unit vector in the direction of S_2T_1. Analogously,

$$\mathbf{Q}^M = \mathbf{T}_2 + \frac{3\,\kappa_1^{orig}\,v_1^2}{2\,\mathbf{t}_1 \times \mathbf{d}}\,\mathbf{d}. \tag{4.18}$$

It is understood that the velocities v_0 and v_1 are the velocities of the subsegments: $v_0 = |C_0S_1|$, $v_1 = |T_2C_3|$.

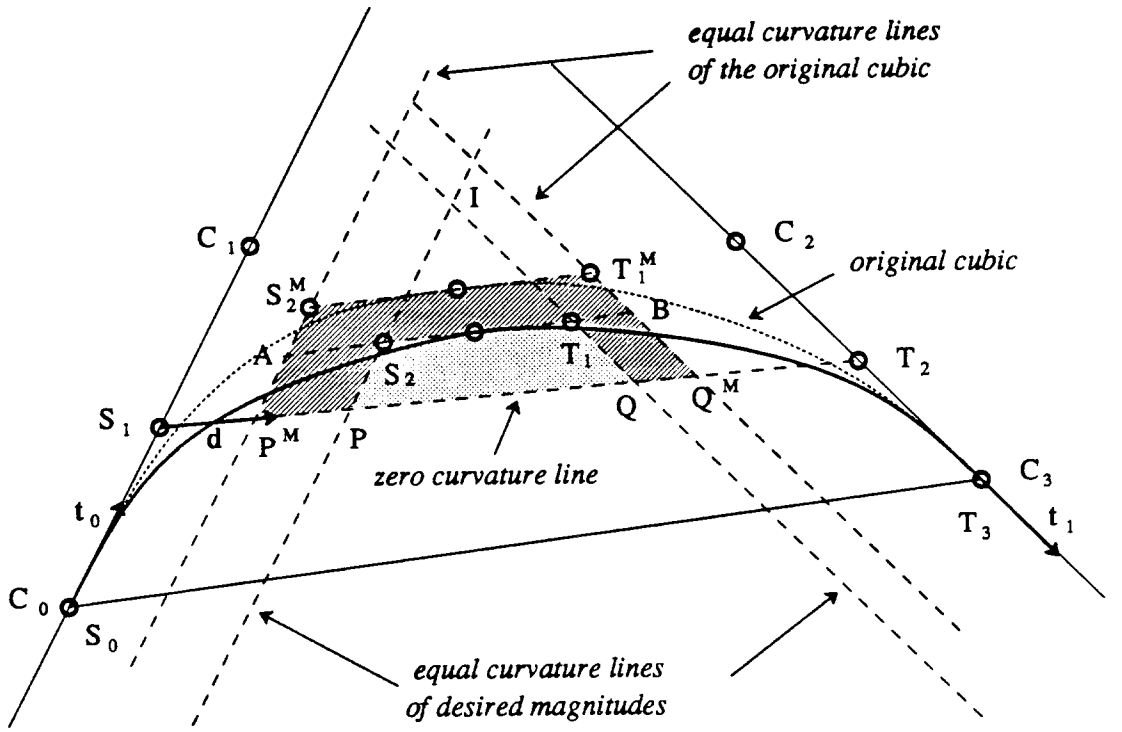


Figure 4.4: *Determination of the breakpoint offset (no inflection point). The two shaded trapezoids are similar.*

Finally, the offset of the breakpoint from the zero curvature line is defined from similarity of the trapezoids $P^M S_2^M T_1^M Q^M$ and PS_2T_1Q. The actual construction computes the ratio r of PQ to P^MQ^M. Then the points A and B on the curvature lines of the original cubic are found in such a way that the ratios P^MA to $P^MS_2^M$ and Q^MB to $Q^MT_1^M$ are all equal to r. The points S_2 and T_1 are then defined by the intersections of the AB line with the equal curvature lines, corresponding to the desired magnitudes.

This choice of S_2 and T_1 satisfies the subdivision property outlined at the beginning of this Section. Indeed, if the prescribed curvatures κ_0 and κ_1 coincide with the curvatures of the original cubic, then $P^M = P$, $Q^M = Q$, the two trapezoids will be identical, and $S_2 = S_2^M$, $T_1 = T_1^M$.

Unfortunately, this approach cannot be generally used if the original cubic curve has an inflection point. Although a pair of G^2 subsegments can still be constructed, the resulting

solution may not be practical. Indeed, if one of the equal curvature lines is nearly parallel to the zero curvature line, then the solution will be very sensitive to even small changes in prescribed curvature. This may result in extra inflection points or even loops in the subsegments. Therefore, a special method is used to deal with this situation.

Case 2. The original cubic has an inflection (Fig. 4.5). Let u be the parametric value, corresponding to the inflection point, i.e. $\mathbf{c}'(u) \times \mathbf{c}''(u) = 0,\ 0 < u < 1$. In this case, we choose the points S_1 and T_2 not by the formulas (4.4) and (4.5), but as the Bézier points of the subdivision of the original cubic at the parametric value u [10]:

$$\mathbf{S}_1 = (1-u)\,\mathbf{C}_0 + u\,\mathbf{C}_1, \tag{4.19}$$

$$\mathbf{T}_2 = (1-u)\,\mathbf{C}_2 + u\,\mathbf{C}_3. \tag{4.20}$$

This essentially assures that S_1T_2 intersects the equal curvature lines (and is not parallel to any of them).

The points S_2 and T_1 are then defined to lie on the S_1T_2 line at the intersection with the equal curvature lines; this would force zero curvature for any S_3 chosen on the S_1T_2 line. We select $S_3 = T_0$ on S_2T_1 according to the u-subdivision:

$$\mathbf{S}_3 = \mathbf{T}_0 = (1-u)\,\mathbf{S}_2 + u\,\mathbf{T}_1. \tag{4.21}$$

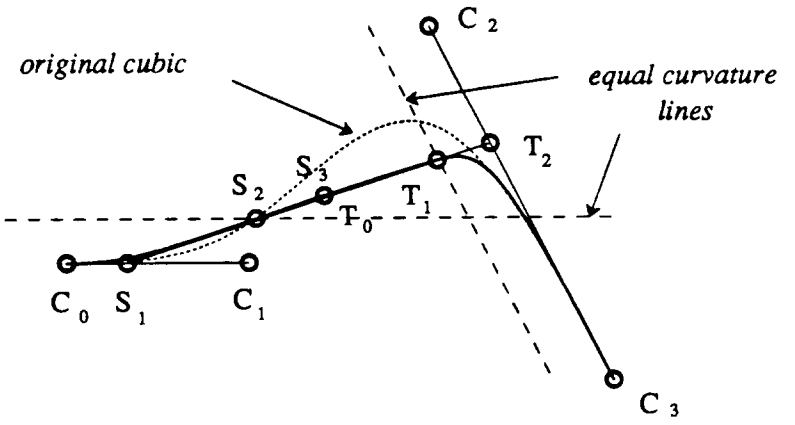


Figure 4.5: *Determination of* S_2 and T_1 *for the inflection case.*

Note that the construction process for S_2 and T_1 for the case without inflection points can also be used here, if the points S_2^M and T_1^M are replaced by the corresponding Bézier points of the u-subdivision. Indeed, in this case, both trapezoids will have zero widths and therefore could be regarded as similar. Finally, again, if prescribed curvatures at C_0 and C_3 coincide with the curvatures of the original cubic, the same curve will be constructed (and represented as two u-subdivision subsegments).

Stability Considerations. Unfortunately, the fact that two different methods are used to define the subsegments makes the approach unstable, i.e. the continuity of the spline dependence on the interpolation points cannot be generally guaranteed. The discontinuity occurs, of course, at the 'borderline' of the two cases. Suppose that the original cubic has an inflection exactly at one of the endpoints, i.e. at the parametric value of 0 or 1 (this will happen if the original Bézier points C_0, C_1 and C_2, or C_1, C_2 and C_3 are collinear). This situation can be regarded as a limiting state of either case. However, the resulting G^2 subsegments will not generally be the same.

Since the same construction process for S_2 and T_1 is used in both cases, the reason for instability is the discontinuity of the dependence of the parametric value of the breakpoint on the inflection point (Fig. 4.6). Note that the parametric value of the inflection point need not lie within the [0,1] interval: it is just a solution of the equation $\mathbf{c}'(u) \times \mathbf{c}''(u) = 0$.

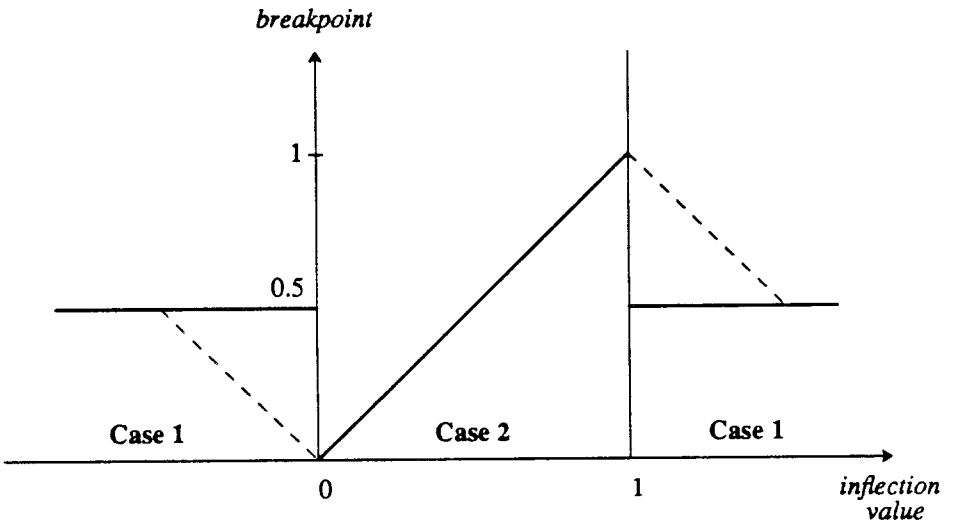


Figure 4.6: *Dependence of the parametric value of the breakpoint on the inflection point if different methods are used for the two cases.*
Dashed lines represent required corrections to make the compound method stable.

However, it is quite simple to make the graph continuous. The required corrections are shown dashed. Essentially, if the parametric value of the inflection point is close to 0 or 1, the breakpoint is also made close to 0 or 1. For example, for the inflection value $u = 1.1$, the breakpoint is inserted at $u = 0.9$, rather than 0.5. The points S_1 and T_2 are then determined by the formulas (4.19) and (4.20). The points S_2^M and T_1^M are defined by the corresponding Bézier points of the u-subdivision; S_2 and T_1 are determined as described in the case with no inflection points. Finally, the breakpoint is inserted according to (4.9).

4.5. Curves with Large Velocity Magnitudes

The equation $\mathbf{c}'(u) \times \mathbf{c}''(u) = 0$ which determines inflection points of a cubic Bézier segment is of degree 3. However, trivial algebra shows that the coefficient of u^3 is identically 0, and this is really a quadratic equation

$$Au^2 + Bu + C = 0, \tag{4.22}$$

where

$$A = a + b - c, \quad B = -2a + c, \quad C = a, \tag{4.23}$$

$$a = (\mathbf{C}_1 - \mathbf{C}_0) \times (\mathbf{C}_2 - \mathbf{C}_1), \quad b = (\mathbf{C}_2 - \mathbf{C}_1) \times (\mathbf{C}_3 - \mathbf{C}_2), \quad c = (\mathbf{C}_1 - \mathbf{C}_0) \times (\mathbf{C}_3 - \mathbf{C}_2). \tag{4.24}$$

Therefore, a single cubic Bézier polynomial with control points C_i, $i = 0,...,3$ can have 2 inflection points (Fig. 4.7a). A necessary (but not sufficient) condition for this is the fact that the line segments C_0C_1 and C_2C_3 intersect [125]. This will never happen if the default velocities from Section 3.3.3 are used and the value of the bulge parameter does not exceed its limit of 1.5. We still need to consider this case, because the user is at liberty to choose even 'unreasonably' high velocity magnitudes that could result in cusps or loops (Fig. 4.7b,c).

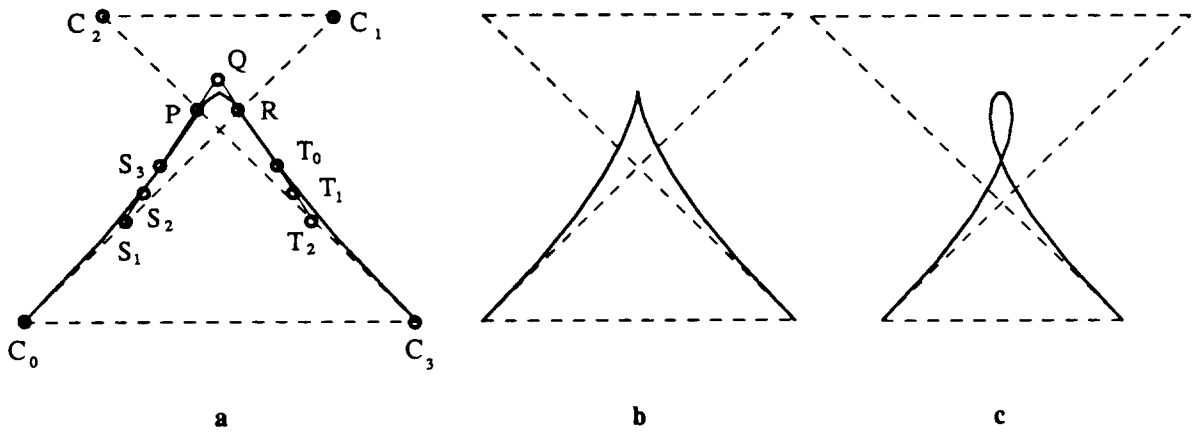


Figure 4.7: *A single cubic Bézier segment with large velocities.*
a. *Two inflection points.* **b.** *A cusp.* **c.** *A loop.*

It seems natural in this case to insert *two* breakpoints exactly at the inflection values u_1 and u_2. Thus, 3 G^2 subsegments will be used to represent a single span between two adjacent interpolation points. The control points of the subsegments are determined as follows:

$$\mathbf{S}_1 = (1 - u_1)\,\mathbf{C}_0 + u_1\,\mathbf{C}_1, \tag{4.25}$$

$$\mathbf{T}_2 = (1 - u_2)\,\mathbf{C}_2 + u_2\,\mathbf{C}_3. \tag{4.26}$$

Let points P and R divide the line segments C_2C_3 and C_0C_1 in the ratios corresponding to u_1 and u_2. Thus, P and R are control points of the corresponding subsegment of the u_1 or u_2-

subdivision of the original cubic (see below):

$$\mathbf{P} = (1 - u_1)\,\mathbf{C}_2 + u_1\,\mathbf{C}_3, \tag{4.27}$$

$$\mathbf{R} = (1 - u_2)\,\mathbf{C}_0 + u_2\,\mathbf{C}_1. \tag{4.28}$$

The intersection of lines S_1P and T_2R will define Q, a double interior control point of the middle segment. Then, the points S_2 and T_1 are defined to lie at the intersection of S_1Q and T_2Q lines with the corresponding equal curvature lines. Finally,

$$\mathbf{S}_3 = (1 - u_1)\,\mathbf{S}_2 + u_1\,\mathbf{Q}, \tag{4.29}$$

$$\mathbf{T}_0 = (1 - u_2)\,\mathbf{Q} + u_2\,\mathbf{T}_1. \tag{4.30}$$

Note that the middle G^2 subsegment is completely determined by the other two.

Finally, suppose that the prescribed curvatures at C_0 and C_3 match the curvatures of the original cubic. In this case, the two cubics with control points C_0, S_1, S_2, S_3, and S_3, Q, P, C_3 will define the u_1-subdivision of the original cubic. Analogously, the two segments with the control points C_0, R, Q, T_0, and T_0, T_1, T_2, C_3 will define the u_2-subdivision. Therefore, all three subsegments will represent the whole curve, and the preservation property outlined in the previous Section will be fulfilled.

Stability Considerations. The special case of using three G^2 subsegments will again result in instability of the method; more complicated blending functions are required. They are summarized on the diagram in Fig. 4.8, which shows possible parametric values of the two inflection points. Unless both u_1 and u_2 lie between 0 and 1, a single inflection value will be determined as indicated on the diagram. The actual breakpoint value will then be adjusted for stability by the function in Fig. 4.6.

The regions A and B correspond to the case when u_1 and u_2 are both close to the [0,1] interval, but are on the opposite sides (Fig. 4.9). In this case, a more sophisticated blending function is required. For example, for the region A, the influence of u_1 on the final inflection value will decrease if u_2 is closer to 1 or u_1 is closer to -0.5. Thus, the following weighting average could be used:

$$u = \frac{u_1\,(u_2 - 1)\,(u_1 + 0.5) + u_2\,(-u_1)\,(1.5 - u_2)}{(u_2 - 1)\,(u_1 + 0.5) + (-u_1)\,(1.5 - u_2)}. \tag{4.31}$$

A similar formulation with u_1 and u_2 interchanged is used for the region B.

If the quadratic equation (4.22) has no real roots, the breakpoint is usually inserted at 0.5. However, to maintain stability, one must detect when the determinant of (4.22) is 'close' to 0, because the case of a double inflection point corresponds to a cusp (Fig. 4.7), and in this case the breakpoint value should be equal to the parametric value of the cusp. Let

$$u_0 = -\frac{B}{2A} \tag{4.32}$$

be the double root of (4.22) if its determinant is 0, and

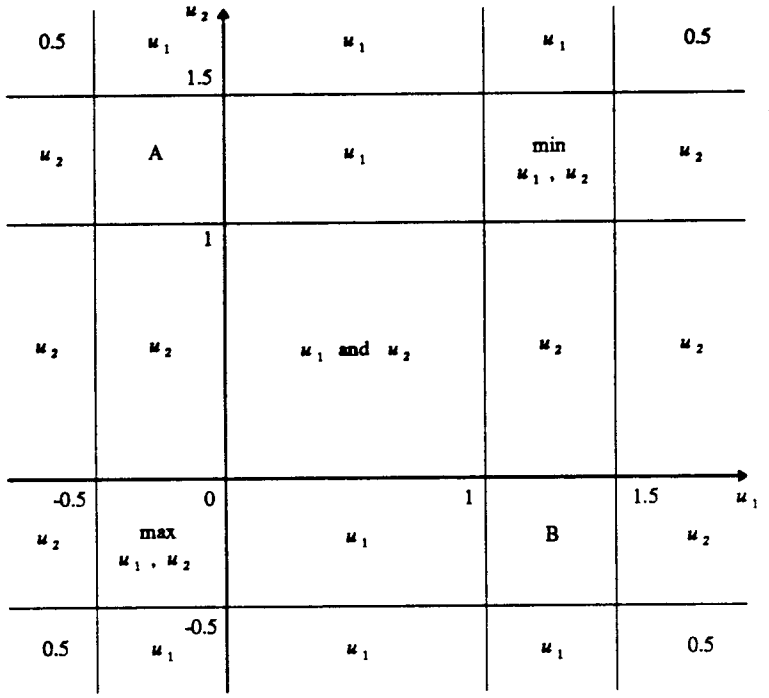


Figure 4.8: *Actual inflection values for various regions of the* (u_1,u_2) *plane.*

Figure 4.9: *Two inflection values corresponding to region* A *from the previous figure.*

$$d = \frac{\sqrt{-(B^2 - 4AC)}}{2|A|}, \tag{4.33}$$

where A, B, and C are as in (4.23). Then, if $d < 0.5$, the inflection value is computed as an appropriate weighted average of u_0 and 0.5:

$$u = \frac{u_0\,(0.5 - d) + 0.5\,d}{0.5}. \tag{4.34}$$

Again, the final breakpoint value is determined by the function in Fig. 4.6.

The fact that 3 segments per span may be required to construct a G^2 curve in some cases and the associated blending functions seem to make the method somewhat complicated. However, it is important to understand that the spline with 2 segments per each span will be sufficient for almost all practical applications. Three segments per span will be required only if the original cubic has 2 inflection points; this only happens if a product of certain ratios of C_0C_1 and

C_2C_3 (Fig. 4.7) lies in a finite interval [125].

Nevertheless, a method that always uses 2 G^2 segments per span would certainly be more elegant. In this case, the blending functions will also not be needed. At present, we have not yet succeeded in developing a robust approach that always produces predictable and reasonable solutions for all cases. More research is required on this subject.

4.6. Default Choice of Curvatures

We have described a way of building G^2 continuous curves with prescribed directions, velocities, and curvatures at each vertex. In the previous Chapter, we showed how to select directions and velocities for good pleasing results. Analogously, we now discuss an automatic way of choosing default curvatures at every vertex.

This choice of proper curvatures at the interpolation points is very important. Indeed, for many curvature values, no pleasing solutions can be constructed. This situation can be generally characterized by the fact that the points S_1, P, Q, and T_2 do not lie on the zero curvature line *in the indicated order* (Fig. 4.4).

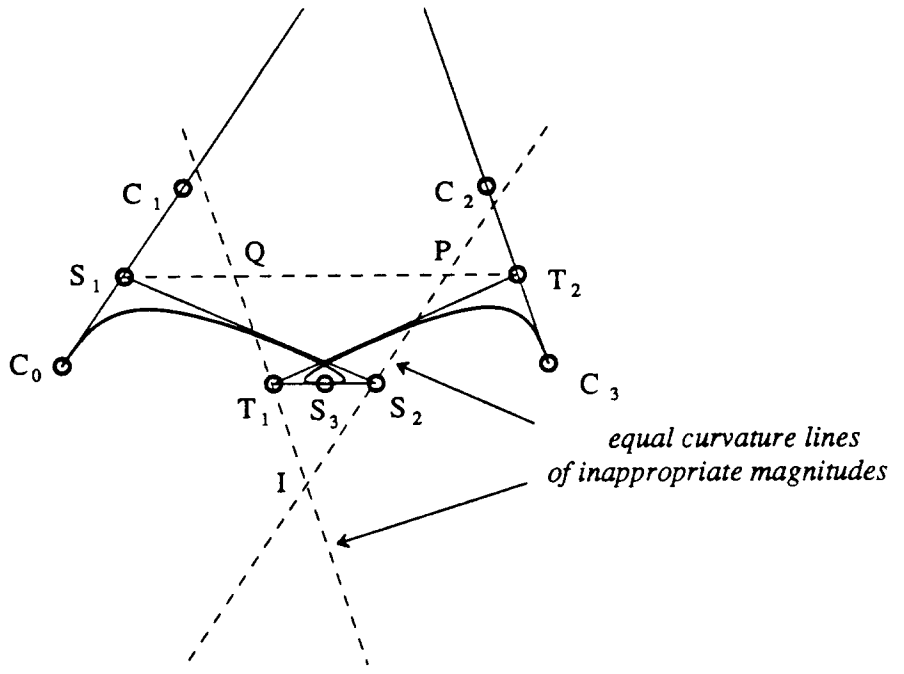


Figure 4.10: *Bad choice of curvatures and velocities at the vertices.*

Fig. 4.10 shows what happens if the curvatures κ_0 and κ_1 are selected to be too large. The problem is, of course, that the intersection point I of the two equal curvature lines falls 'below' the zero curvature line S_1T_2. Although a pair of G^2 subsegments can still be constructed, few

people would call this a pleasing solution due to the self-intersection. A practical system should notify the user that no pleasing spline can be built for the chosen set of values at the endpoints, but it should still be able to produce a solution. This solution can be obtained by applying an appropriate procedure from one of the cases above.

Of course, a 'bad' situation like the one above will not happen if the selected curvatures are equal to the curvatures of the G^1 spline. Therefore, the selected curvatures should be 'as close as possible' to the original ones. This will also keep the G^2 spline similar to the G^1 curve.

From the construction process of the points S_2 and T_1 it is clear that choosing equal curvature lines between the corresponding interpolating point and the equal curvature line of the original cubic will have no adverse effects on the shape of the curve. On the other hand, a large curvature magnitude or the 'wrong' sign can cause extra inflections or loops. For a G^1 spline, there are two curvatures at each interpolation point: κ_p (from the predecessor segment) and κ_s (from the successor segment). Therefore, a good choice for the curvature would be a minimum of these two values if the two curvatures are of the same sign. If the two curvatures are of the opposite signs, this really means that the curve should have an inflection at the current vertex. In this case, we assign the curvature to 0. This formulation also agrees well with the case of joining a linear (with 0 curvature) and a non-linear segments (Fig. 4.1). To summarize,

$$\kappa = \begin{cases} \pm \min(|\kappa_p|, |\kappa_s|) & \text{if } \kappa_p \kappa_s > 0 \\ 0 & \text{if } \kappa_p \kappa_s \le 0 \end{cases} \tag{4.35}$$

The sign of κ corresponds to the sign of κ_p and κ_s.

Note that the restriction to the prescribed curvature to lie between 0 and κ defined by the previous formula would leave no freedom to select a curvature value at a particular interpolation point if $\kappa = 0$ there. Indeed, one cannot construct a reasonable solution if, for example, a zero and nonzero curvatures are prescribed at the two ends of a linear segment. However, the given range of possible curvature values at each point should be sufficient to the designer: the curve can always be made flatter by decreasing a curvature magnitude at a particular vertex. On the other hand, decreasing the velocities (or the bulge parameter) would generally yield larger ranges of permissible curvatures where they are not restricted by the topology of the defining polygon.

4.7. Using Splines of Higher Order

We have presented a local way of constructing interpolating G^2 splines from cubic Bézier segments between original vertices and procedurally introduced breakpoints. In this section we investigate, for purpose of comparison, the implications of using higher order splines, such as quartics and quintics.

A quartic Bézier curve has 3 interior control points. Two of these control points will be completely determined by the given velocity values at the endpoints. Now, if we choose the remaining 'middle' control point to be at the intersection of the equal curvature lines (see Fig.

4.2), the resulting spline will satisfy all the prescribed information. Further discussion of using quartic planar and space curves can be found in [98].

However, equal curvature lines *do not* always intersect. Indeed, suppose that the directions $\mathbf{t}_0$ and $\mathbf{t}_1$ are parallel. Then equal curvature lines would also be parallel, and no quartic curve could satisfy the prescribed curvatures and velocities. If, in addition, curvatures are set to 0, then for *any* value of the velocities no interpolating quartic spline exists. The difficulties of using G^2 quartic splines are also discussed in [83]. Thus, using a quartic spline is really not an improvement over using a single cubic between interpolation points.

Quintic splines, on the other hand, have enough degrees of freedom for an arbitrary choice of directions, velocities, and curvatures. Indeed, the two 'outer' control points will be determined by the velocities, and the two 'inner' control points can be placed anywhere on the equal curvature lines. The position of these points can then be derived from some similarity conditions with the control points of the G^1 cubic spline, degree-elevated to a quintic. Thus, the G^2 method discussed in this Chapter has the following shortcomings compared to quintic splines:

- Two or three segments are required for a single span between adjacent vertices;
- The blending functions that ensure stability of the method are rather complicated;
- Curvature choice at the interpolation points is confined to a range determined by the normals and velocities of the G^1 spline. That is, the curvature at each vertex has to lie within the $[0,\kappa]$ (eq. (4.35)) interval;
- As we will see later (Section 4.9), the method cannot be easily extended to $\mathbf{R}^3$.

We believe, however, that the proposed method is nevertheless a good alternative to quintics for the following reasons:

- Only cubic Bézier curves are used. Although there is a fixed computational overhead involved in constructing breakpoints for cubic G^2 splines, this overhead does not depend on the number of evaluations needed to render the curve. Cubic splines with breakpoints can be rendered more efficiently than single segment quintics;
- Three segments per span and the associated complicated blending functions are needed only if the user assigns 'unreasonably' high values to the velocities;
- It is often the case that a designer needs to construct a curve through a given set of points without restrictions on velocities and curvatures. In this case, the method will always produce a reasonable solution with two cubic segments per each span.

4.8. Comparison of Various Methods

In this Section, we consider several methods of local interpolation with Bézier curves and analyze the tradeoffs between them (Table 4.1). The following methods are compared:

1. Natural cubic spline;
2. A combination of methods [19] and [56] that adjusts velocities by solving the corresponding quadratic system of equations for each curve segment and inserts breakpoints when no solution exists;
3. A procedural G^2 method, described in this Chapter;
4. Quartic splines method;
5. Quintic splines method.

Property	Method				
	Natural splines	1 or 2 segm if needed	Procedural G^2 splines	4th order Bezier	5th order Bezier
Direction choice	no	yes	yes	yes	yes
Curvature choice	no	limited	limited	usually yes	yes
Velocity choice	no	no	yes	yes	yes
Continuity order	C^2	G^2	G^2	usually G^2; sometimes G^1	G^2
Stability	yes	no	yes	yes	yes
Efficiency considerations	Need to solve global linear systems	Need to solve quadratic systems	Two cubics per segment	High order	Even higher order

Table 4.1: *Comparison of various local methods.*

We can see from this table that procedural G^2 splines combine efficiency (cubics; no need to solve numerical systems), flexibility (choice of shape parameters), and stability. We have shown that for planar curves, the relatively high cost of quintic splines can be reduced by using pairs of cubics without any loss in the degree of continuity or in the adjustability of the curve.

The following three figures show G^2 curves and the corresponding curvature plots. In the first two figures, the normal direction was based on the angle bisector (Section 3.3.2.2), while the Collinear Segments rule (Section 3.4.2) was applied to the last figure. The velocities and curvatures for all three figures were selected according to the appropriate default procedures (Sections 3.3.3.3 and 4.6). The interpolation points correspond to longer ticks on the horizontal axis of the curvature plots, while the shorter ticks represent the breakpoints.

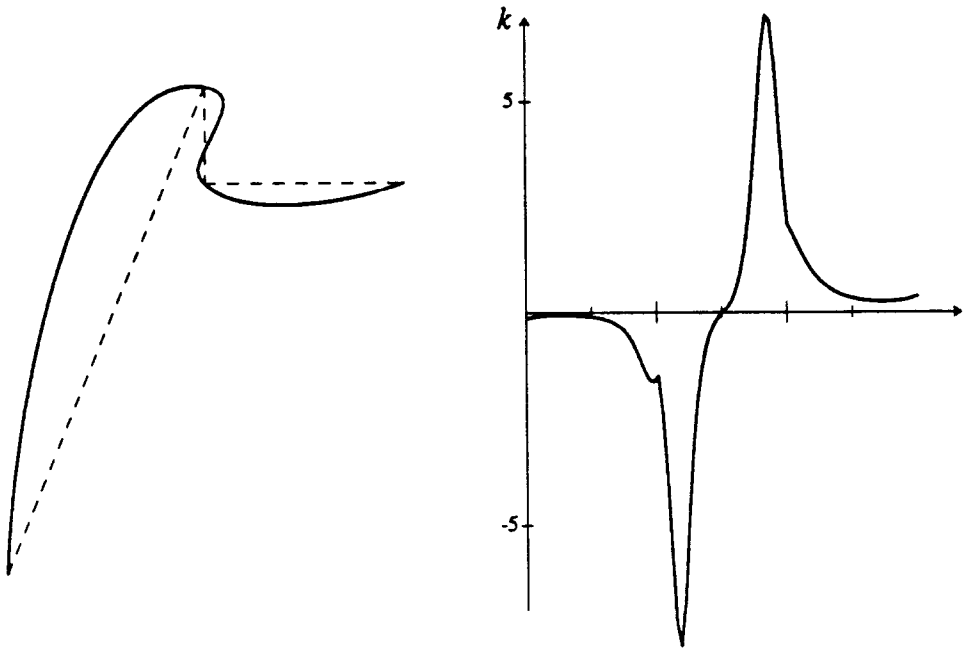


Figure 4.11: *A G^2 curve and a corresponding curvature plot.*

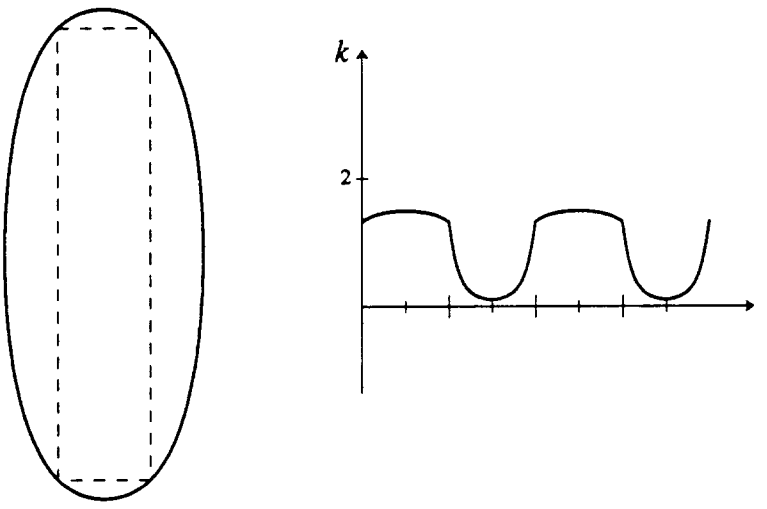


Figure 4.12: *A G^2 curve and a corresponding curvature plot.*

4.9. Extensions to R^3

The G^1 spline, discussed in the previous Chapter, can be readily utilized to interpolate a set of points in space as well as in the plane. Unfortunately, the cubic G^2 spline cannot be extended as easily to R^3. This is due to the fact that in addition to tangent directions and curvature values

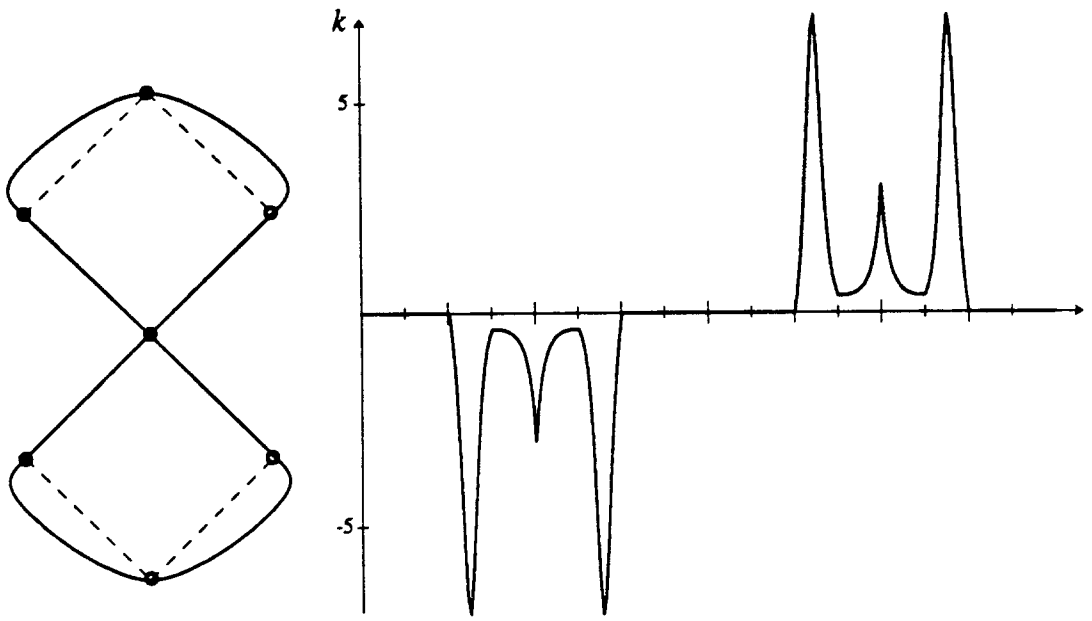


Figure 4.13: *A G^2 curve and a corresponding curvature plot. Collinear Segments rule was used for normal selection.*

at interpolation points, *osculating planes* also have to be matched between subsequent segments (Fig. 4.14). For cubic G^2 Bézier curves this means that 5 points $C_{i-1,s}$, C_{ip}, P_i, C_{is}, and $C_{i+1,p}$ must be coplanar. This extra constraint is specific to $\mathbf{R}^3$: for planar interpolation sets, osculating planes at every vertex trivially coincide with the plane of the defining polygon.

The problem of selecting a common osculating plane at interpolation points presents a major difficulty of using a step-by-step approach of constructing planar G^2 curves in $\mathbf{R}^3$. A straightforward generalization of the $\mathbf{R}^2$ method could proceed as follows:

- Construct a G^1 curve and compute normals and velocities at each interpolation point;
- Compute the two curvatures and the two osculating planes from the predecessor and the successor segments (Section 4.6);
- Heuristically define a common curvature value and a common osculating plane at each vertex;
- Use a procedure similar to the one discussed in Section 4.4.2 to define the appropriate control points of the two subsegments.

Fig. 4.15 illustrates this process. Curvature continuity at the breakpoint requires that the points S_1, S_2, S_3, T_1, and T_2 be coplanar.

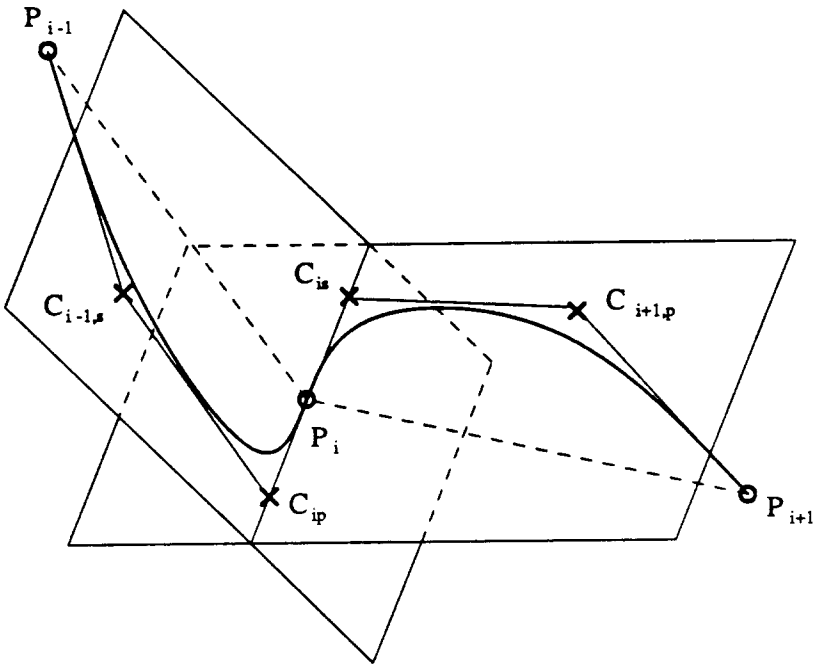


Figure 4.14: *Two cubic segments in space.*
The two osculating planes must coincide for G^2 continuity.

However, for some 'unfortunate' choices of osculating planes no solution at all can be obtained. For example, if the osculating planes as well as the tangent directions at a pair of adjacent vertices are parallel, then for any choice of the plane $S_1S_2S_3T_1T_2$, the curvatures of the two subsegments at the breakpoint will always be of opposite signs (Fig. 4.16). Thus, no G^2 spline can be constructed.

A possible way to deal with this problem is to try to choose osculating planes in such a way that a pair of G^2 cubic subsegments can always be constructed. Unfortunately, 'fixing' an osculating plane at a certain vertex could trigger a 'ripple effect', i.e. osculating planes at the neighbor vertices would also have to be adjusted, and so on. It seems that only a global scheme could assign osculating planes to the vertices to guarantee a G^2 cubic solution.

Despite the above problem, a G^2 cubic spline through a given set of points in $\mathbf{R}^3$ can still be built. A trivial solution could just force zero curvatures at all the interpolation points and at the breakpoints by doubling the interior control points of each cubic subsegment. However, the resulting curves are usually of poor visual quality.

A possibly better solution would force zero curvatures at interpolation points only, and would require each pair of cubic subsegments to be planar. Then a $\mathbf{R}^2$ procedure for finding control points of the subsegments could be used. Of course, in this formulation, the freedom of choosing tangent directions at the vertices will have to be sacrificed; it will be defined by the

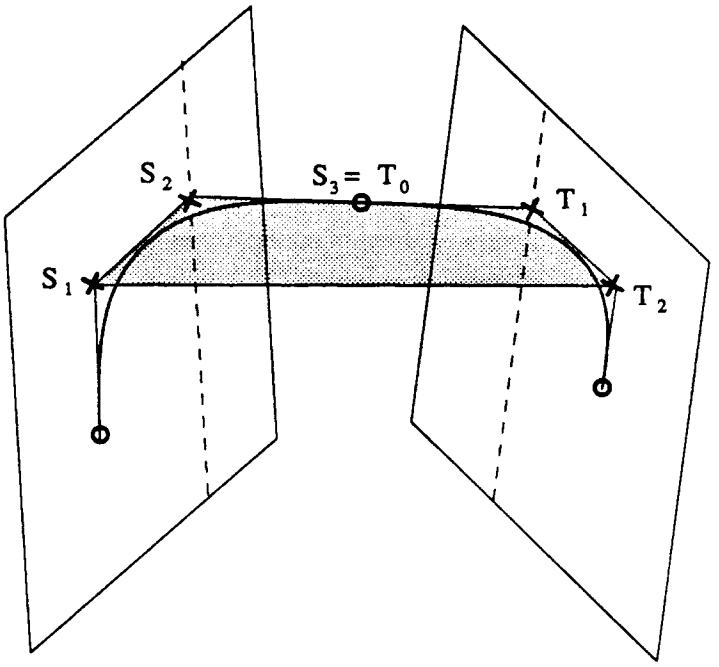


Figure 4.15: *Two curvature continuous cubic segments in* $\mathbf{R}^3$.
The 5 points defining the shaded area are coplanar.

intersection of the planes of the adjacent curve segments.

Representation of G^2 curves in $\mathbf{R}^3$ with cubic Bézier curves is an open research issue at this point. It may well be the case that the cubic curves do not possess enough degrees of freedom to construct curvature continuous space curves of good visual quality. Using quintic splines may be the natural alternative for $\mathbf{R}^3$.

4.10. Summary

We have presented a G^2 method for local interpolation with cubic Bézier curves. This method is a direct extension of the G^1 method from the previous Chapter. The extra degree of geometric continuity is achieved at the cost of using two cubic segments between every pair of vertices.

At every interpolation point, the user can specify and locally modify the *normal*, the two *velocities*, and the *curvature* magnitude. However, default values for these *shape parameters* can always be determined automatically. The default normals and the velocities come from the G^1 method (Chapter 3), which together with the proper curvature selection guarantees that the spline will be of good visual quality.

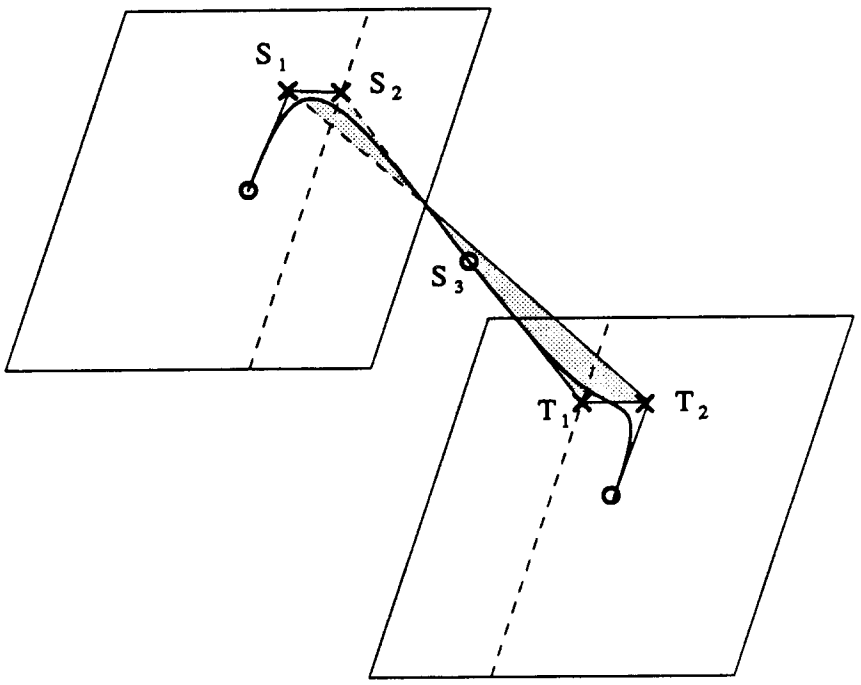


Figure 4.16: *Two subsegments cannot be G^2 at the breakpoint due to the choice of osculating planes.*

A local cubic G^2 interpolating spline with shape parameters is new to the CAGD. Previous approaches either were global [1, 28, 92], had no shape parameters [23], or were of a higher order [34, 65, 69].

5

Procedural Construction of Boundary Curves

This chapter describes a natural extension of procedural curve interpolation (Chapter 3) to the generation of boundary curves that define interpolating surfaces. Given a set of interpolation points in $\mathbf{R}^3$ connected by edges into a network of quadrilateral and/or triangular faces, we replace each original edge with a cubic boundary curve. Our procedure determines surface normals at each interpolation point, tangent directions for cubic boundaries, and, finally, the derivative magnitudes of all curves meeting at the vertex. Various rules for curve construction are generalized, and new rules that apply to surfaces only are introduced. As a result, the method produces surfaces of high visual quality for most situations.

5.1. Introduction

In this Chapter, we extend the procedural method of curve interpolation to surfaces, i.e. to the construction of smooth surfaces through a given set of points in $\mathbf{R}^3$. The basic goals we are trying to achieve in both cases are very similar. The surface is constructed from a sequence of procedural steps that may take special situations into account. As a result, the final surfaces produced with this method are closer to what a designer would expect than the ones generated by traditional techniques.

As in the case of curves, we require that the procedural surfaces satisfy some desirable properties. These properties include coordinate system independence, local control, simplicity of representation, stability, and visual quality. Please refer to Section 3.2.1 for a detailed discussion of these properties.

The problem of constructing a smooth surface through a set of points can be conceptually split into 3 separate stages.

1. The topology of underlying vertex set is defined by connecting the vertices by *edges* and arranging edge-bounded regions into *faces* (not necessarily planar). The topological structure is usually known a priori by the application or is given explicitly by the user. However, this step could also be performed by Delaunay triangulation or similar algorithms [109].

2. The edges defined in the previous step are substituted by curves. At each interpolation point, these curves must share a common tangent plane to permit the final surface to be G^1. This step can be split further into 3 substeps: first, a *vertex normal* is defined at each interpolation point; these will also be the normals of the final surface at the interpolation points. Second, suitable *directions* for each edge are determined on the tangent plane defined by the vertex normal. Third, the final shapes of the boundary curves are defined by placing Bézier points on the corresponding tangent lines. Each of these steps will be described in detail in this Chapter.

3. The curve mesh, produced at the previous step, defines surface patches that must be filled in. Each curve in the mesh is a boundary of 2 neighbor patches. The surface patches must be defined so that G^1 continuity is maintained across these boundary curves. This step will be discussed in Chapter 6.

This conceptual pipeline of surface construction provides a high degree of modularity and flexibility to a modeling system, because changes can be made at any step in the design process. This approach is clearly preferable over schemes that produce a surface directly from a set of interpolation points or from a topological mesh. In these latter methods, construction of the boundary curves and surface patches is typically performed simultaneously [81, 106, 116].

5.2. Geometric Parameters

In Chapter 3, we have chosen Bézier cubics for representation of interpolating curves because of their simplicity, wide acceptance, and ease of local control. For the same reasons, we use Bézier cubics to represent boundary curves in interpolating surfaces. Thus, the vertex normals and velocities of the boundary curves at a shared interpolation point will be the natural analogs of the geometric parameters of the interpolating curve discussed in Chapter 3. In addition, however, *tangent directions* for each boundary are required. Since the surface normal only determines the plane that contains the tangent directions, they must also be regarded as geometric parameters (that have no counterpart in a univariate case).

For each vertex, the construction of boundary curves proceeds as follows (Fig. 5.1):

1. Vertex neighbors are identified using the topological structure of the mesh. A vertex can have an unlimited number of neighbors.

2. A vertex normal is chosen. This normal defines a tangent plane at the current interpolation point.

3. Suitable tangent directions for each boundary curve meeting at the current vertex are selected. These directions are constrained to lie in the tangent plane defined by the vertex normal.

4. Suitable velocities are chosen for all relevant boundary curves. Equivalently, the Bézier points are placed along the tangent lines defined in the previous step. For a particular curve, the position of the Bézier point C_k is determined by the velocity magnitude v_k:

$$\mathbf{C}_k = \mathbf{P}_0 + \frac{1}{3}\, v_k \cdot \tag{5.1}$$

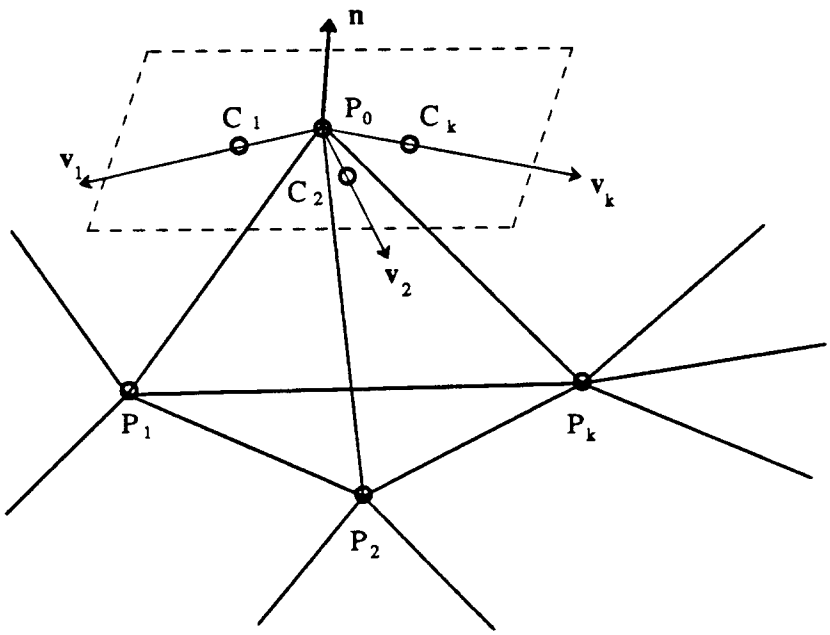


Figure 5.1: *Geometric parameters of boundary curves: vertex normal, tangent directions, and velocities.*

Clearly, modifying any of the above geometric parameters will alter the shape of the resulting curve mesh, and thus the resulting surface. This fact has been mentioned in the literature [31, 99, 116]. However, the default choice of these parameters is not an easy one. In the following Sections, we will study and compare various methods of selecting vertex normals, tangent directions, and velocity magnitudes with the goal of producing surfaces of high visual quality.

5.3. Determination of the Normal

As mentioned above, determination of suitable vertex normals is the first step in the surface construction process. In this Section, we review several approaches to normal determination, based on the normals of the faces of the defining mesh, on the edge directions of the mesh, and on some least squares methods.

5.3.1. Methods Based on Face Normals

For the case of interpolating curves, the normal can be defined as a weighted average of the normals of adjacent defining polygon sides (Section 3.3.2). An obvious generalization to surfaces is to represent the normal as a weighted average of the normals to the faces joining at the given vertex. Thus, if m edges meet at a vertex, m normals defined by pairs of adjacent edges must be computed. We now look at several natural choices for the weighting factors.

Arithmetic Mean. The easiest way to define a normal is to use an arithmetic mean of all face normals:

$$\mathbf{n}_{MN} = \sum_{i=1}^{m} \hat{\mathbf{n}}_i, \tag{5.2}$$

where $\hat{\mathbf{n}}_i$ are unit face normals. This method may produce a lopsided bulge in the surface if there are several adjacent faces with small angles at the vertex (Fig. 5.2).

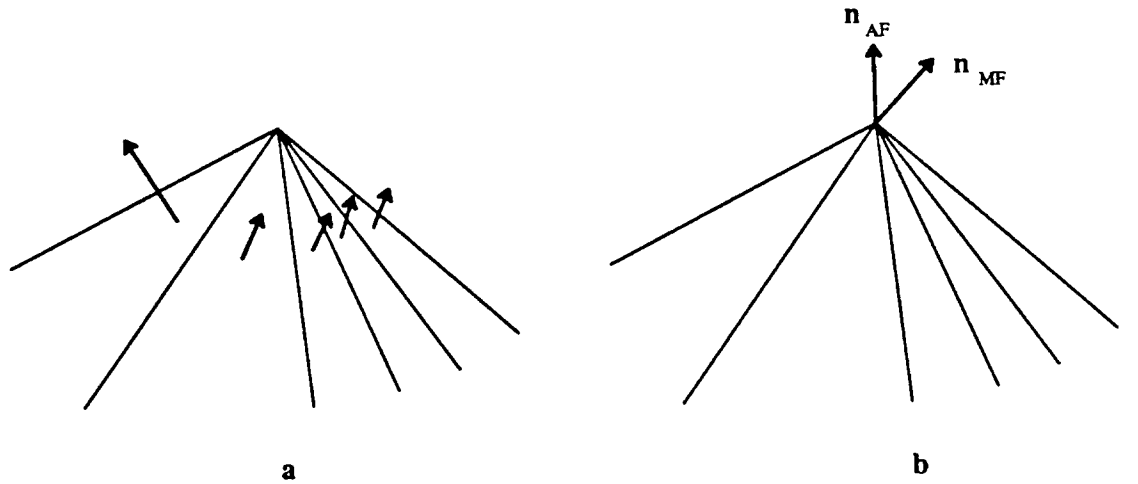


Figure 5.2: *Face based normal methods.*
a. *Face normals.* **b.** *Arithmetic Mean and Angle Average normals.*

Angle Average. The drawback of the previous method can be avoided if the face normals are weighted with the angle α_i between the corresponding two edges:

$$\mathbf{n}_{AN} = \sum_{i=1}^{m} \alpha_i \hat{\mathbf{n}}_i, \tag{5.3}$$

Additionally, this method is *tessellation-independent*. Indeed, if an extra edge is added on one of the faces, the total combined influence on the normal of the two split subfaces will remain the same due to angle weighting. Finally, the Angle Average method can be regarded as a generalization of the Bisector normal (Section 3.3.2.1), because $\mathbf{n}_{AN}$ is independent of the lengths of the polygon edges and generally lies within the pyramid of extended edges (see Section 5.3.2).

Directly Weighted Normals. This is a direct generalization of Catmull–Rom method for curves. Again, the final normal is a weighted average of face normals, this time with the weight factor being the area defined by the pair of adjacent edges:

$$\mathbf{n}_{DN} = \sum_{i=1}^{m} A_i \, \hat{\mathbf{n}}_i , \tag{5.4}$$

Note that areas A_i depend only on the two edges joining at the vertex; it is not the area of the whole face in the mesh which can have many sides that need not be planar.

This method has problems similar to the ones that Catmull–Rom method has for curves, i.e. it leads to asymmetric surfaces if the areas of any two faces at the current vertex differ by a significant amount (cf. Fig. 3.3a).

Inversely Weighted Normals. It can be argued that a small facet in the topological mesh conveys strong information about the normal direction. Then smaller face areas should be more important in determining the normal. In this case, the weight factor is the inverse of the corresponding face area:

$$\mathbf{n}_{IN} = \sum_{i=1}^{m} \frac{1}{A_i} \hat{\mathbf{n}}_i , \tag{5.5}$$

This method is also a generalization of the corresponding approach for interpolating curves.

5.3.2. Duality between Face Normals and Extended Edges

Consider a vertex that has m neighbors in the input polyhedral mesh. Then there are m edges meeting at the vertex naturally forming a pyramid (Fig. 5.3). This pyramid can be extended symmetrically through the vertex, resulting in the *pyramid of extended edges*. Let the unit directions of the extended edges be be $\hat{\mathbf{l}}_i$, and angles between pairs of adjacent edges be $\alpha_{i,i+1}$. Finally, let the dihedral angles between adjacent faces be β_i.

There always exists a *dual* of this pyramid: a *pyramid of face normals*. It is formed by the unit normals $\hat{\mathbf{n}}_{i,i+1}$ to the faces of the pyramid of the original edges of the mesh. It can be easily seen that the angles between the adjacent normals $\hat{\mathbf{n}}_{i-1,i}$ and $\hat{\mathbf{n}}_{i,i+1}$ are $\pi - \beta_i$ and that the dihedral angles in the pyramid of face normals are $\pi - \alpha_{i,i+1}$.

This duality suggests that all methods that are based on weight averaging of face normals have their analogs for extended edges, and vice versa (see Section 5.3.3). The next logical step would be to unite the two approaches with a goal of producing a method that always gives a vertex normal that lies within both pyramids. Such a method would be a generalization of the same

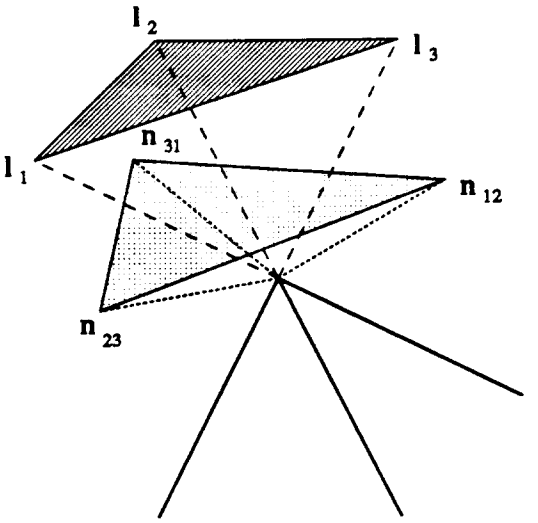


Figure 5.3: *Pyramids of extended edges and face normals. Face normals are shown dotted, and extended edges dashed.*

basic approach that was used for interpolating curves in Section 3.3.2.2.

Unfortunately, this unification may not be possible for surfaces. For curves, the angle bisector of the defining polygon could be formed by averaging either the two side normals or the two edge extensions. This is no longer true for the pyramids of the extended edges or of the face normals which can be concave and have arbitrarily many edges. The two approaches can only be combined if constants a_i and $b_{i,i+1}$ are found such that

$$\sum a_i \, \hat{\mathbf{l}}_i = \sum b_{i,i+1} \, \hat{\mathbf{n}}_{i,i+1}. \tag{5.6}$$

These constants must not depend on $\hat{\mathbf{l}}_i$ and $\hat{\mathbf{n}}_{i,i+1}$. It may well be that a_i and $b_{i,i+1}$ do not exist for a general case. An alternative approach would to be to form the intersection of the two pyramids, and define the vertex normal in the direction from the vertex to the center of gravity of their intersection. Although this method will clearly generate a desired normal, it is computationally very expensive and therefore may not be practical.

5.3.3. Edge Based Normal Methods

We now list normal methods based on extended edges that are the duals of the face based methods of the previous Section. Again, these methods are generalizations of the corresponding methods for curves. They can only be used when the 3D angle at the tip of the vertex is 'acute'. Indeed, when the mesh edges are nearly planar or form a saddle surface (i.e. intersection of the pyramid of extended edges with any half-space with the current vertex on its bounding plane is not empty), a weighted average of extended edges can be zero.

Arithmetic Mean. In this approach, the average of all edge directions is used:

$$\mathbf{n}_{ME} = \sum_{i=1}^{m} \hat{\mathbf{l}}_i . \tag{5.7}$$

Dihedral Angle Average. Since the dual of the angle $\alpha_{i,i+1}$ between the two edges is $\pi - \beta_i$, where β_i is the dihedral angle between the two faces, we have:

$$\mathbf{n}_{AE} = \sum_{i=1}^{m} (\pi - \beta_i)\, \hat{\mathbf{l}}_i . \tag{5.8}$$

Like its dual $\mathbf{n}_{AN}$, this method is tessellation-independent in the sense that adding an extra edge to an already existing face will not change the resulting normal. Indeed, the dihedral angle at the newly added edge will be equal to π so that this edge will have no influence on the final normal.

Directly Weighted Edges. In this case, the weight factor is just the length l_i of the corresponding edge:

$$\mathbf{n}_{DE} = \sum_{i=1}^{m} l_i\, \hat{\mathbf{l}}_i . \tag{5.9}$$

Inversely Weighted Edges. Finally, one can also use inverse edge lengths as a weight factor:

$$\mathbf{n}_{IE} = \sum_{i=1}^{m} \frac{1}{l_i}\, \hat{\mathbf{l}}_i . \tag{5.10}$$

5.3.4. Methods Based on Least Squares Approximation

Rather than representing a normal as a weighted average of face normals or extended edges, other means can be used. For example, one can interpolate the current vertex and approximate its nearest neighbors with a simple known surface, such as a sphere, an ellipsoid, or a quadric surface. This surface can be found by the well-known least squares minimization technique. The vertex normal is then defined to be the normal of this approximation surface at the vertex.

5.3.4.1. Best Spherical Fit

As mentioned above, one of the choices for an approximating surface is a sphere that passes through the current vertex [99]. The best-fitting sphere can be computed using least squares method. The resulting normal in this case is very close to the Inversely Weighted normal $\mathbf{n}_{IN}$ (Section 5.3.1), because of the strong influence of closely spaced points (Fig. 5.4). However, the least squares method is computationally more expensive than $\mathbf{n}_{IN}$. More importantly, the method fails if the nearest neighbors form a saddle surface.

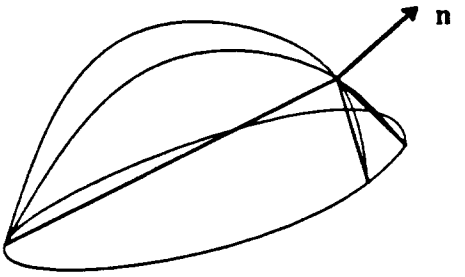


Figure 5.4: *Normal computed with best-fitting sphere method.*

5.3.4.2. Best Fitting Quadric Surfaces

A less restrictive approach of the same general idea is to use best-fitting quadratic surfaces instead of spheres. Assuming that a coordinate system has been defined with the current vertex at the origin, its neighbors can be approximated with a quadric surface of the type

$$f(x,y) = A\ x^2 + B\ xy + C\ y^2 + D\ x + E\ y. \tag{5.11}$$

The coefficients A, B, C, D, E are then determined by the least squares method.

Although the best-fitting surface will always exist, this approach has an obvious disadvantage. Indeed, if the vertex has less than 5 neighbors, it is generally possible to construct several different quadric surfaces that actually *interpolate* the neighbors. One possible solution to that is to set some of the coefficients (for example, D and E) to 0. This may not be enough, however. For example, vertex neighbors may lie on intersections of an ellipsoid and a hyperboloid.

Another possible way to deal with non-uniqueness of the best-fitting surface is to take second nearest neighbors (i.e. neighbors of neighbors) into account, so that the quadric surface approximates them as well. While this approach is feasible, it is even more computationally intensive than the best-fitting sphere method. Moreover, it is still possible to have vertex constellations that permit non-unique interpolating quadrics.

5.3.5. Discussion

In this Section, we have presented various approaches to the normal selection. These approaches can be split into 3 groups: methods based on face normals (Section 5.3.1), edge based methods (Section 5.3.3), and methods based on least squares techniques (Section 5.3.4). The last 2 groups have serious drawbacks: the edge based methods can't handle saddle surfaces, while the least squares approaches may have the same problem in addition to being computationally expensive.

Methods based on the face normals also have several problems, as discussed in Section 5.3.1, but at least they are robust enough to be used in most configurations of interpolation points. Rather than discussing their tradeoffs against each other, we proceed directly to the problem of

the selection of the tangent directions. We will show that for best results, the normals and the tangent directions need to be chosen in concert, as in the Opposite Edge method. In Section 5.6.2, formal evaluations will show that this method is indeed preferable over the others.

5.4. Determination of Tangent Directions

Once the normal at a vertex has been determined, the next step is to define tangent directions for each boundary curve meeting at that vertex. Obviously, these tangent directions must lie in the tangent plane defined by the normal. As mentioned above, this step has no counterpart in the construction of interpolating curves.

5.4.1. Projection Method

The most obvious way to define the tangent directions is to project the mesh edges onto the tangent plane (Fig. 5.5a). Thus, a tangent direction corresponding to a particular mesh edge will depend only on the vertex normal and this edge. Therefore, this method is tessellation-independent because adding an extra edge does not change tangent directions corresponding to other edges.

For this method, however, the normal *has to* lie within the pyramid of extended edges. Indeed, when the normal is collinear with one of the edges, the tangent direction of this edge becomes undefined. Moreover, if the normal is perturbed slightly near this position, the tangent direction will exhibit wildly unstable behavior.

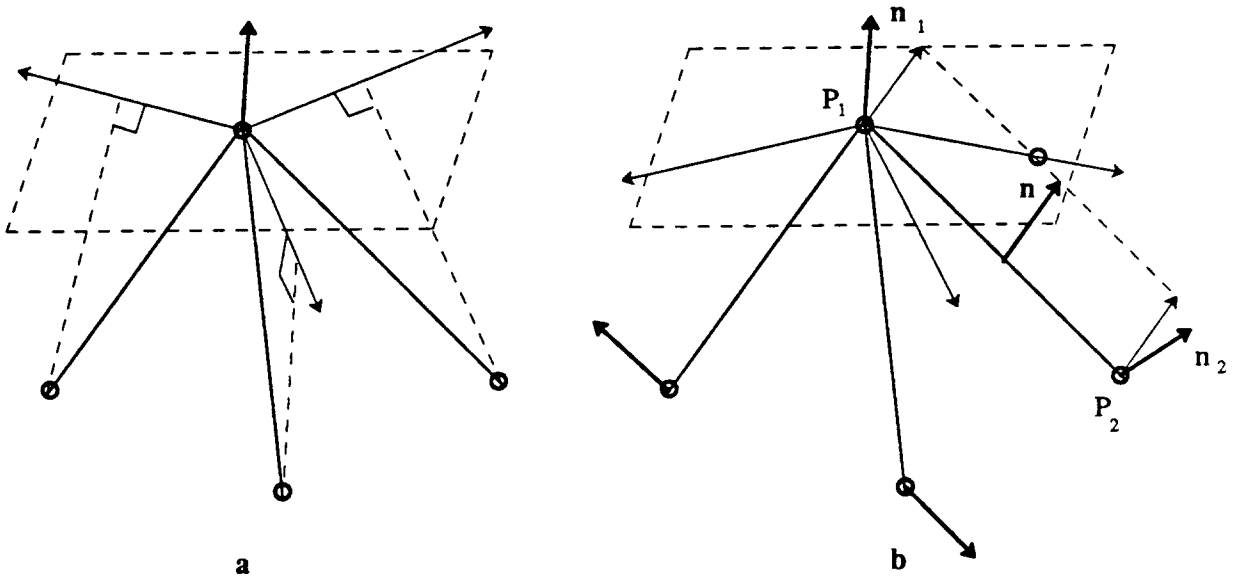


Figure 5.5: *Methods for tangent directions.*
a. *Projection.* **b.** *Planar boundary.*

5.4.2. Planar Boundary Method

One could argue that boundary curves should be made as simple as possible. With respect to defining tangent direction, simplicity can be translated to the goal of producing planar boundary curves. Suppose we need to construct a boundary curve between vertices P_1 and P_2 (Fig. 5.5b). We compute the average $\mathbf{n}$ of vertex normals $\mathbf{n}_1$ and $\mathbf{n}_2$ and define the plane of the boundary curve by $\mathbf{n}$ and P_1P_2. The tangent directions are then given by the intersection of this plane with the respective tangent planes. The Planar Boundary method is also tessellation-independent.

Similarly to the Projection Method, this method can fail for extreme situations when $\mathbf{n}_1$ and $\mathbf{n}_2$ differ strongly. Even if we disregard the case when $\mathbf{n}_1 = -\mathbf{n}_2$, tangent directions may lie in the 'wrong' order on the tangent plane. This means that the circular order of the projections of the vertex neighbors onto the tangent plane may not be the same as the circular order of corresponding tangent directions. In the final surface a wrap-around cusp may then result and the G^1 continuity condition may be violated.

5.4.3. Opposite Edge Method for Normal and Tangents

The fact that both methods for tangent directions may fail for certain constellations of interpolation points can be explained by the use of rather simplistic approaches. Indeed, in either case, a tangent direction depends only on the vertex normals and on the underlying edge and not on other vertex neighbors. A more sophisticated approach should take the neighbor information into account. Of course, as a result, the property of tessellation independence may be lost.

Projection and Planar methods not only fail for strongly asymmetric meshes, but they can also produce clearly undesirable boundary curves. As an example, consider a torus that is sampled at 4 uniformly spaced points on each of 4 small circles placed at 4 uniformly spaced cuts of the big circle. The resulting quadrilateral mesh is shown in Fig. 5.6a. Assume that the normals at all 16 points are set to coincide with the torus normals at these points.

At the points on the outer and the inner big circles, proper tangent directions will clearly be produced due to symmetry. However, both Projection and Planar methods will generate a straight boundary line between points P_1 and P_2, whereas it is clear that this boundary should be close to a circular arc. Therefore, the resulting surface will have undesirable bulges on the inner side (Fig. 5.6b). A surface with better boundary curves is produced by the Opposite Edge method discussed below (Fig. 5.6c).

This points out the general problem — neither the choice of the vertex normal nor of the tangent directions can be very successful if they are carried out in a myopic way that only looks at minimal information and ignores any additional evidence that could reveal more clearly the overall geometrical context. The Opposite Edge method is an improvement in that it at least integrates the selection of the vertex normals with the tangent directions resulting in smooth boundary curves through the given interpolation point.

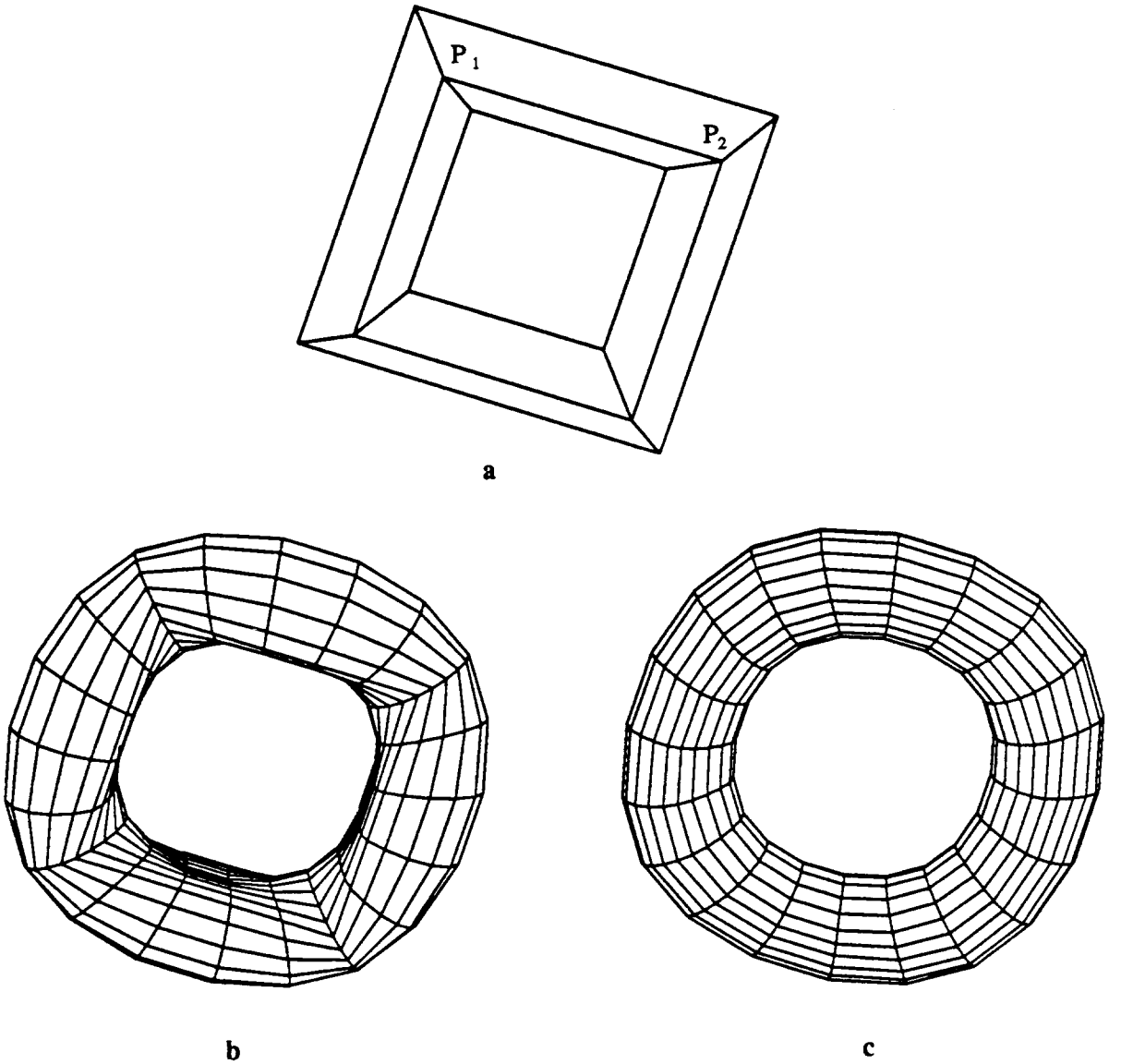


Figure 5.6: *Interpolation of a torus.* **a.** *Polyhedral mesh.*
b. *Surface produced with Projection method for tangent directions.*
c. *Surface produced with Opposite Edges method for tangent directions.*

5.4.3.1. Case of Quadrilateral Meshes

For the particular case of a toroidal frame, opposite tangent directions had to be 'smoothed' to generate a more realistic toroidal surface. This process of 'smoothing' opposite tangent directions can be easily applied to the meshes in which each vertex has exactly 4 neighbors.

Suppose that vertex P has 4 neighbors P_1, P_2, P_1^{op}, and P_2^{op} (Fig. 5.7a). Assuming that P_1 is topologically opposite to P_1^{op}, and P_2 to P_2^{op}, the tangent directions are determined simply by

constructing procedural curves (with Bisector normal) through vertices P_1, P, P_1^{op} and then through P_2, P, P_2^{op}. Thus, opposite boundary curves will join with G^1 continuity at P. The resulting curve mesh will consist of two sets of G^1 curves that are very similar to isoparametric curves on a predefined surface.

In this case there is actually no need to compute the normal, because it is implicitly defined by the cross product of the two tangent directions at the vertex. Furthermore, it is clear that this combined method of normal and tangent directions determination is guaranteed not to fail for any non-degenerate constellation of interpolation points. Testing of several asymmetrical meshes with concave angles and saddle points confirmed that the Opposite Edge method is robust even in these situations (Section 5.6.2).

Also, it is still possible to predefine normals at the vertices. In this case, the G^1 tangent directions could be just projected onto the tangent plane. Alternatively, the tangent directions could be forced to follow any change in the normal, so that the geometric constellation of the normal, the tangent plane and the tangent directions on that plane is rigid and can only change its orientation, but not the shape. This situation is very similar to the one in interpolating curves, where the tangent direction has to always follow the normal in order to be orthogonal to it. This approach is superior to projecting the tangent directions onto the tangent plane, because there are no restrictions in choosing the normal. For example, one can select a normal collinear with one of the G^1 tangent directions.

5.4.3.2. Case of Arbitrary Meshes

The opposite edge method can be easily extended to meshes with arbitrary topological structure. Two cases must be considered.

Case 1. The number of neighbors m of the current vertex P is even. In this case, for each neighbor vertex P_i, we identify its topological opposite P_i^{op} and construct a procedural curve through the points P_i, P, P_i^{op}. Therefore, there will be a total of $m/2$ G^1 boundary curves meeting at current vertex (Fig. 5.7a).

Case 2. The number of neighbors m of the current vertex P is odd. In this case, for each neighbor vertex P_i, we identify its topologically opposite sector $P_i^{op\,1}PP_i^{op\,2}$ and define its *virtual opposite* P_i^{op} to be the midpoint of $P_i^{op\,1}$ and $P_i^{op\,2}$. Then a procedural curve through P_i, P, P_i^{op} is constructed and its tangent is used to define the tangent direction of the edge P_iP (Fig. 5.7b).

In both above cases, several different tangent directions will be generated that need not lie in the same plane (unless $m = 4$). Therefore, a normal has to be defined at the current vertex. The tangent directions will then be simply projected onto the tangent plane. The normal can be defined by one of the normal methods from Section 5.3.1. However, a more natural choice for the normal should be based on the computed tangent directions. At the current vertex, the m tangent directions form a pyramid with m faces. The Opposite Edge normal $\mathbf{n}_{OE}$ is then computed simply as an arithmetic average of the m face normals of this pyramid. This approach also

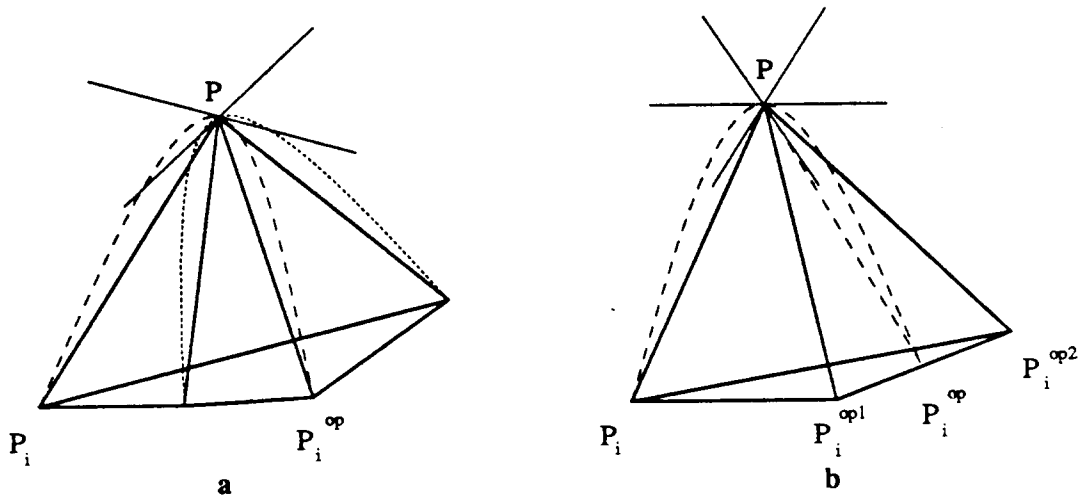


Figure 5.7: *Opposite Edge method.* **a.** *Even number of neighbors.* **b.** *Odd number of neighbors.*

agrees well with the case of exactly 4 neighbors.

Similarly to the Angle Average normal $\mathbf{n}_{AN}$, one could try to weight average face normals of the pyramid of the tangent directions by the angles between the adjacent directions. However, since the faces of the pyramid of the tangent direction are nearly coplanar, the difference between the resulting normal and $\mathbf{n}_{OE}$ can be expected to be quite small. Testing showed that this is indeed the case. Therefore, plain arithmetic averaging is preferable for efficiency reasons.

The Opposite Edge method is very robust in the sense that it can handle arbitrary constellations of points. This can be seen, for example, on a polyhedral model for a saddle surface. While edge-based and least squares normal methods fail for this surface altogether, the Opposite Edge method produces a surface with less curvature variation compared to all the other methods (Section 5.6.2).

5.5. Determination of Velocities

Computation of the actual position of Bézier points of each boundary curve is the last step in their construction. At this stage, the tangent directions of the boundaries at each vertex are known, and only their *velocities*, or derivative magnitudes, remain to be determined. Since the boundary curves are cubics, Bézier points will be located at distances of one third of these magnitudes from the corresponding vertices. Again, we will utilize the knowledge we gained from studying the analogous problem of velocity determination for interpolating curves (Section 3.3.3).

5.5.1. Generalizations of Curve Interpolation Schemes

Consider Fig. 5.8. In this figure, edges of the mesh adjacent to the vertex P are shown. The velocity v_i in the direction from P to P_i will always depend on the corresponding edge length l_i, but it may or may not depend on $l_j, j \neq i$. Thus, similarly to the univariate case, the velocity methods can be classified into 2 groups: ones that depend only on the underlying side of the mesh, and ones that look at some or all neighbors of the current vertex.

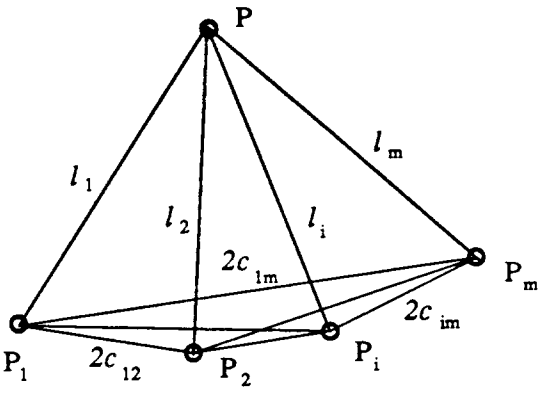


Figure 5.8: *Constructions for velocity determination.*

For both groups, velocity methods are rather straightforward generalizations of the corresponding approaches for interpolating curves (Section 3.3.3.2). Listed below are the extensions of some curve interpolation methods that produced reasonable results for the case of patch boundaries.

Edge Length Method. In this method, the velocity corresponding to a certain edge in the mesh is just made equal to the length of this edge:

$$v_{i,L} = l_i. \tag{5.12}$$

Similarly to Projection and Planar Boundary methods for tangent directions, this approach doesn't depend on other neighbor vertices and therefore is tessellation-independent.

Weighted Edge Length. A more sophisticated version of the above approach would take all the lengths l_j into account:

$$v_{i,WL} = \frac{\sum_{j=1}^{m} k_{ij}\, l_j}{\sum_{j=1}^{m} k_{ij}}. \tag{5.13}$$

Here k_{ij} are the weight factors. One possible way to define k_{ij} is to make all of them equal. A better choice would be to define $k_{ii} = 1$ and for $j \neq i$,

$$k_{ij} = \begin{cases} |\cos(P_i P P_j)| & \text{if } \cos(P_i P P_j) \le 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

Thus, v_i will be influenced only by the edges that form obtuse angles with PP_i. This would allow surfaces with strongly varying principal curvatures (i.e. a cylinder) to be constructed, because velocities in nearly perpendicular tangent directions will not affect each other.

Weighted Catmull–Rom and Edge Length Average. This method corresponds to the default velocity v_{CL} used for interpolating curves. It was computed as an average of the edge length and half chord length between the previous and the next vertex (Section 3.3.3.3). For surfaces, the natural extension of this method should utilize some measure of chord lengths c_{ij} between some vertex neighbors.

However, some of the c_{ij} seem to be rather irrelevant to the velocities at P. Rather than using chord length between vertex neighbors, a measure of 'stepping across' for each tangent direction is needed. This measure can be provided by the Opposite Edge method.

5.5.2. Opposite Edge Method

The idea of the Opposite Edge method for normal and tangent directions is to construct these geometric parameters from procedural curves (Section 5.4.3). These curves have topologically opposite vertices as their endpoints and pass through the current vertex. It is therefore natural to use this same curve information to define the last geometric parameter, the velocity as well. In other words, v_i is determined as follows:

1. Given P_i, identify the topologically opposite vertex P_i^{op} for even number of edges, or construct the virtual opposite vertex as the midpoint between the endpoints of the topologically opposite sector for odd number of edges.
2. Construct a procedural curve through P_i, P, and P_i^{op} (with the Bisector normal $\mathbf{n}_B$ and Catmull–Rom and Edge Length average velocities v_{CL}). Define v_i as the velocity at P_0 towards P_i.

Since the construction of procedural curves was required by the Opposite Edge method for the determination of tangent directions (Section 5.5.1), no additional computational effort is required.

5.6. Comparison of Methods for Geometric Parameters

We have tested various methods for determining normal, tangent directions, and velocities on a set of polyhedral meshes. This set included strictly quadrilateral meshes where each point has exactly 4 neighbors as well as arbitrary meshes with no restrictions. For some of the meshes, interpolation points were sampled from an a priori known surface, such as a sphere or a torus, and for the others, vertices were placed at random.

5.6.1. Testing Tools

As in the case of interpolating curves, we measured the quality of the resulting surface in terms of a minimization of certain curvature characteristics. Three types of curvature were measured: Gaussian curvature K, mean curvature H, and absolute curvature A. The latter is defined as the sum of the absolute values of principal curvatures k_1 and k_2 [43]:

$$A = |k_1| + |k_2|. \tag{5.15}$$

Absolute curvature will always recognize curvature in the surface even if $K = 0$ (cylinder) or $H = 0$ (hyperboloid) and may be the most reliable measure of the three.

For each boundary curve, the values at several sample points for all 3 types of curvature were computed on both sides of the boundary. As the resulting surface is not necessarily G^2, these values are generally different. Since it is relatively hard to compute the curvature of Gregory patches, a subdivision technique (Chapter 7) was used to represent the surface with Bézier patches. Furthermore, for each boundary curve, the integral $\int |\kappa_1 - \kappa_2| \, ds$ of curvature differences was computed. Here κ_1 and κ_2 are curvature values (Gaussian, mean, or absolute) on either side of the boundary. Four different measures were used in comparison tests:

1. Maximum curvature value on all boundaries.
2. Maximum difference between two curvature values on either side of a common boundary.
3. Maximum integral of the difference of curvature values.
4. The sum of these integrals for all boundaries.

The measurements were performed on many polyhedral models; however, here we present the results for only nine of them:

1. A triangular mesh for a sphere (Fig. 5.11a). This mesh was generated by sampling the sphere at uniformly spaced values of the (u,v) parameter space.
2. In the above mesh, the points were shifted by a random amount in the (u,v) space (small enough not to change the topological structure) and then resampled.
3. A quadrilateral mesh for a rotational solid in Fig. 5.12a. It was generated by rotating a sine wave around a line that does not intersect it.
4. Again, the mesh points were shifted by a random amount in the parameter space and then the surface was resampled.
5. A quadrilateral mesh for a torus (Fig. 5.6a).
6. A quadrilateral mesh for a surface, used by Franke [49] for comparison of scattered data interpolation methods (Fig. 5.9).
7. A random triangular mesh in Fig. 5.10.

8. A mesh where vertices were placed at random heights over a regular quadrilateral domain.

9. A saddle surface, formed by 7 vertices and 6 triangular faces (Fig. 5.11).

The results are listed in Tables 5.1, 5.2, and 5.3. The 4 numbers that appear in each square of the table correspond to the 4 different measures of curvature listed above. Although all 3 types of curvature were measured, we believe that it is sufficient to list only the absolute curvature A. Indeed, the variation of values for one type of curvature between different methods was very similar to another.

For the comparison of the effect of various approaches on a particular geometric parameter, the same methods were used for all other parameters. For comparison of the normal methods, the Opposite Edge method was used for the tangent directions, and the Edge Length method for the velocities. For comparison of the tangent direction methods, the Opposite Edge method was used for the normals and the Edge Length method for the velocities. Finally, the Opposite Edge method for both normals and tangent directions was used for comparison of the velocity methods.

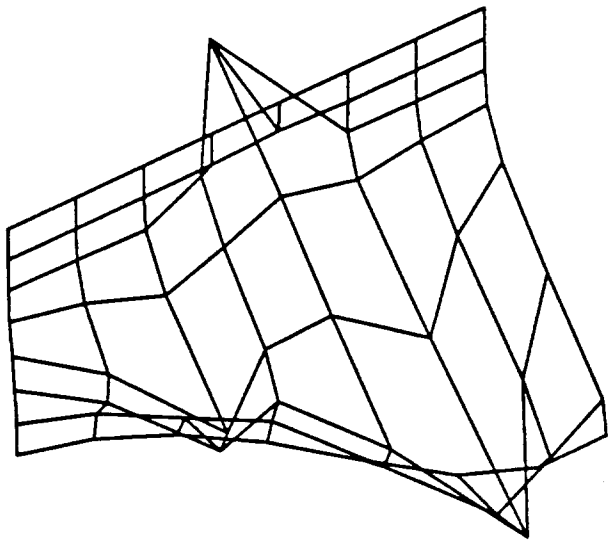


Figure 5.9: *A test quadrilateral mesh.*

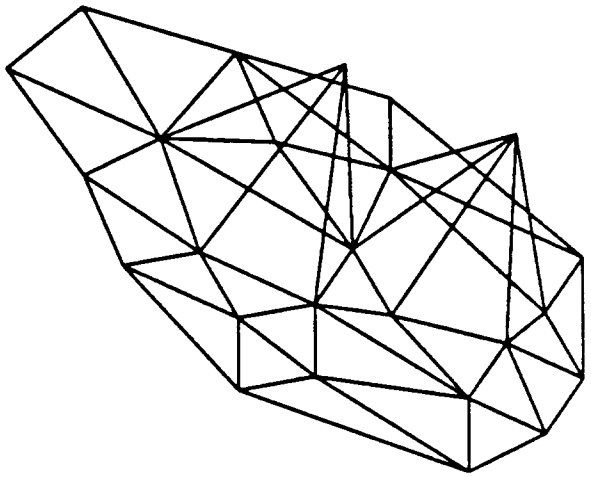


Figure 5.10: *A test triangular mesh.*

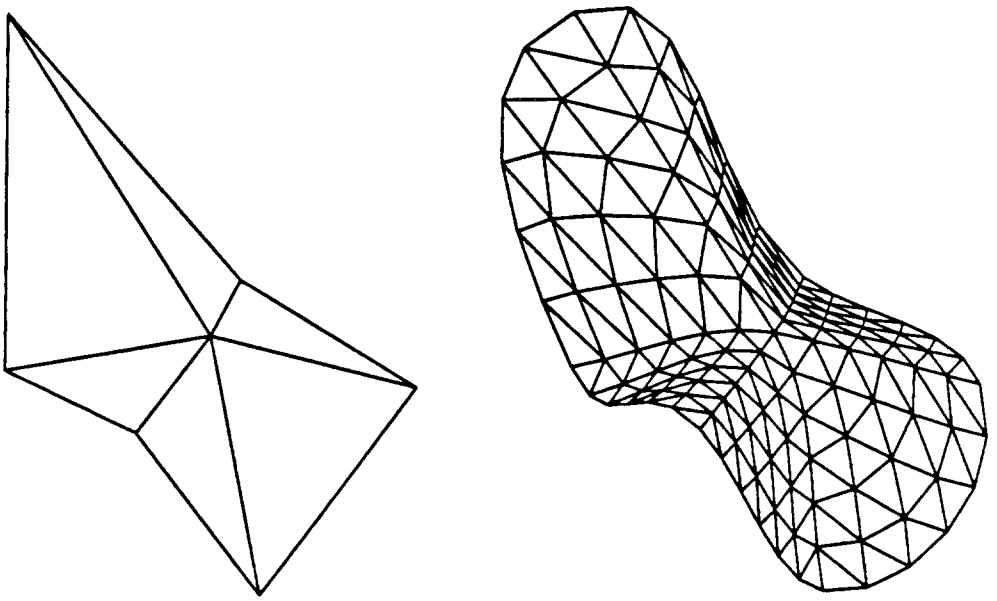


Figure 5.11: a. *A polyhedral model for a saddle surface.*
b. *The interpolating surface, obtained with the Opposite Edge method.*

Polyhedron	Normal Method				
	Face normal average n_{MN}	Angle weighted average n_{AN}	Face area weighted average n_{DN}	Inverse face area weighted average n_{IN}	Opposite Edge normal n_{OE}
1	5.7	5.4	5.5	5.8	5.8
	3.5	3.6	3.1	3.9	3.8
sphere	1.6	1.8	1.1	2.2	1.8
	46.8	73.1	42.2	59.6	65.8
2	14.8	15.3	20.6	19.7	12.6
resampled	9.8	11.4	17.2	12.7	10.6
sphere	7.5	5.8	13.4	8.0	7.8
	137.2	128.0	156.3	153.6	124.6
3	3.4	3.3	3.3	3.5	3.4
rotational	0.9	1.0	0.7	1.0	0.9
solid	1.8	2.3	1.4	2.2	1.9
	26.7	30.6	20.0	33.1	28.4
4	8.9	7.8	10.3	7.5	6.9
resampled	7.3	6.6	8.0	6.6	5.8
rotational	4.0	4.6	7.7	4.8	3.8
solid	73.1	75.9	81.1	75.3	66.4
5	1.8	2.7	1.8	1.8	1.8
	1.8	2.5	1.8	1.8	1.8
torus	16.5	28.7	16.5	16.5	16.5
	131.8	229.7	131.8	131.8	131.8
6	36.4	29.9	41.7	31.0	25.3
Franke	20.3	19.6	24.8	15.9	14.7
surface	4.9	4.7	6.5	3.5	4.2
	63.3	64.0	66.0	60.2	54.6
7	5.4	4.8	6.2	5.7	4.9
random	3.8	3.8	5.1	4.2	3.6
triangular	8.5	8.0	7.0	11.1	6.4
mesh	73.8	68.5	79.0	76.5	73.9
8	43.4	43.1	44.3	42.9	38.9
random	17.2	17.6	17.1	17.4	14.4
quadrilateral	1.5	1.5	1.7	1.5	1.6
mesh	44.7	44.0	46.1	44.5	42.5
9	2.0	2.1	2.7	2.3	2.3
saddle	1.1	1.2	1.9	1.0	1.2
surface	2.8	2.8	3.9	2.9	2.6
	8.3	8.4	12.2	7.6	8.0

Table 5.1: *Curvature comparison of methods for the normal determination. The Opposite Edge method was used for the tangent directions, and the Edge Length method for the velocities.*

Polyhedron	Tangent Direction Method		
	Projection method	Planar method	Opposite Edge method
1	3.1	3.0	5.8
	0.9	0.8	3.8
sphere	0.6	0.6	1.8
	23.7	23.6	65.8
2	41.4	33.9	12.6
resampled	36.8	28.7	10.6
	36.8	28.6	7.8
sphere	193.4	167.4	124.6
3	3.5	3.5	3.4
rotational	0.6	0.6	0.9
	0.9	0.9	1.9
solid	17.6	17.6	28.4
4	278.1	99.9	6.9
resampled	276.4	98.4	5.8
rotational	24.9	10.8	3.8
solid	129.8	82.6	66.4
5	4.0	4.0	1.8
	2.1	2.1	1.8
torus	29.8	29.8	16.5
	238.3	238.3	131.8
6	25.5	25.9	25.3
Franke	17.1	18.8	14.7
	5.9	6.3	4.2
surface	57.8	59.7	54.6
7	7.2	7.5	4.9
random	5.6	6.3	3.6
triangular	13.1	16.0	6.4
mesh	110.4	115.8	73.9
8	39.1	40.0	38.9
random	17.4	15.4	14.4
quadrilateral	2.0	2.6	1.6
mesh	52.4	48.7	42.5
9	8.3	2.4	2.3
saddle	7.2	2.0	1.2
	24.7	10.1	2.6
surface	48.3	23.8	8.0

Table 5.2: *Curvature comparison of methods for determination of the tangent directions. The Opposite Edge method was used for the normals, and the Edge Length method for the velocities.*

Polyhedron	Velocity Method		
	Edge Length	Weighted Edge Length	Opposite Edge
1	5.8	9.2	7.5
	3.8	7.2	5.2
sphere	1.8	2.7	2.2
	65.8	100.3	76.2
2	12.6	28.3	26.9
resampled	10.6	25.8	24.6
	7.8	10.0	11.1
sphere	124.6	177.2	178.0
3	3.4	4.2	4.1
rotational	0.9	1.2	1.2
	1.9	2.7	2.6
solid	28.4	39.4	38.7
4	6.9	8.4	8.8
resampled	5.8	7.1	7.5
rotational	3.8	4.7	4.8
solid	66.4	75.4	75.4
5	1.8	2.5	2.5
	1.8	2.3	2.3
torus	16.5	20.9	20.9
	131.8	167.0	167.0
6	25.3	38.1	38.1
Franke	14.7	13.6	14.0
	4.2	2.7	2.8
surface	54.6	54.8	54.7
7	4.9	7.0	7.2
random	3.6	5.0	5.1
triangular	6.4	10.0	10.0
mesh	73.9	96.4	96.5
8	38.9	58.3	55.6
random	14.4	19.3	20.7
quadrilateral	1.6	1.8	1.9
mesh	42.5	53.5	54.5
9	2.3	2.4	2.2
sadlle	1.2	1.4	1.3
	2.6	3.8	3.6
surface	8.0	8.8	9.4

Table 5.3: *Curvature comparison of methods for the velocities. The Opposite Edge method was used for the normals and the tangent directions.*

5.6.2. Evaluation of the Results

5.6.2.1. Normals and Tangent Directions

For the meshes of irregular topological structure (#7 in the list above) or for the meshes with scattered vertex locations (#2, #4, #6, #8), the Opposite Edge method for normals and tangent directions is generally preferable over other methods. The case of a 6-patch surface with a single saddle point (#9) also shows that the Opposite Edge method is better equipped to handle saddle surfaces. We also observe that the difference in curvature variation between the normal methods is not as significant as the difference between the methods for tangent directions. This once again confirms the importance of having boundary curves join smoothly at the vertices.

In some cases, it was actually necessary to default to the Opposite Edge method for tangent directions at several vertices when other methods were tested because no G^1 surface could be produced. This situation usually arose when the normal method used was different from the Opposite Edge method. However, if the input mesh was sampled from some known surface (#1, #3), the results were not so conclusive, and in several cases the Projection method produced better results in terms of curvature minimization. There are two possible reasons for that.

First, the Opposite Edge method may not work very well if the input mesh does not reflect the symmetry of the underlying surface. For example, consider a sphere with a natural mesh structure of parallels and meridians. Suppose that only triangular faces are allowed in the mesh; then each of quadrilateral faces has to be split diagonally (Fig. 5.12a). Since there are 5 edges meeting at vertices that are the neighbors of a pole, application of the Opposite Edge method will slightly distort the directions of the meridians there (Fig. 5.12b).

The second reason is the fact that our curvature tests serve not as the absolute criteria of the quality of the surface, but only as one possible indicator. Consider again an example of a sphere or a general surface of revolution with a quadrilateral mesh of parallels and meridians. The Opposite Edge method in this case will reproduce parallels and meridians as boundaries of the surface; that's what one would expect of a good method in this case (Fig. 5.13b). The Projection method will also reproduce the meridians, but the boundary curves in the other direction will be slanted (Fig. 5.13c). In this case, worse results in terms of curvature minimization for the Opposite Edge method can be explained by the fact that G^1 continuous parallels in areas of rapidly varying radius (i.e., near poles of the sphere) have larger curvatures than the more direct connections of, say, the Projection Method. Therefore, the difference of curvatures of interpolating polynomial patches at these boundaries can also be expected to be high (see Tables 5.1 and 5.2 for polyhedrons #1 and #3).

However, if the interpolation points are shifted by a random amount in the parameter space of the surface and then resampled, the Opposite Edge method gives again better results (compare meshes #1 to #2 and #3 to #4 from Tables 5.1 and 5.2). This shows once more that the Opposite Edge method is more robust for irregular situations.

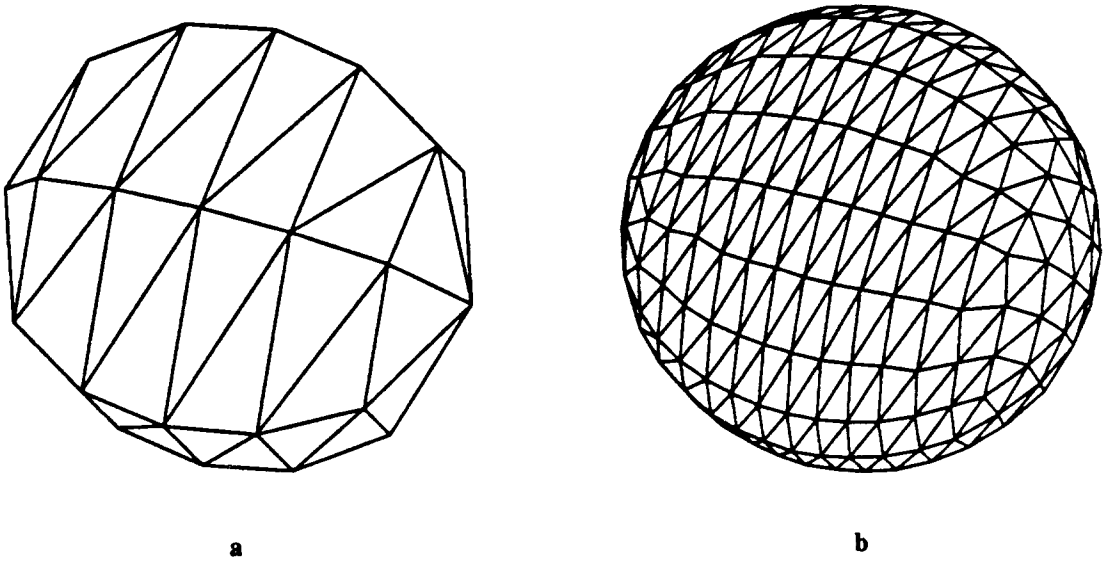


Figure 5.12: a. *A triangular mesh for a sphere.*
b. *Surface, resulting from application of Opposite Edge method. Meridians and parallels are distorted near poles.*

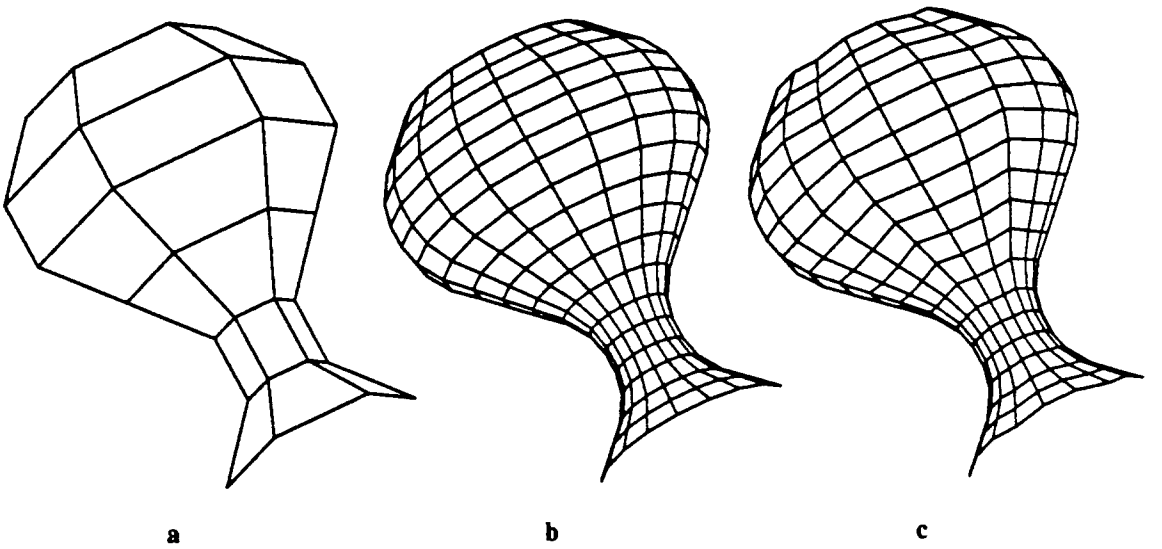


Figure 5.13: a. *A mesh for a surface of revolution.*
b. *Surface produced by the Opposite Edge method.*
c. *Surface produced by the Projection method.*

From the above evaluation, we draw the following conclusions. If the interpolation points come from some experiment and are not expected to lie on some a priori known surface, then the Opposite Edge method is clearly the one to be used. The Opposite Edge method also produces very good results if the input mesh is in some sense symmetric (for example, points sampled in a regular way from a surface of revolution).

On the other hand, if the surface to be reproduced is known to possess certain symmetries, and the input mesh *does not* have these symmetries, Projection or Planar Boundary methods may be preferable. None of these methods, though, will be able to reproduce a symmetric surface, because they only work with the mesh and have no knowledge of the underlying symmetries. However, if the shape of the desired surface is known, a designer will most likely know what the boundary curves should look like. He then can directly feed this information to the system.

So, from practical point of view, one should use the Opposite Edge method for irregular meshes. The Opposite Edge method also produces good results if the vertices lie on a predefined surface and the mesh reflects the symmetries (if any) of the interpolation points. Usually, however, for regular meshes generated from an a priori known surface, boundary curves are also known. In this case, the step of constructing them may be omitted altogether.

5.6.2.2. Evaluation of Velocity Methods

We have utilized the curvature comparison tools used for normal and tangent directions evaluations. For all test cases, the Edge Length velocity v_L produced the best results in terms of curvature minimization. This result is somewhat surprising, because for interpolating curves, an average of Catmull–Rom and Edge Length v_{CL} is superior. However, for surfaces, interpolation points are usually spaced more regularly than they are for curves in our test examples in Chapter 3. Thus, situations with greatly varying adjacent edge lengths and acute angles for which velocity v_{CL} gives better results arise quite seldomly. Simplicity is an additional obvious advantage of the Edge Length method.

5.7. Procedural Rules

In this Section, we extend the notion of procedural rules used for curve interpolation to the case of surfaces. The basic idea remains the same: a set of *patterns* of the defining mesh is introduced that triggers the application of certain *rules*. These rules may depend not only on the immediate vertex neighbors, but also on its second neighbors, etc. A *threshold* is usually associated with each rule that determines how close the actual mesh is to the pattern. Please refer to Section 3.4.1 for more details.

5.7.1. Coplanar Faces

This situation corresponds to the case of collinear segments for curves (Section 3.4.2). If several adjacent faces in the mesh are coplanar, this may signal that all the vertices in these faces should have the same normal (Fig. 5.14). A similar situation may arise, for example, at the boundary curve between a circular cylinder and a hemisphere of the same radii. In this case, normals at the boundary should preferably coincide with those of the cylinder.

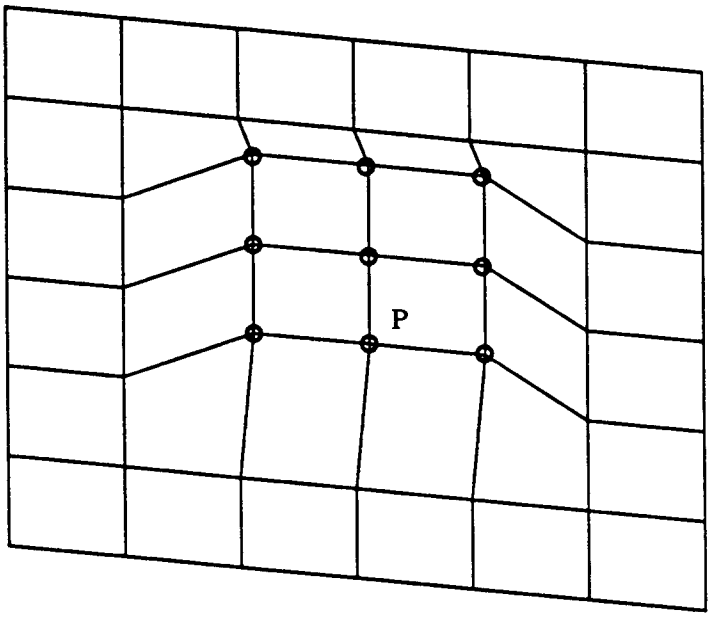


Figure 5.14: *The case of coplanar faces. Marked vertices should have the same normal* **n**.

Suppose that the normal at vertex P need to be determined (Fig. 5.14). We represent it as follows in analogy with Section 3.4.2:

$$\mathbf{n} = (1 - \max(k_i))\,\hat{\mathbf{n}}_D + \sum k_i\,\hat{\mathbf{n}}_i . \tag{5.16}$$

Here $\hat{\mathbf{n}}_D$ the default normal that is computed without application of any special rules, $\hat{\mathbf{n}}_i$ are face normals of the faces that contain P, and k_i are weight factors, defined as follows. For each face, containing P, we find the maximum dihedral angle β_i between this face and its adjacent faces. Then k_i are defined as follows:

$$k_i = \begin{cases} 0 & \text{if } \cos(\beta_i) > t \\ (t - \cos(\beta_i)) / (1 + t) & \text{if } -1 \le \cos(\beta_i) \le t \end{cases} \tag{5.17}$$

Thus, if none of the dihedral angles is close to 180° ($\cos(\beta_i) > t$), then the normal is defined by the default method, i.e. $\mathbf{n} = \hat{\mathbf{n}}_D$. Otherwise, the closer β_i is to 180° (and $\cos(\beta_i)$ to −1), the closer the corresponding k_i will be to 1, and the bigger the influence of the corresponding $\hat{\mathbf{n}}_i$ will be on the final normal. In the limiting case, k_i will be 1, if the current face has at least one adjacent

coplanar face, in which case the default normal $\hat{\mathbf{n}}_D$ will have no influence. The threshold t is needed for stability of the approach and can be set, for example, to 0.95 (cf. Section 3.4.2).

The surface, resulting from the application of this rule to the mesh in Fig. 5.14, is shown in Fig. 5.15.

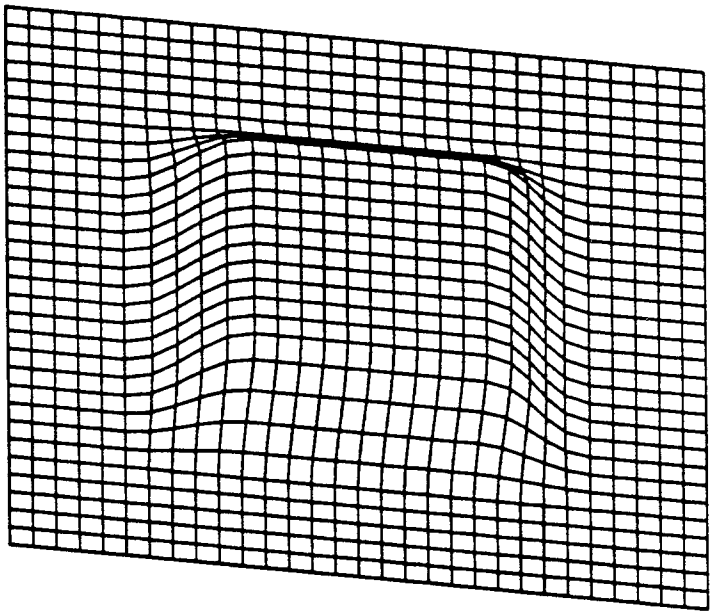


Figure 5.15: *Surface, constructed with an application of the Coplanar Faces rule to the mesh from the previous Figure.*

5.7.2. Disproportionate Areas

As mentioned earlier, it may be desirable to assign greater weights to faces of smaller areas because they may convey strong information about the normal direction. One possible way to achieve this goal is to use Inversely Weighted normals $\mathbf{n}_{IN}$ (Section 5.3.1) as the default rule. An alternative approach would be to form a weighted average between the default normal $\hat{\mathbf{n}}_D$ and the normals $\hat{\mathbf{n}}_i$ of the smaller face.

Suppose that m faces F_i with normals $\hat{\mathbf{n}}_i$ meet at a certain interpolation point. For each face F_i, we compute the ratio r_i of the largest area of all other faces to the area of F_i. Then the final normal $\mathbf{n}$ is determined as a weighted average:

$$\mathbf{n} = \hat{\mathbf{n}}_D + \sum_{r_i > R} \left(\frac{r_i}{R} - 1\right) \hat{\mathbf{n}}_i . \tag{5.18}$$

Here R is a threshold that can be set, for example, to 10.

As an example, consider Fig. 5.16. In this figure, the area of the face $P_1P_2P_3$ is small compared to the areas of adjacent faces. Therefore, normals at P_1, P_2, and P_3 should be close to the normal of the face $P_1P_2P_3$.

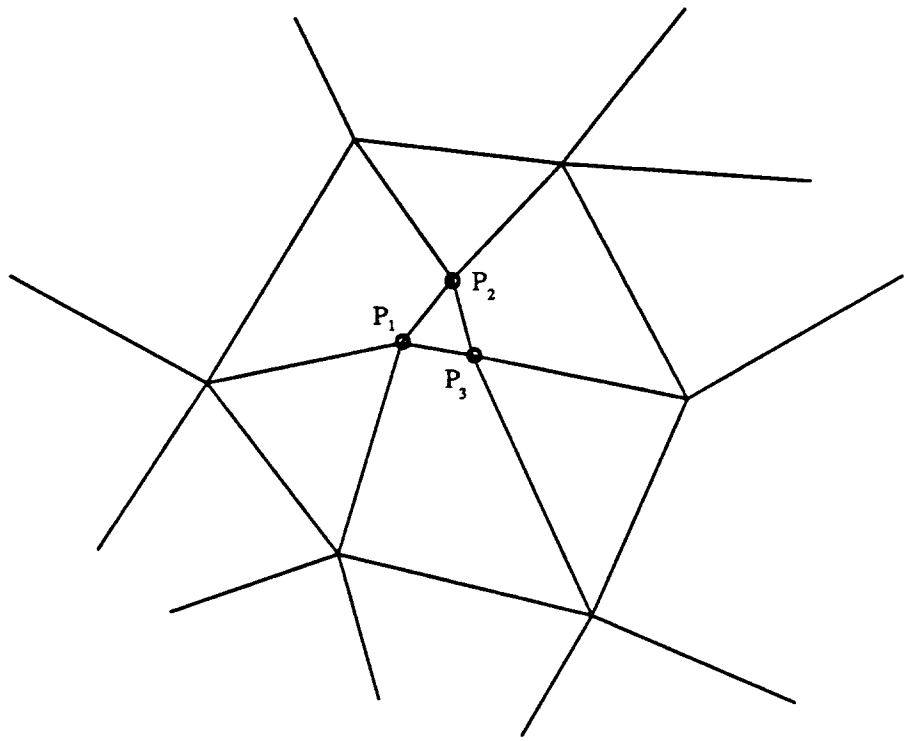


Figure 5.16: *The case of disproportionate areas.*
Normals at P_1, P_2, P_3 should be close to the normal of the face $P_1P_2P_3$.

5.7.3. Rules for Boundary Points

Similarly to the endpoints of a defining polygon for curves, the boundary points of a defining mesh for surface have to be treated differently. We now describe several possible rules for determination of normals, tangent directions, and velocities at the boundaries of surfaces.

5.7.3.1. Normal Construction at Boundary Points

Face Average. The simplest possible rule for normal construction at a boundary is to have no special rules at all. Indeed, it is possible just to utilize the usual face averaging at boundary points as well (Section 5.3.1). However, the resulting surfaces will usually change sign of Gaussian curvature near the boundary which may be undesirable. This effect is similar to an extra inflection in the curve near its endpoint if the Perpendicular method for normals is used (Section 3.4.5.1).

Symmetric Method. To avoid inflections, normals of the neighbor vertices of boundary points can be used to define the required normal. Two cases must be considered. In the first case, the current boundary vertex P_0 has at least one neighbor that does not lie on the boundary (Fig. 5.17a). Suppose that P_i are P_0's neighbors that lie inside the defining mesh. Then the normals at P_i can be defined in a usual manner, call them $\hat{\mathbf{n}}_i$. For each $\hat{\mathbf{n}}_i$, we find its symmetric image $\hat{\mathbf{n}}_i^s$ with respect to the plane passing through the midpoint of P_0P_i and perpendicular to it. The required normal at P_0 is then computed as an arithmetic average of all $\hat{\mathbf{n}}_i^s$.

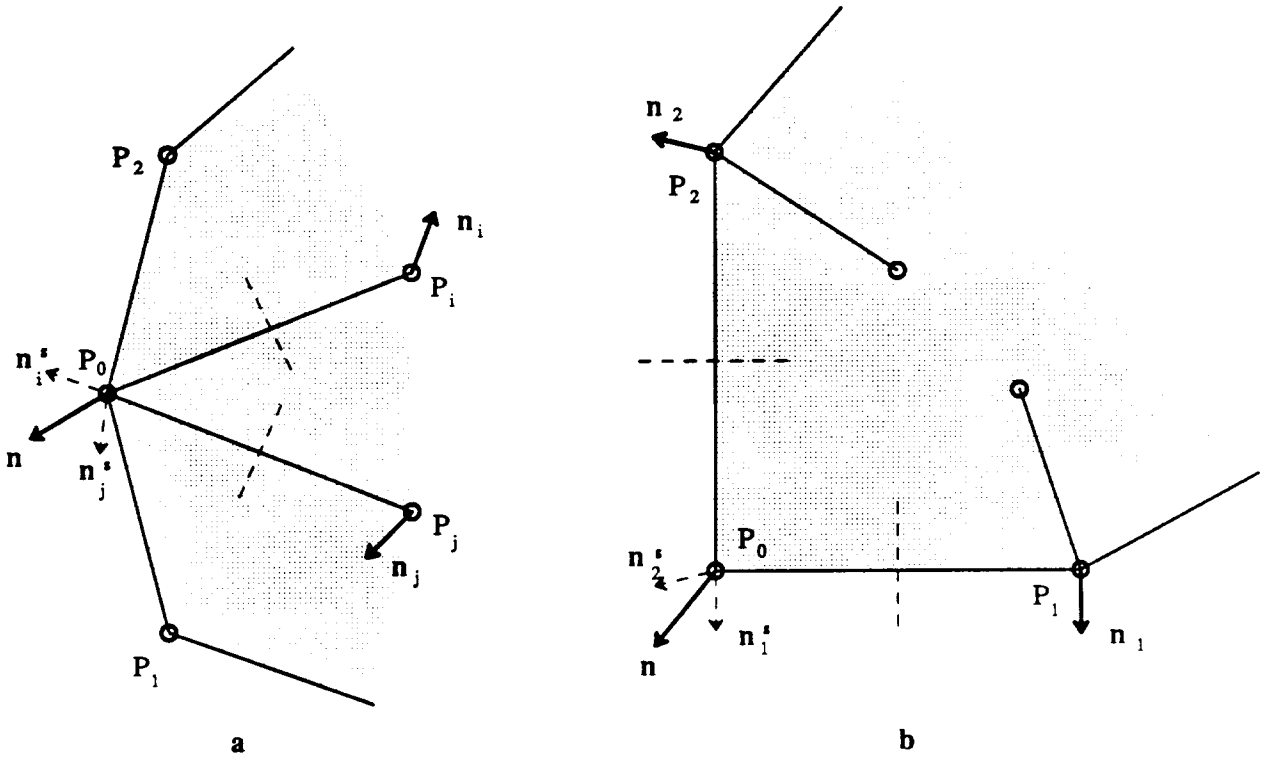


Figure 5.17: *Normal determination at boundary points.*
a. *Non-corner point.* **b.** *Corner point.*

In the second case, the only two neighbors P_1 and P_2 of the current vertex P_0 are also boundary points. In this case, P_0 can be regarded as a *corner point* (Fig. 5.17b). The normals $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ of P_0's neighbors can be computed as in the previous case. Finally, the normal $\mathbf{n}_0$ at P_0 is computed as an average of symmetric images of $\hat{\mathbf{n}}_1^s$ and $\hat{\mathbf{n}}_2^s$ with respect to the midpoints of the corresponding edges P_0P_1 and P_0P_2.

Different procedures for boundary and corner points need to be used to avoid the ordering problem. Indeed, the normals and tangent directions are first computed for the interior points of the mesh. Then this information is used for determination of geometric parameters at the boundary non-corner points. Finally, the normals and the tangent directions at the corner points are

computed using already computed geometric parameters at the other boundary points.

An additional advantage of this method is the fact that proper normals are generated for symmetric meshes. For example, if the defining mesh is an octahedron or an icosahedron with one or several faces removed, then the normals at boundary points, as well as the normals at interior points, will coincide with the normals to the enclosing sphere.

5.7.3.2. Construction of Tangent Directions at Boundary Points

Projection and Planar Boundary Methods. Clearly, these methods require no special treatment at the boundary points.

Symmetric Method for Opposite Edges. The idea of the Symmetric method for normals at boundary points can also be used to define tangent directions. Again, we consider two cases that correspond to the current vertex P_0 being a non-corner or a corner point. In the first case (Fig. 5.18a), the tangent directions at P_0 towards inner points P_i are determined by computing symmetric images t_i^s of the corresponding directions t_i at P_i. The symmetry plane again passes through the midpoint of P_0P_i and is perpendicular to it. Finally, the directions at P_0 towards its two neighbor boundary points P_1 and P_2 are taken from the procedural curve through $P_1P_0P_2$.

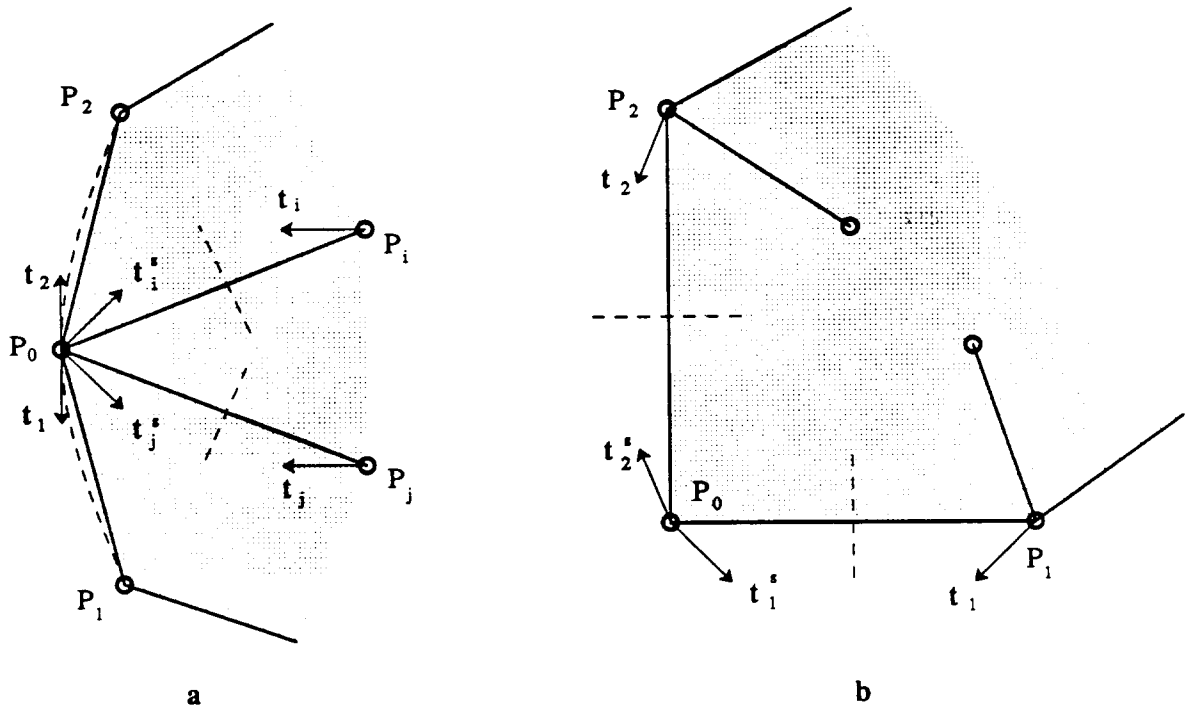


Figure 5.18: *Determination of tangent directions at boundary points.*
a. *Non-corner point.* **b.** *Corner point.*

In the case of the corner point (Fig. 5.18b), the tangent directions at P_0 are just symmetric images of the corresponding directions at P_1 and P_2.

5.7.3.3. Velocity Determination at Boundary Points

Since the Edge Length method (Section 5.5.1) is used for velocity determination, no special rules are required for the points on the mesh boundary.

5.8. Summary

In this Chapter, we have described the first step of building a smooth surface from an initial polyhedral mesh: the construction of boundary curves. Each edge of the polyhedral mesh is replaced by a cubic Bézier curve in such a way that all the curves sharing a certain vertex have a common tangent plane. It is then possible to define G^1 continuous patches with the constructed curves as their boundaries (Chapter 6). The union of these patched defines the final interpolating surface.

We use the same approach that we have used for the construction of interpolating curves (Chapter 3). In this approach, construction proceeds in separate procedural stages. At each stage, a certain geometric parameter associated with a curve is determined. There are 3 such parameters: *vertex normals*, *tangent directions*, and *velocities*.

At each procedural step, *default* values are assigned to the geometric parameters. These values are usually determined from a geometric, rather than an algebraic, point of view. The geometric reasoning generally produces surfaces that are close to what a designer expects. However, default values can always be manually overridden to suit application's particular need.

Besides determination of geometric parameters, our procedure may invoke *special rules*, associated with the shape of the defining polyhedral mesh. An application of a special rules usually improves the shape of the resulting surface.

The procedural approach is very robust in the sense that surfaces of good visual quality are produced even for irregular data sets. Moreover, the approach is very flexible in the sense that several geometric parameters may be preset by the user. The system then determines the remaining ones automatically.

6

Filling in Patches with Shape Parameters

In this chapter, a procedure for filling quadrilateral and/or triangular Gregory patches in a mesh of cubic Bézier curves is described. Along any boundary curve the shape of the two adjoining patches can be locally controlled with *shape parameters*, such as *tilt*, *bulge* and *shear*. The procedure is a simple extension of Chiyokura's method [27] that gives extra control to the user while preserving first order geometric continuity between the patches.

6.1. Introduction

In this Chapter, we address the problem of fitting surface patches into a network of cubic Bézier boundary curves, produced by the procedural method, discussed in Chapter 5. All the curves meeting at a network joint share a common tangent plane. This condition is necessary for constructing a G^1 interpolating surface. The curve mesh is otherwise unrestricted.

Many different approaches to interpolating a curve network have been published. However, the majority of them can not deal with completely unrestricted meshes; the curves have to satisfy some additional requirements. One of the pioneering approaches [41] describes a way to fit Bézier patches into a network of cubic curves, provided that their control points satisfy an *area ratio constraint* (Section 6.2.2). In Sarraga's method [115] quadrilateral Bézier patches of high order (up to 6) are blended into meshes of unrestricted cubic Bézier curves. However, no more than 5 curves may meet in an interior network point. A similar approach is discussed in [111]. In a recent development, Peters [100] published a method of interpolating with one polynomial piece per facet, provided that each interpolation point satisfies a *vertex enclosure constraint*.

The methods that do interpolate unrestricted meshes are admittedly more complicated. The split-domain approaches use several subpatches per facet; the subpatches also have to be G^1 across internal boundaries [39, 73, 100]. Alternatively, quadrilateral and triangular interpolants that typically require transversal derivative data on the boundaries have been proposed [61, 64, 67, 93]. A classification by Peters [101] compares various methods and resulting surfaces.

As already mentioned in previous Chapters, the ability to interpolate unrestricted boundary curves is very important for the flexibility of an interactive design system. In the following Sections, we look in greater details at some of the interpolation methods, and conclude that extension of Chiyokura's interpolation with Gregory patches is most suitable for our needs. Furthermore, the available degrees of freedom along boundary curves allows us to modify cross-boundary derivative vectors independently of each other; this adds more flexibility to our design system.

6.2. Conditions for G^1 Continuity between Neighbor Patches

6.2.1. General Formulation

A lot of research is being conducted now on the general geometric continuity conditions between two adjacent patches. Recent publications describe G^1 conditions between polynomial and rational patches of an arbitrary order [33, 82] and even conditions for G^2 continuity [35]. However, in this Chapter, we will only need to use G^1 continuity conditions between bicubic quadrilateral and/or quartic triangular Bézier or Gregory patches. We now proceed to describe them.

Suppose that two patches, Φ and Ψ, meet along the boundary Γ (Fig. 6.1). Then, for any point v on Γ, we can find $D\Phi(v)$, a cross-boundary derivative of Φ at $\Gamma(v)$, and $D\Psi(v)$, a cross-bound derivative of Ψ at the same point. Let $D\Gamma(v)$ be the tangent vector at $\Gamma(v)$. Then the necessary and sufficient conditions for patch continuity is coplanarity of all three vectors:

$$\det\left(D\Phi(v), D\Psi(v), D\Gamma(v)\right) = 0, \quad v \in [0,1]. \tag{6.1}$$

Patches Φ and Ψ can either be bicubic quadrilateral or quartic triangular Bézier or Gregory patches. Consider the case when the quadrilateral and the triangular Gregory patches join across the common cubic boundary. For the quadrilateral patch, $D\Phi(v)$ is just the usual partial derivative $\Phi_u(0,v)$. For the triangular patch, $D\Psi(v)$ is the radial derivative (2.44).

Both bicubic quadrilateral and quartic triangular patches have two internal control points along each edge, and the same procedure for finding these control points can be used.

The common cubic boundary of the two patches is used as a cubic for the evaluation of the bicubic quadrilateral patch, but has to be degree-elevated to the fourth degree for the evaluation of the quartic triangular patch.

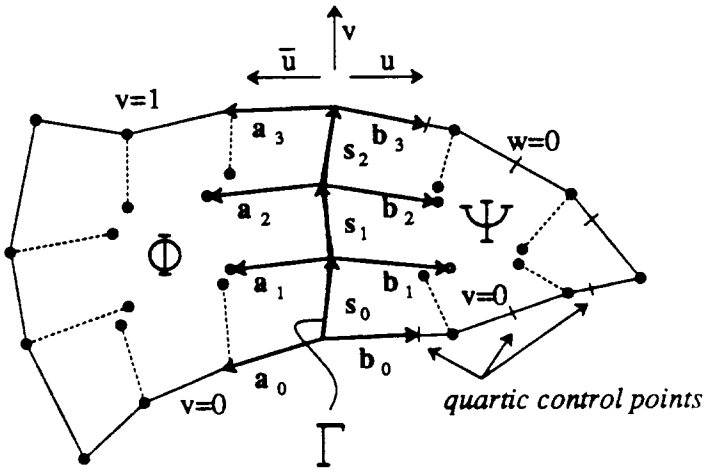


Figure 6.1: *A pair of patches, joining across a common boundary.*

$D\Phi(v)$ and $D\Psi(v)$ are polynomials of third degree in v, while $D\Gamma(v)$ is a polynomial of second degree. They can be expressed in terms of Bernstein polynomials and control points:

$$D\Gamma(v) = 3\sum_{i=0}^{2} \mathbf{s}_i B_i^2(v), \quad D\Phi(v) = 3\sum_{i=0}^{3} \mathbf{a}_i B_i^3(v), \quad D\Psi(v) = 3\sum_{i=0}^{3} \mathbf{b}_i B_i^3(v). \qquad (6.2)$$

The above expressions for cross-boundary derivatives are valid for bicubic quadrilateral patches only; for quartic triangular patches, the coefficient 3 is changed to 4 [38]. Also, for triangular patches, the vectors $\mathbf{t}_0$ and $\mathbf{t}_3$ point to the quartic points of a degree-elevated cubic boundary.

If we evaluate the determinant equation for $v = 0$ and for $v = 1$, that is, for the endpoints of the boundary curve, we can see that the vectors $\mathbf{a}_0$, $\mathbf{b}_0$, $\mathbf{s}_0$ must be coplanar; analogously, vectors $\mathbf{a}_3$, $\mathbf{b}_3$, $\mathbf{s}_2$ must also lie in the same plane. These conditions are called *endpoint conditions* for the boundary curve or *vertex planarity conditions* for the two neighbor patches.

The determinant equation is a polynomial of degree 8 in v, which yields 9 equations for the coefficients of the various degrees of v. Two of these equations represent endpoint conditions. The remaining 7 equations for the 4 unknown vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ proved to be very complicated and impractical to solve [84].

The fact that there are 7 constraints across each boundary (and, therefore, 5 degrees of freedom) implies that it is impossible to fit quartic triangular and/or bicubic quadrilateral Bézier patches into already defined cubic boundaries. Indeed, for triangular patches, there 3 inside control points, or 9 degrees of freedom; however, each boundary contributes on the average 7/2 = 3.5 constraints, which gives a deficit of 1.5 unfulfilled constraints for each triangle, and, analogously, of 2 unfulfilled constraints for each quadrilateral.

There are several possible way to deal with this lack of degrees of freedom. First, one can raise the degree of the patches. Generally, this approach leads to solving linear systems of equations for each interpolation point. These systems depend on the number of neighbors of the current vertex and may have non-unique solutions [106, 115]. Furthermore, this approach is somewhat complicated by the fact that vertices having even and odd number of vertices have to be treated differently. Indeed, a certain matrix is invertible at odd-points, but rank-deficient at even-points [100].

Another possible approach is to use several subpatches to fill each facet. One of the pioneering methods of this type has been suggested in [41]. In this method, quartic Bézier triangles in a triangular mesh are constructed. Then these triangles are subdivided into 3 subpatches that are adjusted to meet smoothly along their boundaries. A similar approach is discussed in [73] and generalized in [99]. However, the latter method is rather complicated and could also lead to solutions of large systems of equations. In Chapter 7, we will present a very simple closed-form solution to represent quadrilateral facets with 5 bicubic Bézier patches, and triangular facets with 3 quartic Bézier patches.

Finally, Gregory patches can be used to define the surface. Unlike Bézier patches, Gregory patches have no twist constraints (Section 2.3.4), and therefore have twice as many interior control points. These control points can be readily derived from boundary curve information.

We now look at several methods of positioning the interior control points of adjacent patches so that w can guarantee G^1 continuity between them.

6.2.2. Farin's Method

Rather than trying to solve the above determinant equation, one can premultiply each derivative by an unknown polynomial, and then equate the resulting linear combination to zero:

$$\alpha(v)\, D\Phi(v) + \mu(v)\, D\Psi(v) + \lambda(v)\, D\Gamma(v) = 0. \tag{6.3}$$

Since $D\Phi(v)$ and $D\Psi(v)$ are of degree 3, while $D\Gamma(v)$ is of degree 2, it is clear that

$$\mathbf{deg}\,(\alpha(v)) = \mathbf{deg}\,(\mu(v)) = \mathbf{deg}\,(\lambda(v)) - 1. \tag{6.4}$$

In this case, a system of *vector* equations, i.e. three identical scalar systems, one for each coordinate, will have to be solved. This is simpler than solving the determinant equation which mixes the components along the different axes. A discussion of various degrees that polynomials $\alpha(v)$, $\mu(v)$, and $\lambda(v)$ can be assigned, is given by Peters [100]. The simplest solution, however, is to premultiply $D\Phi$ and $D\Psi$ by a constant, and $D\Gamma$ by a linear polynomial.

This is the approach, taken by Farin [41]. Since one of the coefficients can always be set to 1, this leaves 3 coefficients to be determined. However, planarity conditions on either end of the boundary give 4 constraints. This means that the original control points of the cubic boundaries have to satisfy an additional constraint. Farin formulates this as a constraint on the areas of the triangles in Fig. 6.2 ($n = 3$):

$$\frac{\mathbf{area}(R_0S_0S_1)}{\mathbf{area}(S_0T_0S_1)} = \frac{\mathbf{area}(R_{n-1}S_{n-1}S_n)}{\mathbf{area}(S_nS_{n-1}T_n)}. \tag{6.5}$$

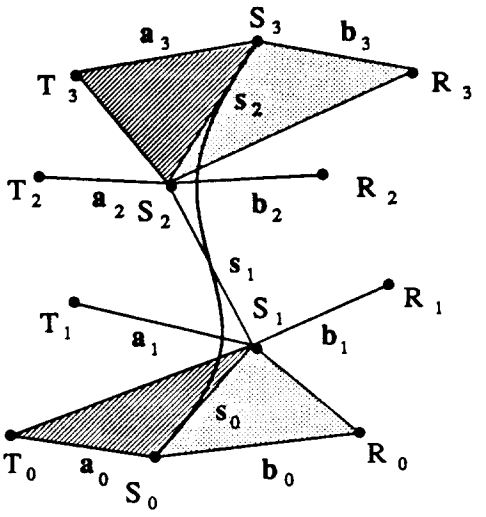


Figure 6.2: *The ratios of the areas of darkly and lightly shaded triangles must be equal.*

Since T_0,S_0,S_1 and R_0 are in the same plane, T_0 can be expressed as follows in barycentric coordinates:

$$\mathbf{T}_0 = \alpha_1 \mathbf{S}_0 + \alpha_2 \mathbf{S}_1 + \alpha \mathbf{R}_0, \quad \alpha_1 + \alpha_2 + \alpha = 1. \tag{6.6}$$

Analogously,

$$\mathbf{T}_n = \alpha_3 \mathbf{S}_{n-1} + \alpha_4 \mathbf{S}_n + \alpha \mathbf{R}_n, \quad \alpha_3 + \alpha_4 + \alpha = 1. \tag{6.7}$$

Then the geometric continuity is guaranteed by

$$\mathbf{T}_i = \frac{n-i}{n}(\alpha_1 \mathbf{S}_i + \alpha_2 \mathbf{S}_{i+1} + \alpha \mathbf{R}_i) + \frac{i}{n}(\alpha_3 \mathbf{S}_{i-1} + \alpha_4 \mathbf{S}_i + \alpha \mathbf{R}_i), \; i = 1,\ldots,n-1. \tag{6.8}$$

Note that $\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4$. This condition is the algebraic equivalent of the area ratio constraint.

Thus, Farin's method allows the computation of interior Bézier control points on either side of the boundary, provided that the boundary control points satisfy the area ratio constraint. Therefore, the cubic boundaries cannot be chosen completely freely, which is the main drawback of the method.

6.2.3. Premultiplication with Higher Order Polynomials

It is possible to raise the degree of the polynomials $\alpha(v)$, $\mu(v)$, and $\lambda(v)$ and premultiply $D\Phi$ and $D\Psi$ by linear, and $D\Gamma$ by quadratic polynomials. Now there are 4 constraints at either end and 7 coefficients of polynomials, one of which can be set to 1. This leaves 2 coefficients to

be chosen freely. Our vector polynomial is of degree 4, so one can expect 5 equations for the various degrees of v. Two of these equations, however, are fulfilled automatically at the ends of the boundary, leaving just three vector equations. Thus, if R_1 (Fig. 6.2) is set, and the 2 extra coefficients are chosen (note that again there are 5 degrees of freedom), then R_2, T_1 and T_2 can be determined. Thus one can find all four control points with just one computational pass across each boundary. However, it's not clear how to choose R_1 to ensure a 'symmetric' choice of control points.

We can go even further and premultiply the derivatives by two quadratic and one cubic polynomial, giving a total of 10 coefficients. One of them again can be set to 1, 4 will be determined from planarity conditions, leaving a free choice of the remaining 5 coefficients. Now there are three more free constants than in the approach described in the previous paragraph. However, there is a very important advantage: our polynomial is now of degree 5, two conditions at the ends are satisfied, and so there are *four* vector equations for *four* points R_1, R_2, T_1, T_2! No guesswork is needed here in terms of control points and there is some hope for a 'symmetric' solution, as soon as the 5 coefficients, representing all the degrees of freedom that exist, are picked.

We have succeeded in obtaining a closed-form solution in terms of the unknown coefficients. However, the resulting equations are too cumbersome for practical use, and it is not yet clear how to choose these unknown coefficients. A general discussion of this general approach with derivation of bounds on the degrees of premultiplying factors is given in [100].

6.2.4. Chiyokura's Method

The main idea of Chiyokura's method [27] is to construct two special C^1 continuous *basis* patches, Ψ' and Φ', on either side of the seam and then to determine internal control points for the real patches so that each is G^1 continuous with the basis patch on the other side of the seam. Thus, in our example, the real patch Φ will be fit to the basis patch Ψ' and the real patch Ψ will be fit to the basis patch Φ'. Because of the transitivity of geometric continuity, the two real patches are guaranteed to meet with G^1 continuity as well (Fig. 6.3).

To avoid Farin's area ratio constraint, Chiyokura restricts the basis patches to have only *quadratic* cross-boundary derivatives. This makes it easier to solve the determinant equation for each pair of a real and a basis patch, and the degree of this equation will be only 7, leading to exactly 6 constraints in the determination of the two interior control points of the real patch.

Consider the case of joining the real patch Φ and a basis patch Ψ' (Fig. 6.4). G^1 continuity would be guaranteed if $D\,\Phi$ can be expressed as a linear combination of $D\,\Psi'$ and $D\,\Gamma$, multiplied by some linear scalar functions in v:

$$\sum_{i=0}^{3} \mathbf{a}_i\, B_i^3(v) = (k_0(1-v)+k_1 v) \sum_{i=0}^{2} \mathbf{b}'_i\, B_i^2(v) + (h_0(1-v)+h_1 v) \sum_{i=0}^{2} \mathbf{s}_i\, B_i^2(v). \qquad (6.9)$$

The coefficients of this equation for $(1-v)^3$ and v^3 result in the planarity constraints at the mesh

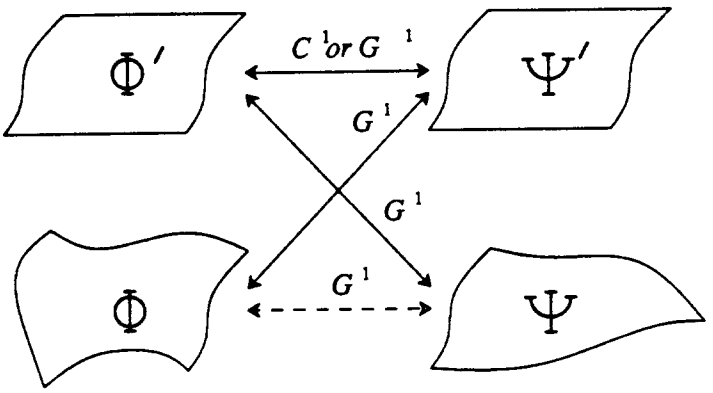


Figure 6.3: *The relationship of the two real patches and the two basis patches.*

vertices:

$$\mathbf{a}_0 = k_0\mathbf{b}'_0 + h_0\mathbf{s}_0, \quad \mathbf{a}_3 = k_1\mathbf{b}'_2 + h_1\mathbf{s}_2. \tag{6.10}$$

The interior control points of the real patch Φ can then be determined from:

$$\mathbf{a}_1 = \frac{1}{3}(2k_0\mathbf{b}'_1 + k_1\mathbf{b}'_0 + 2h_0\mathbf{s}_1 + h_1\mathbf{s}_0), \quad \mathbf{a}_2 = \frac{1}{3}(k_0\mathbf{b}'_2 + 2k_1\mathbf{b}'_1 + h_0\mathbf{s}_2 + 2h_1\mathbf{s}_1). \tag{6.11}$$

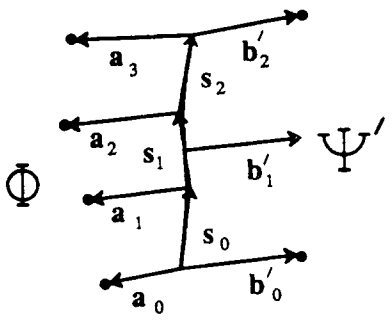


Figure 6.4: *Control points of the real patch Φ and the basis patch Ψ'.*

Chiyokura originally defined $\mathbf{b}'_0$ to be a unit vector in the plane of $(\mathbf{a}_0, \mathbf{s}_0)$, orthogonal to $\mathbf{s}_0$, and $\mathbf{b}'_2$ to be a unit vector in the plane of $(\mathbf{a}_3, \mathbf{s}_2)$, orthogonal to $\mathbf{s}_2$. However, for some highly warped boundary meshes, this choice led to some unsatisfactory 'wiggles' in the patches, and thus different default values that are more naturally derived from the given geometry of the boundary curves are more preferable [25]:

$$\mathbf{b}'_0 = \frac{\mathbf{b}_0 - \mathbf{a}_0}{|\mathbf{b}_0 - \mathbf{a}_0|}, \quad \mathbf{b}'_2 = \frac{\mathbf{b}_3 - \mathbf{a}_3}{|\mathbf{b}_3 - \mathbf{a}_3|}. \tag{6.12}$$

Chiyokura's default value for the middle control vector is:

$$\mathbf{b}'_1 = \frac{1}{2}(\mathbf{b}'_0 + \mathbf{b}'_2). \tag{6.13}$$

Chiyokura's method has several desirable features. All the patch information is derived exclusively from boundary curves. It is local, so that a change in a cubic boundary results in changes on only the two patches adjoining this boundary. The computation is done independently for each patch and it is very simple. A smooth pleasing-looking surface is produced for 'reasonable' meshes of cubic boundary curves. For these reasons, Chiyokura's method is currently used in our implementation.

6.3. Shape Parameters between Adjoining Patches

In this approach, all the patch information is derived exclusively from boundary curves. The method provides some good default placement of interior control points of Gregory patches to assure geometric continuity between the patches. However, there are 5 unused degrees of freedom available on each boundary (Section 6.2.1). By using some of these degrees of freedom, our approach preserves the geometric continuity and the interpolating property but offers the user the opportunity to modify local behavior of the patches near the common boundary. This is achieved by varying some *shape parameters* associated with the seam between the two patches, which, in turn, alter the position of the interior control points. The default shape parameters correspond to the default shape produced by Chiyokura's revised method [25].

We start by analyzing Chiyokura's method in greater detail and identifying several implicit assumptions made during the construction of the control points. We then look in detail at all the available degrees of freedom present in the system with the purpose of utilizing as many of them as possible to alter the shape of the surface. Finally, we present a set of practical shape parameters that is used in our actual implementation.

6.3.1. Construction of Geometrically Continuous Patches

6.3.1.1. Degrees of Freedom

Chiyokura's method results in a unique default solution for a given mesh. This uniqueness is a consequence of several assumptions implicitly made during the construction of the basis patches which throw away several additional degrees of freedom (Fig. 6.4):

1. The basis patches are constructed to be C^1 continuous, when simple G^1 continuity would be sufficient.
2. The vectors $\mathbf{b}'_0$ and $\mathbf{b}'_2$ are chosen in a very special manner determined by the geometry of the cubic boundary curves at their end vertices. In general, both of these vectors have two degrees of freedom; it is only necessary that they each lie in the given tangent plane at the

vertex.

3. The central vector $\mathbf{b}'_1$ is chosen as the arithmetic mean between $\mathbf{b}'_0$ and $\mathbf{b}'_2$, thus forcing a linear cross-boundary derivative for the basis patches. This vector can be chosen without any constraints and thus can add three more degrees of freedom.

Considering just a single basis patch and the conditions that guarantee its G^1 continuity with a real patch, one can see that the vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ depend on the 9 coordinates of the $\mathbf{b}'_i$'s. However, since $\mathbf{b}'_0$ and $\mathbf{b}'_2$ are restricted by planarity constraints and since proportional scaling of the $\mathbf{b}'_i$'s does not change the vectors $\mathbf{a}_i$, there are only 6 free parameters. We thus seem to have as many free parameters that can affect the shape of the final patch as there are coordinates to be determined.

It is thus conceivable to give the user control over these 6 degrees of freedom to fine-tune the shape of the surface in the neighborhood of the seam. However several practical issues need to be considered.

1. First of all, we know that the overall system of two Gregory patches joining with G^1 continuity on a cubic boundary curve has only 5 degrees of freedom (Section 6.2.1). From this we can infer that there is some redundancy in the set of parameters $\mathbf{b}'_i$, and that they may be strongly coupled in their effects on the shape of the final surface.

2. We need to be concerned whether all the parameters have some useful influence on the final shape. It turns out that the lengths and directions of the two outer vectors $\mathbf{b}'_0$ and $\mathbf{b}'_2$ have very little influence on the surface shape since, when the real patch is formed, they get 'normalized' to the cross boundary derivative vectors that are implicitly given by the Bézier points of the cubic boundary curves. In some cases, when these two vectors are chosen to deviate strongly from the 'natural' directions of the cross boundary derivative, some unpleasant 'wiggles' may be produced in the Gregory patches.

3. Controlling the 6 degrees of freedom in the $\mathbf{b}'_i$'s and then using these three vectors to define C^1 continuous basis patches couples the shape of the two final Gregory patches more strongly than they need to be coupled. It seems worthwhile to investigate how much independent control of the two patches can be gained if we generalize the continuity of the two basis patches from C^1 to G^1.

The control points of the real patch are completely determined by the control points of the basis patches, or, equivalently, the vectors $\mathbf{b}'_i$. A detailed analysis of the constraints imposed by G^1 continuity on these vectors is therefore necessary.

6.3.1.2. Geometric Continuity Between a Pair of Basis Patches

Consider a pair of (basis) patches with quadratic cross-boundary derivatives (Fig. 6.5) Then the tangent and cross-boundary derivatives can be expressed as (we omit the primes on $\mathbf{a}'$ and $\mathbf{b}'$ in this Section):

$$D\Gamma(v) = 3\sum_{i=0}^{2} \mathbf{s}_i\, B_i^2(v), \quad D\Phi'(v) = \sum_{i=0}^{2} \mathbf{a}_i\, B_i^2(v), \quad D\Psi'(v) = \sum_{i=0}^{2} \mathbf{b}_i\, B_i^2(v). \qquad (6.14)$$

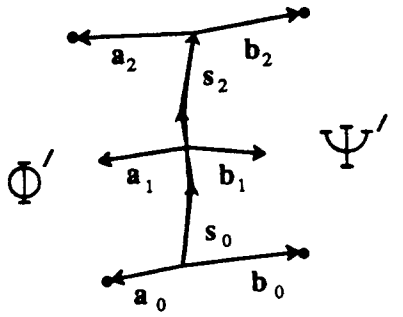


Figure 6.5: *G^1 continuity conditions between two basis patches.*

The determinant equation in this case will be of degree 6, giving 7 constraints for the 18 coordinates of $\mathbf{a}_i$ and $\mathbf{b}_i$, $i = 1,2,3$. Keeping the outer vectors in the two planes defined at the two vertices automatically satisfies two equations, leaving 5 constraints for the 14 free coordinates. Thus we seem to have 9 degrees of freedom. However, manipulating the outer vectors is not very effective and may result in unpredictable distortions in the patches. It is preferable to set them to some reasonable default positions, thus reducing to only 6 the degrees of freedom in the set of control vectors (the coordinates of $\mathbf{a}_1$ and $\mathbf{b}_1$). On the other hand, it appears promising to try to manipulate the two central control vectors $\mathbf{a}_1$ and $\mathbf{b}_1$ individually, resulting in non-collinearity and/or different magnitudes of these vectors, with the goal to be able to modify the shape of one patch independently of the other.

Assuming that $\mathbf{b}_0 = -\mathbf{a}_0$ and $\mathbf{b}_2 = -\mathbf{a}_2$, expanding the determinant equation, and equating its coefficients of $v^i(1-v)^{6-i}$, $i = 0,\ldots,6$ leads to the following system:

$$\begin{aligned}
\det(\mathbf{a}_0, \mathbf{b}_0, \mathbf{s}_0) &= 0,\\
\det(\mathbf{a}_0, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_0) &= 0,\\
\det(\mathbf{a}_1, \mathbf{b}_1, \mathbf{s}_0) + \det(\mathbf{a}_0, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_1) &= 0,\\
4\det(\mathbf{a}_1, \mathbf{b}_1, \mathbf{s}_1) + \det(\mathbf{a}_0, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_2) + \det(\mathbf{a}_2, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_0) &= 0, \qquad (6.15)\\
\det(\mathbf{a}_1, \mathbf{b}_1, \mathbf{s}_2) + \det(\mathbf{a}_2, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_1) &= 0,\\
\det(\mathbf{a}_2, \mathbf{a}_1+\mathbf{b}_1, \mathbf{s}_2) &= 0,\\
\det(\mathbf{a}_2, \mathbf{b}_2, \mathbf{s}_2) &= 0.
\end{aligned}$$

The first and the last equations are the planarity conditions, which are satisfied if we keep the outer vectors in the planes given at the vertices. The remaining 5 equations can be viewed as constraints to determine the 6 coordinates of $\mathbf{a}_1$ and $\mathbf{b}_1$. Thus, in the most general case, if we

predefine $\mathbf{a}_0$, $\mathbf{a}_2$, $\mathbf{b}_0$, and $\mathbf{b}_2$, there is only 1 degree of freedom left.

However, if we set $\mathbf{b}_1 = -\mathbf{a}_1$, all five equations will be trivially satisfied, and we will have 3 degrees of freedom left. This special case corresponds to enforcing *parametric continuity* between the basis patches. It turns out that making the basis patches to be C^1 rather than G^1 continuous, is not actually restrictive, but gives us the freedom to choose $\mathbf{a}_1$ freely! This is a consequence of the special way we have chosen the outer control vectors.

6.3.1.3. Trying to Solve the Determinant Equation by Premultiplication

Rather than trying to solve a complicated determinant equation as the one above, one can premultiply each derivative by an unknown polynomial, and then equate the resulting linear combination to zero:

$$\alpha(v)\, D\Phi(v) + \mu(v)\, D\Psi(v) + \lambda(v)\, D\Gamma(v) = 0. \tag{6.16}$$

In this case, a system of *vector* equations, i.e. three independent scalar systems, one for each coordinate, will have to be solved. This is simpler than trying to solve the determinant equation which mixes the components along different axes. Moreover, the systems will be linear in terms of the coordinates of the control vectors $\mathbf{a}_i$ and $\mathbf{b}_i$. In this Section we investigate how much freedom we lose through such methods.

Consider a pair of basis patches in which the 3 derivatives are quadratic. The determinant equation will be of degree 6 leading to 7 constraints (including the planarity conditions). If we premultiply each derivative with a constant, we will have 3 vector equations (for orders 0 through 2), and hence 9 constraints. However, given the extra two premultipliers (the third can always be assumed to be 1), there are again 7 net constraints on the basic degrees of freedom.

Raising the degree of the premultiplying polynomials by 1, introduces an additional vector equation with 3 more constraints, but at the same time yields 3 more premultiplication coefficients. Thus the number of actual constraints is unchanged, and it appears that the order of the premultiplication polynomial is not important.

However, the degree of the premultiplier does change the amount of 'coupling' between the different vector equations and affects the generality of the solution. Premultiplying by a constant or a low-order polynomial may introduce additional constraints or remove redundancies already present in the system. On the other hand, if the premultiplier is high enough, so that the degree of the resulting equation is the same as the degree of the determinant equation, premultiplication will be equivalent to solving the determinant equation directly.

Premultiplication with low-order factors is often appropriate for practical applications. It gives perfectly valid solutions. The disadvantage is, of course, that the resulting solutions do not have the full generality of the determinant equation. The limitations on the solutions found by low order premultiplication can easily be seen on some special degenerate cases. In our example, if $\mathbf{a}_0$, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_0$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are all parallel but not necessarily equal in magnitude, then the patches

will be G^1 continuous — but this solution will be overlooked if premultiplication with constants were used.

There is another problem: low order premultiplication may not be effective. The determinant equation will be satisfied if some constants α and β can be found such that

$$\mathbf{a}_0 = \alpha\mathbf{b}_0 + \beta\mathbf{s}_0, \quad \mathbf{a}_1 = \alpha\mathbf{b}_1 + \beta\mathbf{s}_1, \quad \mathbf{a}_2 = \alpha\mathbf{b}_2 + \beta\mathbf{s}_2. \tag{6.17}$$

We could define one of the basis patches by choosing the six free parameters of the $\mathbf{b}_i$'s (Section 6.2.1). We then still can pick freely the two parameters α and β. The obvious way to use these two parameters is to produce a non-collinearity and a difference in magnitude between the central vectors $\mathbf{a}_1$ and $\mathbf{b}_1$ so that the 'bulges' of the two patches can be controlled in position and size independently for the two adjoining patches. The problem is again one of 'renormalization'. A particular choice of α and β will also determine $\mathbf{a}_0$ and $\mathbf{a}_2$, not just $\mathbf{a}_1$. When the final Gregory patch gets derived from the basis patch on the other side of the seam, an inherent scaling and realignment of the cross-boundary derivative vectors takes place, so that the outer vectors $\mathbf{a}_0$ and $\mathbf{a}_2$ fall onto the vectors implied by the first Bézier points on the transversal boundary curves of this patch. This operation may diminish or completely wipe out the desired effect near the middle of the seam.

Premultiplication of each derivative with a linear polynomial rather than with a constant leads to a set of 10 ($2+3+3+2$, for various degrees of the polynomial) equations for 19 variables: two coordinates for each of $\mathbf{a}_i$, $\mathbf{b}_i$, $i = 0, 2$; three coordinates each for $\mathbf{a}_1$ and $\mathbf{b}_1$, and 5 coefficients of premultiplying polynomials. This leaves us with 9 degrees of freedom for this system. This number agrees with the number of freedoms derived earlier from the determinant equation (Section 6.2.2). However, a linear function still does not give us the necessary independent control over the middle control vectors $\mathbf{a}_1$ and $\mathbf{b}_1$.

It would be preferable to produce the non-collinearity of the central vectors while leaving the outer vectors oriented in their 'natural' directions given by the geometry of the transversal boundary curves. This requires premultiplication of the individual derivative expressions by a function that is at least of degree two — we need to be able to affect the center of the function while keeping the ends clamped.

However, the resulting system of 5 nonlinear vector equations for the various degrees of the Bernstein polynomial is too complicated to yield to a general solution. If $\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_2, \mathbf{b}_2$ are assumed to be predetermined, a single solution for $\mathbf{a}_1$ and $\mathbf{b}_1$ could be expected in general. The only special assumption for the outer vectors that we could find that would yield more flexibility, was the same condition used in the case of linear premultiplication:

$$\mathbf{a}_0 = -\mathbf{b}_0 \text{ and } \mathbf{a}_2 = -\mathbf{b}_2. \tag{6.18}$$

Again this results in the constraint $\mathbf{a}_1 = -\mathbf{b}_1$, i.e. C^1 continuity between the two basis patches.

The same analysis was also carried out for the more general case of a cubic cross-boundary derivative function. We did succeed in expressing the internal control points as (very involved)

functions of the premultiplication coefficients (see Appendix 1). However, this solution is without any practical merit. First, is too complicated; but more importantly, the effect of the values of the premultiplication coefficients has no intuitively understandable correlation with any desirable shape modifications (Section 6.3.2.1).

6.3.2. Introducing Shape Parameters

It clearly seems preferable to extract from the given set of dependencies and constraints a small but useful set of user friendly *handles*, that influence the shape of the surface near the seam in a robust and predictable manner. We first consider what types of shape control might be most desirable from a user's point of view, then discuss the controls that we have been able to realize.

6.3.2.1. Desirable Shape Controls Along a Given Seam

In the context of this work, the following features are considered unalterable geometrical constraints:

1. The shape of the cubic seam,
2. the tangent planes at the vertices at either end of the seam,
3. and the condition that the two patches interpolate the seam with G^1 continuity at all points of the common boundary.

From our earlier analysis (Section 6.2.1) we know that this system of two Gregory patches still has 5 degrees of freedom. Here are our goals how we would like to use these 5 freedoms to affect the shape of the surface in the neighborhood of the seam to the extent that the mathematical constraints allow us to do so.

1. *Tilt* controls the surface normal near the midpoint of the seam, by swiveling the surface around an axis collinear to the tangent vector in the middle of the seam. Of course the effect of this swivel operation will diminish towards the ends of the seam where the surface must smoothly match up with the tangent planes at the vertices.
2. *Bulge* controls the 'roundness'—'tension' of the patches near the middle of the seam by adjusting the magnitude of the cross-boundary derivative control vectors $\mathbf{a}_1$ and $\mathbf{b}_1$. If possible, we would like to adjust the bulge of patch Φ and of patch Ψ individually so as to be able to generate an asymmetric bulge with respect to the seam, or, conversely, correct for an asymmetric bulge produced by two meshes of unequal size or shape. It may be desirable to form symmetrical and anti-symmetrical combinations of these two parameters — if indeed they can be controlled independently. This would give the user the option to first control the 'general' (or 'global') bulge and then to fine-tune the bulging for any asymmetric bias.
3. *Shear* controls the shifting of the 'peaks of the bulges' in a direction parallel to the seam. Again, we hope that there is enough freedom in the system of equations to give us some independent control to do such a shifting of the bulge individually on the two patches. And

again, we might consider forming symmetrical and anti-symmetrical combinations of these two operations to shift the peaks symmetrically towards one end of the seam or to shear them antisymmetrically in opposite directions.

These operations are shown conceptually in Fig. 6.6. A detailed analysis, summarized in Sections 6.3, has shown that it is not possible to extract all five desirable individual controls from the system of equations.

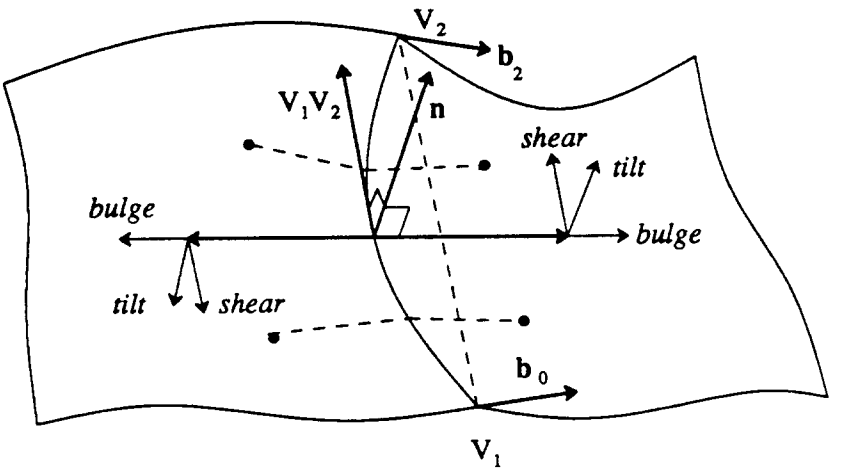


Figure 6.6: *Desirable shape parameters on a seam between two patches.*

6.3.2.2. Practical Shape Parameters

Using the special case of $\mathbf{a}'_1 = -\mathbf{b}'_1$, we have extracted a subset of shape parameters that best realize the desired controls discussed in Section 6.3.2.1. The outer control vectors for the C^1 continuous basis patches, $\mathbf{a}'_0 = -\mathbf{b}'_0$ and $\mathbf{a}'_2 = -\mathbf{b}'_2$, are determined according to equation (6.12). The middle control vector $\mathbf{a}'_1 = -\mathbf{b}'_1$ starts out from Chiyokura's default position, but can be modified with 3 degrees of freedom leading to the three shape parameters described below.

Tilt. The tilt parameter controls the component of the $\mathbf{b}'_1$ vector parallel to the cross product of the bulge and shear directions (see below). A non-zero tilt will result in one patch being 'lifted' locally near the seam, while the other patch is 'lowered'. The default value for tilt is 0, resulting in a surface that has the same tangent planes as Chiyokura's default solution.

Bulge. The bulge parameter defines the length of $\mathbf{b}'_1$ (in units of $\frac{1}{2}(\mathbf{b}'_0 + \mathbf{b}'_2)$). The default value is 1, which corresponds to Chiyokura's linear interpolation. A large positive bulge would result in flattening of the surface near the seam, while a small bulge will cause the surface to look like tightly stretched skin between the boundary curves. Large negative values will cause cusps and self-intersection of the resulting surface.

Shear. The shear parameter affects the component of $\mathbf{b'}_1$ parallel to the vector connecting the two endpoints of the shared boundary curve; it basically moves the control points of the Gregory patches parallel to the seam. A non-zero shear would cause the patches on either side of the boundary to move 'sideways' in different directions. The default value for shear is zero.

6.3.3. Results

The shape parameters have been systematically tested for their robustness and predictability on a set of eight different pairs of quadrilateral meshes ranging from completely flat to severely warped in both directions. Some of the more interesting cases are shown below (Fig. 6.3 – 6.7). The shape parameters are expressed in terms of the *control* vector (*bulge, tilt, shear*), with the default being (1,0,0). The common seam between the two patches is marked by bold dots at the end of the shared boundary curve.

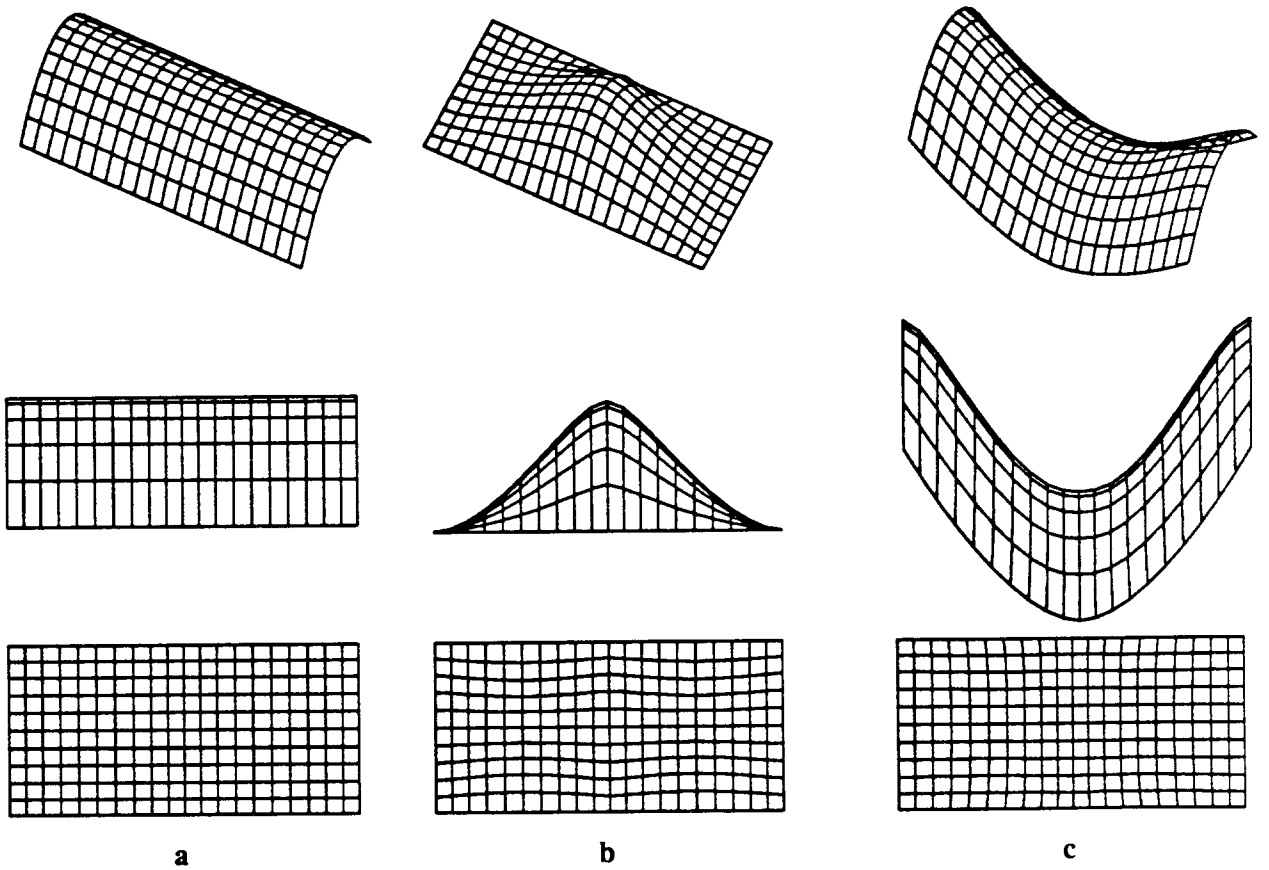


Figure 6.3: *Various test patches. Side and top views are also shown.*

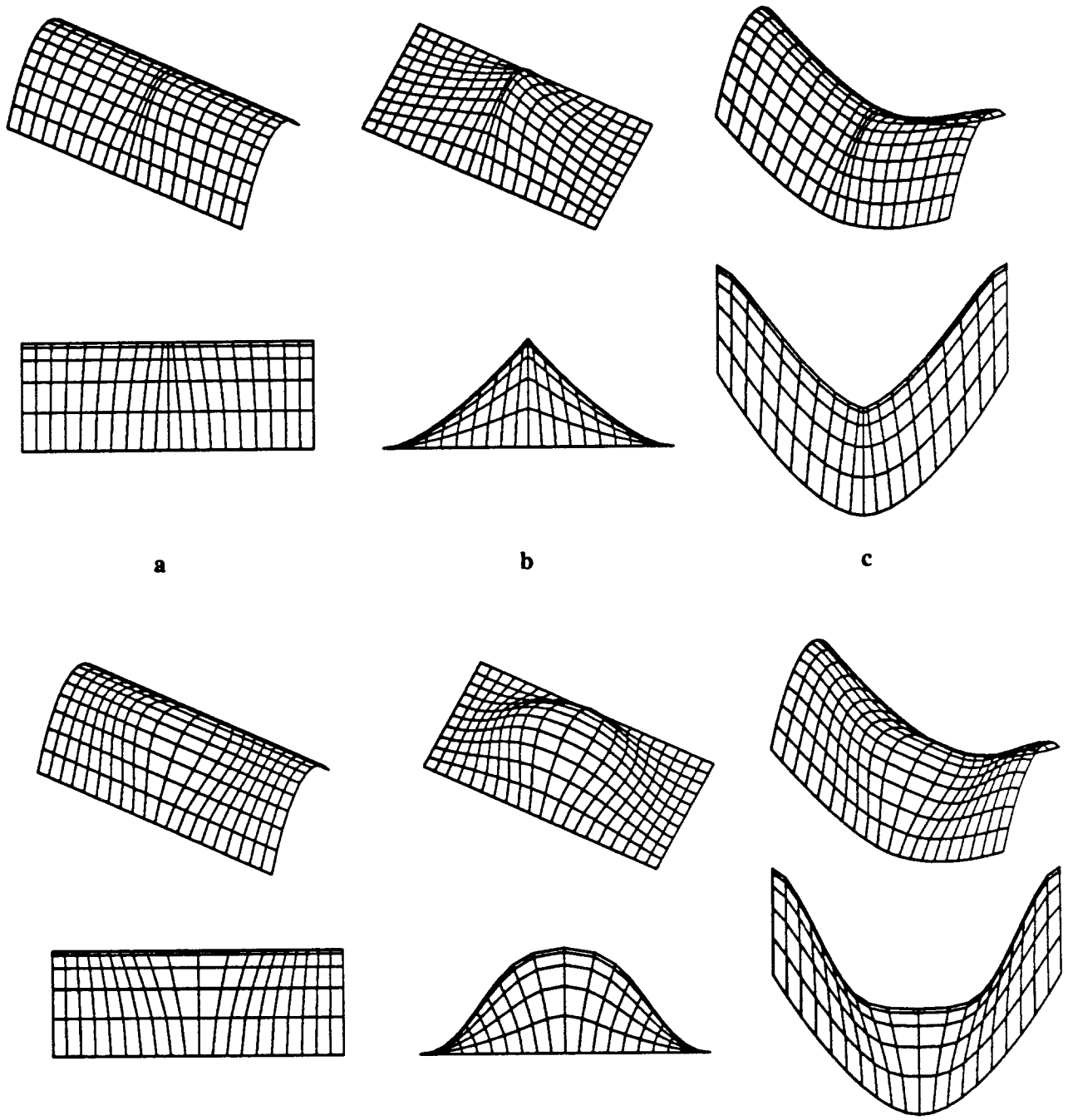


Figure 6.4: *Effects of the two values of the bulge parameter on the patches from Fig. 6.3 The control vector is (-1,0,0) for the top, and (5,0,0) for the bottom patches.*

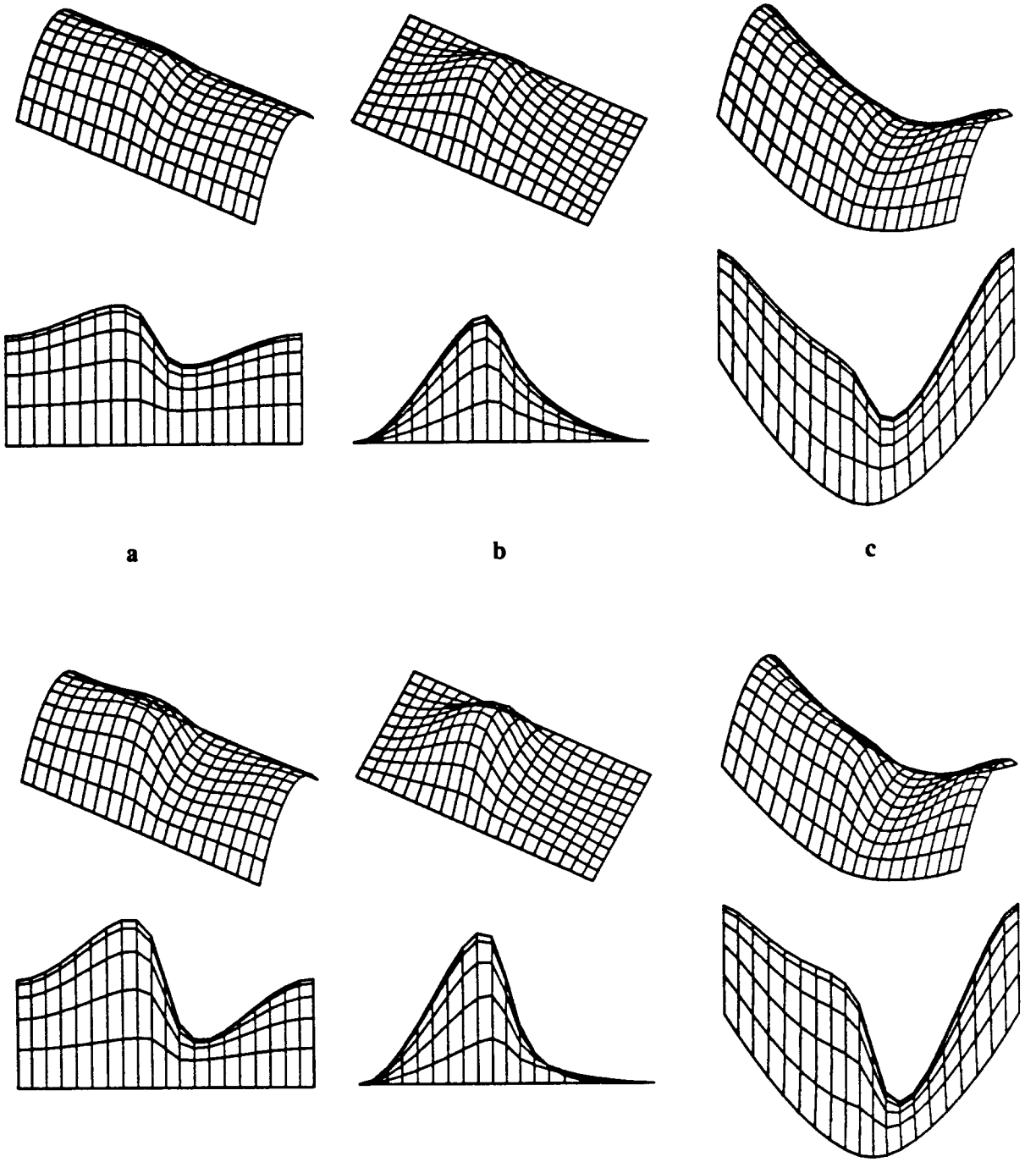


Figure 6.5: *Effects of the two values of the tilt parameter on the patches from Fig. 6.3. The control vector is (1,5,0) for the top, and (1,10,0) for the bottom patches.*

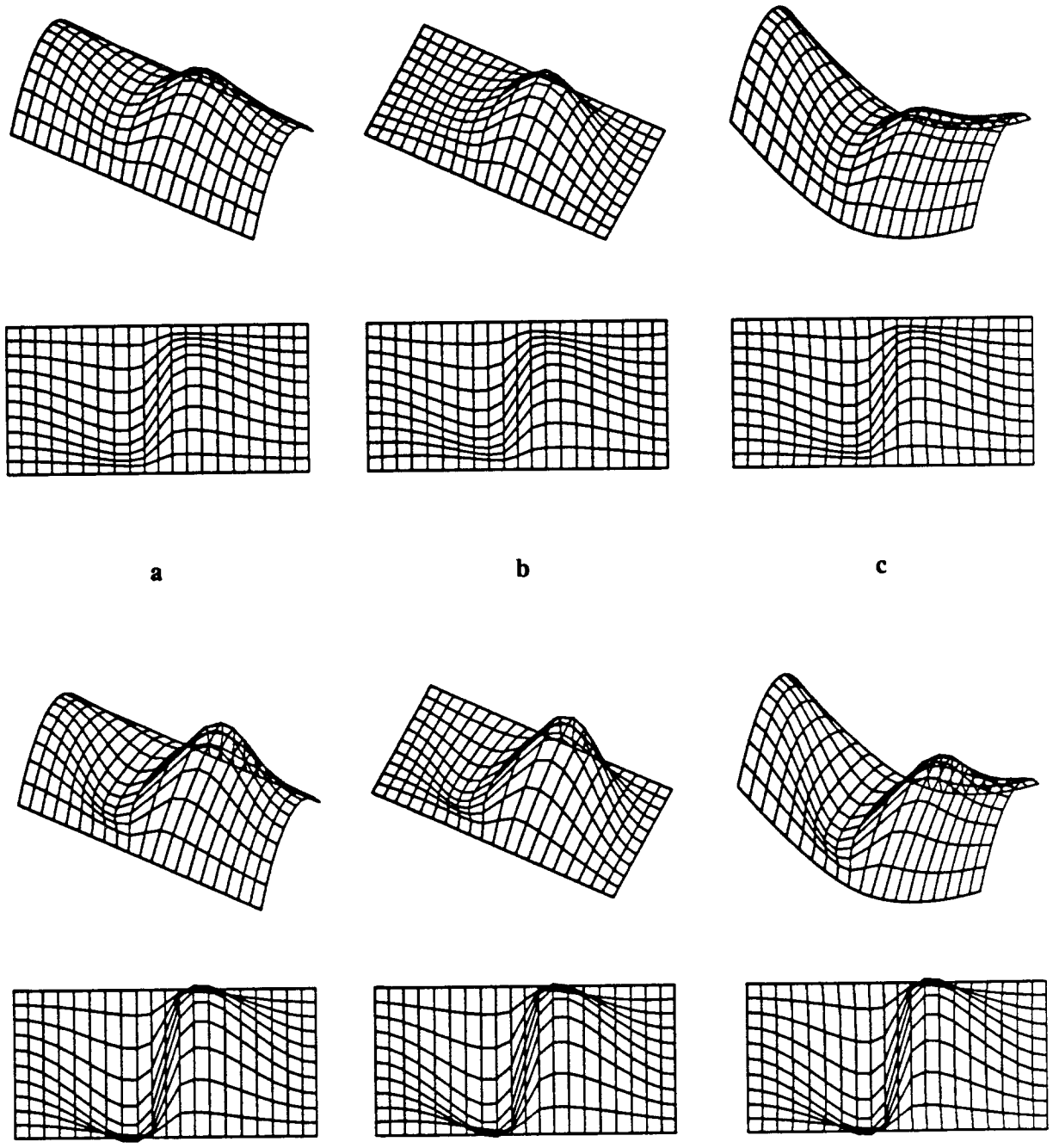


Figure 6.6: *Effects of the two values of the shear parameter on the patches from Fig. 6.3 The control vector is (1,0,5) for the top, and (1,0,10) for the bottom patches.*

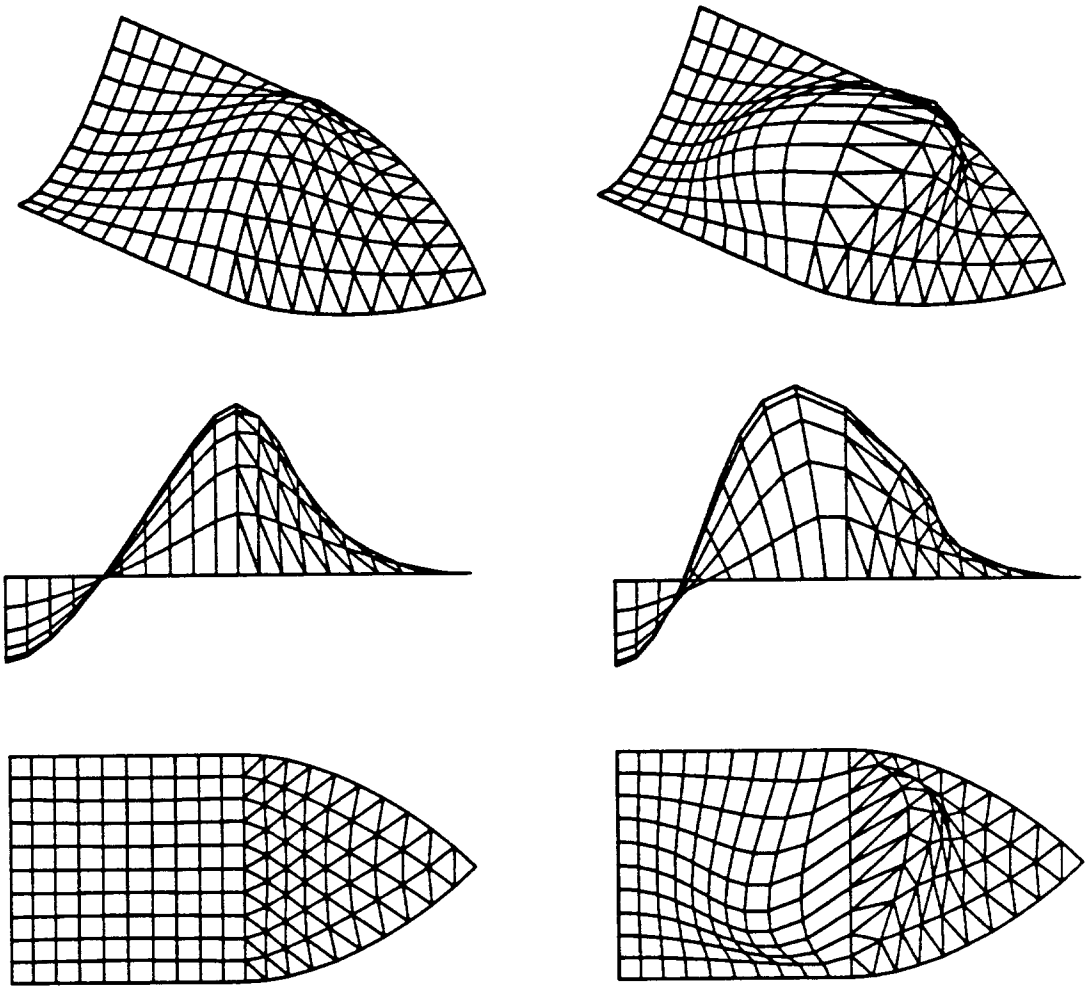


Figure 6.7: *A quadrilateral and a triangular patch with default (left) and non-default (right) values for tilt, bulge, and shear. The control vector for the patch on the right is (5,5,5).*

In addition, a random sampling of triangular and quadrilateral patches with strongly curved boundaries has also been used for testing. Of course, for all three shape parameters, values can be found that will produce a self-intersecting surface, but for reasonable values, the resulting patch shapes look pleasing and seem to be of practical use. Fig. 6.7 shows a strongly warped case involving a triangular and a quadrilateral patch.

In Fig. 6.4, the pair of patches **b** with the control vector (−1,0,0) appears to have a crease. For small bulge values the control points of the two patches lie very close to their common boundary. However, these patches are still G^1 continuous, and the seeming discontinuity in the tangent plane is due to the relatively small number of isoparametric curves used for rendering.

6.4. Summary

The described procedure for changing the local behavior of a surface composed of Gregory patches is a simple extension of Chiyokura's method. The discussed shape parameters are implemented in the UNICUBIX system.

As discussed in Section 6.2.1, there are 5 degrees of freedom across each boundary for the 12 unknown coordinates of the 2 pairs of the interior control points. The simple approach described in this paper utilizes 3 out these 5 degrees of freedom to provide 3 intuitively understandable, easy to use shape parameters: *bulge*, *tilt*, and *shear*. These parameters affect the shape of *both* adjoining patches. We have not succeeded in extracting additional degrees of freedom to control the shape of the individual patches. It appears that this is made impossible by the constraint that we demand G^1 continuity along the *entire* boundary curve. As a matter of fact, it appears that the special case

$$\mathbf{a}_0' = -\mathbf{b}_0',\ \mathbf{a}_1' = -\mathbf{b}_1',\ \mathbf{a}_2' = -\mathbf{b}_2'$$

is a "lucky break" yielding 3 separate and independent shape parameters in a strongly coupled and highly constrained system of equations.

The shape parameters can be viewed as high-level controls that either affect the overall appearance of an object, or the local shape of some patches. The surface produced by Chiyokura's method is the default solution which the user can change to satisfy particular needs. Several shape parameters can have non-default values at the same time. They can be changed globally, or individually for some of the boundaries. Moreover, each cubic boundary curve could have additional shape parameters that control the shape of the curve itself. These features provide a rich set of tools for fine-tuning the shape of an object.

Although the method has been tailored to quadrilateral and triangular Gregory patches, a subdivision procedure can be used to convert the surface into a union of Bézier patches (Chapter 7).

7

Representation with Bézier Patches

In this chapter, a procedure for representation of the surface with Bézier, rather than with Gregory patches is described. A single Gregory patch is substituted by several Bézier patches. Simple efficient ways to construct the patches for triangular, regular n-gon, and quadrilateral faces are presented. The resulting surface is guaranteed to be geometrically continuous across all interior and exterior boundaries of the Bézier patches.

7.1. Introduction

Chiyokura's method, used for surface construction in Chapter 5, works exclusively with Gregory patches. The two interior control points of the Gregory patch associated with each boundary are computed independently of the other interior control points. In Bézier patches, each interior control point is 'shared' by the two 'adjacent' boundaries, and, therefore, the above method will not work for blending Bézier patches across *given* cubic boundaries (Section 2.3.4).

As mentioned in Section 2.3.4, Gregory patches are *rational* and their parametric degree (7) is quite high. It would be preferable to use *polynomial* Bézier patches of lower degree, because many existing highly efficient algorithms for subdivision, intersection, etc. could then be used. This Chapter describes the process of substituting a Gregory patch by a union of several Bézier patches in such a way, that overall G^1 continuity is maintained. This is almost a subdivision of one patch into several, except that the original patch is actually never defined.

We start by looking at many potential ways of subdividing a triangle, quadrilateral, or a general polygonal mesh into triangular and/or quadrilateral Bézier patches. Since the cubic boundaries are predetermined, Chiyokura's method will produce a pair of Gregory interior control points for each vertex of the polygon. These pairs of vertices will not necessarily merge into a single vertex to produce a Bézier patch. Thus, an extra *interior boundary* is needed that passes through each vertex of the initial polygon, and thereby produces two separate interior control

points associated with each vertex even for the case of Bézier patches.

The general approach of using several non-overlapping patches to represent a single facet in a defining topological mesh is usually referred to as *splitting approach*. Several methods of this type have been proposed [39, 73, 100]. However, in these methods, large systems of linear equations typically have to solved for each patch. In our approach, expressions for control points of the patches are given in a *closed form*, making the process of splitting very simple and efficient.

7.2. Filling the Mesh with Triangular Bézier Patches

7.2.1. Triangular Mesh Subdivision

In the simplest case, if we want to fill a triangular mesh with Bézier patches, the subdivision has to look as shown in Fig. 7.1. We construct three Bézier patches and ensure geometric continuity (G^1) between them. $B_i, C_i, D_i,\ i = 1,2$ are control points on the cubic curves, which are considered predetermined as far as this subdivision task is concerned. $S_i, P_i,\ i = 1,2$ are the control points of the cubic boundaries between the subpatches and are yet to be determined, as well as the three control points inside each subpatch.

Assume for the moment that S_1, S_2, and S_3 are known. Points $L_{13}, M_{13}, L_{12}, K_{12}, K_{23}, M_{23}$ can then be determined without much difficulty using Chiyokura's method [25]. They ensure smoothness across the given external boundaries of the patch.

Clearly, S_1 must lie in the plane of $T_1B_1C_1$, and analogously, S_2 in the plane of $T_2C_2D_2$, S_3 in the plane of $T_3B_2D_1$, and, finally, Z in the plane of $P_1P_2P_3$. Thus, there are 3 degrees of freedom each for points $P_1, P_2, P_3, N_{12}, N_{13}, N_{23}$ and 2 degrees of freedom each for points S_1, S_2, S_3, Z. This gives a total of 26 degrees of freedom.

We use Farin's method [39] of ensuring G^1 continuity between the subpatches. For subpatches T_1ZT_3 and T_1ZT_2 to be continuous, it is necessary that the following ratios of triangle areas match:

$$\frac{\textbf{area}\ (T_1S_1C_1^q)}{\textbf{area}\ (T_1S_1B_1^q)} = \frac{\textbf{area}\ (P_1ZP_2^q)}{\textbf{area}\ (P_1ZP_3^q)}. \tag{7.1}$$

The superscript q indicates that the corresponding points are the degree-elevated *quartic* control points of the cubic boundaries [41]. For example,

$$\mathbf{C}_1^q = \frac{3}{4}\,\mathbf{C}_1 + \frac{1}{4}\,\mathbf{T}_1, \qquad \mathbf{P}_1^q = \frac{3}{4}\,\mathbf{P}_1 + \frac{1}{4}\,\mathbf{Z}. \tag{7.2}$$

The area ratio constraint can be equivalently expressed in terms of the *cubic* control points:

$$\frac{\textbf{area}\ (T_1S_1C_1)}{\textbf{area}\ (T_1S_1B_1)} = \frac{\textbf{area}\ (P_1ZP_2)}{\textbf{area}\ (P_1ZP_3)}. \tag{7.3}$$

Furthermore, to guarantee G^1 continuity across the internal boundary, two additional vector (six

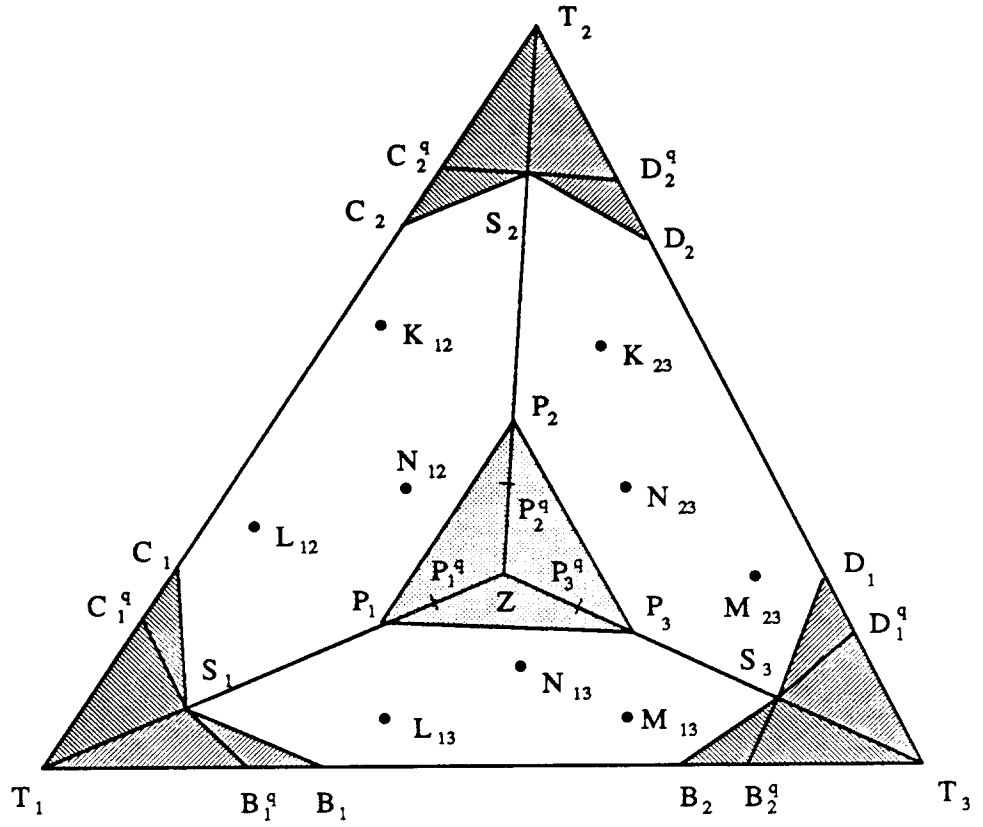


Figure 7.1: *Triangular patch subdivision.*

scalar) equations must be satisfied, involving points $T_1, S_1, P_1, Z, L_{12}, L_{13}, N_{12}, N_{13}$. Therefore, there are 7 constraints across each boundary, which gives a total of 21 for the whole subdivision. So there are 5 extra degrees of freedom left for the whole triangle.

We make use of these extra degrees of freedom as follows. First, we use 2 degrees of freedom and choose Z to be in the *center of gravity* of the triangle $P_1P_2P_3$, that is,

$$\mathbf{Z} = \frac{1}{3}\mathbf{P}_1 + \frac{1}{3}\mathbf{P}_2 + \frac{1}{3}\mathbf{P}_3. \tag{7.4}$$

Therefore, the areas of triangles P_1ZP_2, P_1ZP_3, and P_2ZP_3 are *all equal.*

Now, to satisfy the above area ratio condition, areas $T_1S_1C_1^q$ and $T_1S_1B_1^q$ must be equal, and analogously for the other two interior boundaries. It's sufficient that S_1 lies *anywhere on the median* through T_1 of the triangle $T_1C_1^qB_1^q$ (or of the triangle $T_1C_1B_1$). The exact position of the points S_i will be determined by three chosable parameters. Now, the number of the remaining degrees of freedom is 18 and matches exactly the number of constraints.

There is another way to express the area ratio condition, which we actually used in solving the remaining 6 vector equations. The relevant points can be expressed in barycentric

coordinates, using the corresponding points on the other side of the respective interior boundaries:

$$\begin{aligned}
\text{Boundary } (T_1Z): \quad & \mathbf{C}_1^g = \alpha_1 \mathbf{T}_1 + \alpha_2 \mathbf{S}_1 + \alpha \mathbf{B}_1^g, \quad \alpha_1 + \alpha_2 + \alpha = 1, \\
& \mathbf{P}_2^g = \alpha_3 \mathbf{P}_1 + \alpha_4 \mathbf{Z} + \alpha \mathbf{P}_3^g, \quad \alpha_3 + \alpha_4 + \alpha = 1, \\
\text{Boundary } (T_2Z): \quad & \mathbf{D}_2^g = \beta_1 \mathbf{T}_2 + \beta_2 \mathbf{S}_2 + \beta \mathbf{C}_2^g, \quad \beta_1 + \beta_2 + \beta = 1, \\
& \mathbf{P}_3^g = \beta_3 \mathbf{P}_2 + \beta_4 \mathbf{Z} + \beta \mathbf{P}_1^g, \quad \beta_3 + \beta_4 + \beta = 1, \\
\text{Boundary } (T_3Z): \quad & \mathbf{B}_2^g = \gamma_1 \mathbf{T}_3 + \gamma_2 \mathbf{S}_3 + \gamma \mathbf{D}_1^g, \quad \gamma_1 + \gamma_2 + \gamma = 1, \\
& \mathbf{P}_1^g = \gamma_3 \mathbf{P}_3 + \gamma_4 \mathbf{Z} + \gamma \mathbf{P}_2^g, \quad \gamma_3 + \gamma_4 + \gamma = 1.
\end{aligned} \tag{7.5}$$

The area ratio constraints for all three boundaries are equivalent to the equations

$$\begin{aligned}
\alpha_1 + \alpha_2 &= \alpha_3 + \alpha_4, \\
\beta_1 + \beta_2 &= \beta_3 + \beta_4, \\
\gamma_1 + \gamma_2 &= \gamma_3 + \gamma_4.
\end{aligned} \tag{7.6}$$

When we choose S_i on the median of the corresponding triangle, and Z in the center of gravity of $P_1P_2P_3$, we actually set

$$\begin{aligned}
&\alpha = -1, \quad \alpha_2 = 2 - \alpha_1, \quad \alpha_3 = -\frac{3}{4}, \quad \alpha_4 = \frac{11}{4}, \\
&\beta = -1, \quad \beta_2 = 2 - \beta_1, \quad \beta_3 = -\frac{3}{4}, \quad \beta_4 = \frac{11}{4}, \\
&\gamma = -1, \quad \gamma_2 = 2 - \gamma_1, \quad \gamma_3 = -\frac{3}{4}, \quad \gamma_4 = \frac{11}{4}.
\end{aligned} \tag{7.7}$$

The parameter α can be thought of as the negative ratio of the areas of the triangles $T_1S_1C_1$ and $T_1S_1B_1$, or, which is the same, as the negative ratio of the areas of P_1ZP_2 and P_1ZP_3. The parameters β and γ have the analogous geometric interpretation.

The six vector equations that we need to solve are listed below:

$$\begin{aligned}
\mathbf{L}_{12} &= \frac{2}{3}(\alpha_1\mathbf{S}_1 + (2-\alpha_1)\mathbf{P}_1) + \frac{1}{3}(-\frac{3}{4}\mathbf{T}_1 + \frac{11}{4}\mathbf{S}_1) - \mathbf{L}_{13} \\
\mathbf{N}_{12} &= \frac{1}{3}(\alpha_1\mathbf{P}_1 + (2-\alpha_1)\mathbf{Z}) + \frac{2}{3}(-\frac{3}{4}\mathbf{S}_1 + \frac{11}{4}\mathbf{P}_1) - \mathbf{N}_{13}, \\
\mathbf{K}_{23} &= \frac{2}{3}(\beta_1\mathbf{S}_2 + (2-\beta_1)\mathbf{P}_2) + \frac{1}{3}(-\frac{3}{4}\mathbf{T}_2 + \frac{11}{4}\mathbf{S}_2) - \mathbf{K}_{12}, \\
\mathbf{N}_{23} &= \frac{1}{3}(\beta_1\mathbf{P}_2 + (2-\beta_1)\mathbf{Z}) + \frac{2}{3}(-\frac{3}{4}\mathbf{S}_2 + \frac{11}{4}\mathbf{P}_2) - \mathbf{N}_{12}, \\
\mathbf{M}_{13} &= \frac{2}{3}(\gamma_1\mathbf{S}_3 + (2-\gamma_1)\mathbf{P}_3) + \frac{1}{3}(-\frac{3}{4}\mathbf{T}_3 + \frac{11}{4}\mathbf{S}_3) - \mathbf{M}_{23},
\end{aligned} \tag{7.8}$$

$$\mathbf{N}_{13} = \frac{1}{3}(\gamma_1 \mathbf{P}_3 + (2-\gamma_1)\,\mathbf{Z}) + \frac{2}{3}(-\frac{3}{4}\,\mathbf{S}_3 + \frac{11}{4}\,\mathbf{P}_3) - \mathbf{N}_{23}.$$

With this, points P_1, P_2, P_3 can be determined directly from just one equation each. The remaining 3 equations form a simple linear system, from which N_{12}, N_{13}, N_{23} can be found. Solutions to these equations are:

$$\begin{aligned}
\mathbf{P}_1 &= \frac{1}{8\,(2-\alpha_1)}\,(3\,\mathbf{T}_1 - (8\alpha_1 + 11)\,\mathbf{S}_1 + 12\,\mathbf{L}_{13} + 12\,\mathbf{L}_{12}),\\
\mathbf{P}_2 &= \frac{1}{8\,(2-\beta_1)}\,(3\,\mathbf{T}_2 - (8\beta_1 + 11)\,\mathbf{S}_2 + 12\,\mathbf{K}_{23} + 12\,\mathbf{K}_{12}),\\
\mathbf{P}_3 &= \frac{1}{8\,(2-\gamma_1)}\,(3\,\mathbf{T}_3 - (8\gamma_1 + 11)\,\mathbf{S}_3 + 12\,\mathbf{M}_{23} + 12\,\mathbf{M}_{13}),
\end{aligned} \tag{7.9}$$

$$\begin{aligned}
\mathbf{N}_{12} &= \frac{1}{36}(-9\,\mathbf{S}_1 - 9\,\mathbf{S}_2 + 9\,\mathbf{S}_3 + (4\alpha_1 - 2\beta_1 + 2\gamma_1 + 37)\,\mathbf{P}_1 + (-2\alpha_1 + 4\beta_1 +\\
&\qquad 2\gamma_1 + 37)\,\mathbf{P}_2 + (-2\alpha_1 - 2\beta_1 - 4\gamma_1 - 29)\,\mathbf{P}_3),\\
\mathbf{N}_{13} &= \frac{1}{36}(-9\,\mathbf{S}_1 + 9\,\mathbf{S}_2 - 9\,\mathbf{S}_3 + (4\alpha_1 + 2\beta_1 - 2\gamma_1 + 37)\,\mathbf{P}_1 + (-2\alpha_1 - 4\beta_1 -\\
&\qquad 2\gamma_1 - 29)\,\mathbf{P}_2 + (-2\alpha_1 + 2\beta_1 + 4\gamma_1 + 37)\,\mathbf{P}_3),\\
\mathbf{N}_{23} &= \frac{1}{36}(9\,\mathbf{S}_1 - 9\,\mathbf{S}_2 - 9\,\mathbf{S}_3 + (-4\alpha_1 - 2\beta_1 - 2\gamma_1 - 29)\,\mathbf{P}_1 + (2\alpha_1 + 4\beta_1 -\\
&\qquad 2\gamma_1 + 37)\,\mathbf{P}_2 + (2\alpha_1 - 2\beta_1 + 4\gamma_1 + 37)\,\mathbf{P}_3),
\end{aligned} \tag{7.10}$$

As we can see, the interior control points depend on the arbitrary parameters α_1, β_1, and γ_1. Therefore, the shape of the final surface will also depend on them. This fact will be used in quadrilateral subdivision in Section 7.3.2, where the shape of the patches will be adjusted by the judicial choice of the parameters.

Just as we picked Z in the center of the triangle $P_1P_2P_3$, we can choose S_1 to be in the center of gravity of $T_1C_1B_1$ (the center of gravity of a triangle is at the intersection of its medians), and analogously for S_2 and S_3.

The choice of placing the points Z and S_i at the center of gravity of the corresponding triangle is inspired by looking at the completely regular case of the equilateral triangle. In this case Z would be in the center of gravity of the (big) triangle, and S_i and P_i would divide the lines joining each vertex with Z into 3 equal parts. Cubics control points would also divide each triangle side into three equal parts. But this situation corresponds to our initial choice of picking S_i and Z in the centers of gravity of the corresponding triangles.

In this case, in addition to (7.7),

$$\alpha_1 = \beta_1 = \gamma_1 = -\frac{1}{4}, \quad \alpha_2 = \beta_2 = \gamma_2 = \frac{9}{4}. \tag{7.11}$$

With these simplifications, the control points are expressed as follows:

$$\begin{aligned} \mathbf{P}_1 &= \frac{1}{6}(\mathbf{T}_1 - 3\,\mathbf{S}_1 + 4\,\mathbf{L}_{13} + 4\,\mathbf{L}_{12}), \\ \mathbf{P}_2 &= \frac{1}{6}(\mathbf{T}_2 - 3\,\mathbf{S}_2 + 4\,\mathbf{K}_{23} + 4\,\mathbf{K}_{12}), \\ \mathbf{P}_3 &= \frac{1}{6}(\mathbf{T}_3 - 3\,\mathbf{S}_3 + 4\,\mathbf{M}_{23} + 4\,\mathbf{M}_{13}), \end{aligned} \tag{7.12}$$

$$\begin{aligned} \mathbf{N}_{12} &= \frac{1}{4}(-\mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 + 4\,\mathbf{P}_1 + 4\,\mathbf{P}_2 - 3\,\mathbf{P}_3), \\ \mathbf{N}_{13} &= \frac{1}{4}(-\mathbf{S}_1 + \mathbf{S}_2 - \mathbf{S}_3 + 4\,\mathbf{P}_1 - 3\,\mathbf{P}_2 + 4\,\mathbf{P}_3), \\ \mathbf{N}_{23} &= \frac{1}{3}(\mathbf{S}_1 - \mathbf{S}_2 - \mathbf{S}_3 - 3\,\mathbf{P}_1 + 4\,\mathbf{P}_2 + 4\,\mathbf{P}_3). \end{aligned} \tag{7.13}$$

For triangular subdivision, extensive testing has shown that placing all the four control points S_i and Z in the centers of gravity of the corresponding triangles produces good-looking surfaces. Therefore, the simpler equations (7.12) and (7.13) are used in the actual implementation.

7.2.2. Subdivision of the Patch over an Arbitrary Face

Let us now generalize the problem of the surface subdivision to the case of an arbitrary convex face with n edges. We would like to represent the surface over this face as a union of n Bézier patches, so that every pair of adjoining patches is G^1 continuous.

We use the notation and the approach of the previous section. Again, we know that the control points $L_{i,i+1}$ and $M_{i,i+1}$, $i = 1,...,n$ (Fig. 7.2) can readily be calculated once the points S_i have been chosen. Our task is, then, to determine points S_i, P_i, $N_{i,i+1}$, $i = 1,...,n$, and Z. There are 3 degrees of freedom each for the points P_i and $N_{i,i+1}$, and two degrees of freedom each for the points S_i and Z. Therefore, there are $8n + 2$ degrees of freedom. On the other hand, there are 7 constraints across each interior boundary. Furthermore, a *planarity constraint* at Z must be satisfied: all the points P_i (and, of course, Z) must lie in the same plane. This introduces $n - 3$ more constraints. Thus, there are $8n - 3$ constraints. We can see that there are always 5 degrees of freedom more than there are constraints. Therefore, in theory, *a surface over an arbitrary convex face with* n *sides can be represented as a union of* n *geometrically continuous triangular Bézier patches of degree 4.*

However, it turns out that the planarity condition complicates the system of equations to such an extent, that so far we were not able to find a closed-form solution. For the triangular case, the planarity is always satisfied (three points are always on the same plane!), which leads to the system of vector equations discussed above. The planarity constraint, on the other hand,

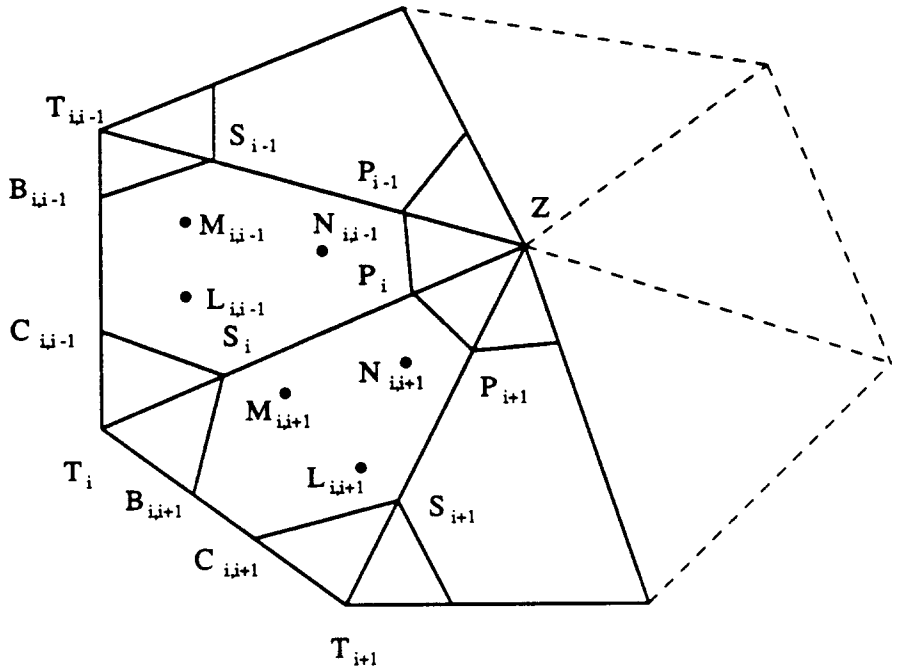


Figure 7.2: *Subdivision of the surface over an arbitrary face.*

mixes all the coordinates, so that the equations can no longer be broken into three separate subsystems. The area ratio constraint also becomes more complicated. Even for the quadrilateral case it is not, in general, possible to pick the center point Z, so that all the areas of triangles P_iZP_{i+1} can be made equal.

A special case arises, when symmetry constraints can be used to fulfill the planarity conditions automatically.

7.2.3. Subdivision of a Regular Polygon

Although we were not able to subdivide an arbitrary n-gon into n geometrically continuous triangular Bézier patches, it is not hard to perform such subdivision for a perfectly symmetrical situation. There are many practical applications of this particular case, such as the representation of a sphere starting with a dodecahedron as the initial object, or the representation of the top of a rocket with a "smoothed" cone.

In this case, one can assume that the control points $B_{i,i+1}$ and $C_{i,i+1}$ are symmetric with respect to the symmetry plane between points T_i and T_{i+1} (Fig. 7.2). Points S_i will then be chosen to lie in the plane, defined by T_i and the rotational symmetry axis through Z. This will ensure that the control points $L_{i,i+1}$ and $M_{i,i+1}$ are also symmetric with respect to the above plane and the above four control points are located in the same position with respect to each triangle. In other words, the n-gon with its vertices and control points $B_{i,i+1}$, $C_{i,i+1}$, $L_{i,i+1}$, $M_{i,i+1}$, and $N_{i,i+1}$

is invariant under the symmetry group D_n of the regular n-gon.

Due to the symmetry of the polygon, the complexity of the system of equations for this case is reduced dramatically, and all the control points can be constructed solely considering a pair of adjacent triangles (Fig. 7.3a).

Let T be the center of the original regular polygon:

$$\mathbf{T} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{T}_i. \tag{7.14}$$

Consider a pair of isosceles triangles $T_{i-1}T\ T_i$ and $T_iT\ T_{i+1}$. The angle ϕ at T is $\frac{2\pi}{n}$ for both these triangles. The point T can be expressed as follows in barycentric coordinates of T_{i-1}, T_i, and T_{i+1}:

$$\mathbf{T} = \alpha \mathbf{T}_{i-1} + \alpha \mathbf{T}_{i+1} + (1 - 2\alpha)\, \mathbf{T}_i \tag{7.15}$$

for some number α. Let K be the midpoint of T_{i-1} and T_{i+1} (Fig. 7.3b). Then the previous equation can be rewritten as

$$\mathbf{T} = 2\alpha\, \mathbf{K} + (1 - 2\alpha)\, \mathbf{T}_i. \tag{7.16}$$

Since

$$\frac{KT}{T_iT} = \frac{T_iT\ \cos(\phi)}{T_iT} = \cos(\phi),$$

then

$$\frac{1 - 2\alpha}{2\alpha} = -\cos(\phi)$$

and

$$\alpha = \frac{1}{2\,(1 - \cos(\phi))}. \tag{7.17}$$

If we set

$$\beta = 2\cos(\phi), \tag{7.18}$$

then

$$\mathbf{T} = \frac{1}{2 - \beta}\,(\mathbf{T}_{i-1} + \mathbf{T}_{i+1} - \beta\, \mathbf{T}_i). \tag{7.19}$$

Analogously,

$$\mathbf{Z} = \frac{1}{2 - \beta}\,(\mathbf{P}_{i-1} + \mathbf{P}_{i+1} - \beta\, \mathbf{P}_i), \tag{7.20}$$

where, as usual, Z is the center of P_i's:

$$\mathbf{Z} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{P}_i. \tag{7.21}$$

Note that the picture in Fig. 7.3 is a projection to the plane of the original polygon and thus point Z is projected onto T.

As in Section 7.2.1 we try to make a reasonable choice for S_i; because of the symmetry of the situation it is natural to make the triangles $T_i T\ T_{i+1}$ and $T_i S_i C_{i,i+1}$ similar:

$$\mathbf{S}_i = \frac{1}{2-\beta}\,(\mathbf{B}_{i-1,i} + \mathbf{C}_{i,i+1} - \beta\,\mathbf{T}_i). \tag{7.22}$$

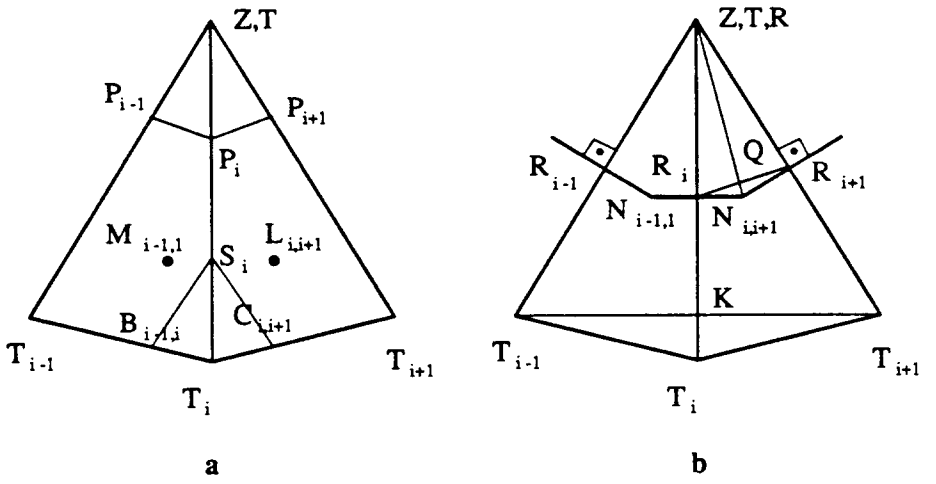


Figure 7.3:
a. *Control points for the regular polygon.* **b.** *Some auxiliary constructions.*

The above equations can be rewritten as follows:

$$\mathbf{C}_{i,i+1} = \beta\,\mathbf{T}_i + (2-\beta)\,\mathbf{S}_i - \mathbf{B}_{i-1,i}, \tag{7.23}$$

$$\mathbf{P}_{i+1} = \beta\,\mathbf{P}_i + (2-\beta)\,\mathbf{Z} - \mathbf{P}_{i-1}. \tag{7.24}$$

In the spirit of equations (7.5) from Section 7.2.1,

$$\mathbf{C}^q_{i,i+1} = \beta_1\,\mathbf{T}_i + (2-\beta_1)\,\mathbf{S}_i - \mathbf{B}^q_{i-1,i}, \tag{7.25}$$

$$\mathbf{P}^q_{i+1} = \beta_2\,\mathbf{P}_i + (2-\beta_2)\,\mathbf{Z} - \mathbf{P}^q_{i-1}, \tag{7.26}$$

where again the superscript q refers to quartic control points on the boundaries, and coefficients β_1 and β_2 are related to β as follows:

$$\beta_1 = \frac{1}{2} + \frac{3}{4}\,\beta, \tag{7.27}$$

$$\beta_2 = \frac{3}{4}\,\beta. \tag{7.28}$$

Now we are ready to use Farin's equations (7.8) from Section 7.2.1:

$$\mathbf{L}_{i,i+1} = \frac{2}{3}\,(\beta_1\,\mathbf{S}_i + (2-\beta_1)\,\mathbf{P}_i) + \frac{1}{3}\,(\beta_2\,\mathbf{T}_i + (2-\beta_2)\,\mathbf{S}_i) - \mathbf{M}_{i-1,i}, \tag{7.29}$$

$$\mathbf{N}_{i,i+1} = \frac{1}{3}(\beta_1 \mathbf{P}_i + (2-\beta_1)\mathbf{Z}) + \frac{2}{3}(\beta_2 \mathbf{S}_i + (2-\beta_2)\mathbf{P}_i) - \mathbf{N}_{i-1,i}. \tag{7.30}$$

The points P_i can therefore be expressed as follows:

$$\mathbf{P}_i = \frac{1}{2(2-\beta)}(4(\mathbf{M}_{i-1,i} + \mathbf{L}_{i,i+1}) - (4+\beta)\mathbf{S}_i - \beta \mathbf{T}_i). \tag{7.31}$$

Notice that we can rewrite the equation for $N_{i,i+1}$ as

$$\mathbf{N}_{i,i+1} = 2\mathbf{R}_i - \mathbf{N}_{i-1,i}, \quad i = 1,\ldots,n, \tag{7.32}$$

introducing auxiliary points R_i:

$$\mathbf{R}_i = \frac{1}{8}((6-\beta)\mathbf{P}_i + 2\beta \mathbf{S}_i + (2-\beta)\mathbf{Z}). \tag{7.33}$$

We are now at the stage where in Section 7.2.1 we had to solve a system of n linear equations to determine the n control points $N_{i,i+1}$. However, because of the symmetry of the regular n-gon, these points can be found more easily. First, they lie in a common plane parallel to the plane of the T_i's. Second, each $N_{i,i+1}$ lies symmetrically with respect to the symmetry plane between T_i and T_{i+1}. Using the auxiliary points R_i, which are the midpoints between two adjacent $N_{i-1,i}$ and $N_{i,i+1}$, we can readily find an expression for the control points.

We wish to express the point $N_{i,i+1}$ in barycentric coordinates of R_i, R_{i+1} and R, where R is the center of all R_i's:

$$\mathbf{R} = \frac{1}{n}\sum_{i=0}^{n} \mathbf{R}_i. \tag{7.34}$$

So,

$$\mathbf{N}_{i,i+1} = \gamma \mathbf{R}_i + \gamma \mathbf{R}_{i+1} + (1 - 2\gamma \mathbf{R}). \tag{7.35}$$

Let Q be the midpoint of R_i and R_{i+1} (Fig. 7.3b). Then

$$\frac{N_{i,i+1}Q}{N_{i,i+1}R} = \frac{N_{i,i+1}R_i \sin(\phi/2)}{N_{i,i+1}R_i/\sin(\phi/2)} = \sin^2(\phi/2)$$

and therefore

$$\frac{1-2\gamma}{2\gamma} = -\sin^2(\phi/2).$$

It follows that

$$\gamma = \frac{1}{1+\cos(\phi)} \tag{7.36}$$

or, in terms of eq. (7.17),

$$\gamma = \frac{2}{2+\beta}. \tag{7.37}$$

Now, from equations (7.35) we can finally obtain an expression for the control points $N_{i,i+1}$:

$$\mathbf{N}_{i,i+1} = \frac{1}{4(\beta+2)} \left((6-\beta)(\mathbf{P}_i + \mathbf{P}_{i+1}) + 2\beta(\mathbf{S}_i + \mathbf{S}_{i+1}) - (\beta^2-4\beta+4)\mathbf{Z} + (\beta^2-2\beta)\mathbf{S}\right), \quad (7.38)$$

where S is the center of the S_i's:

$$\mathbf{S} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{S}_i. \quad (7.39)$$

7.3. Quadrilateral Patch Subdivision

7.3.1. Preliminaries

As we have seen before, for the most general case, an alternative way of subdivision must be found. Since the planarity constraint is the main obstacle in obtaining an easy subdivision, it should be avoided. This can be achieved if no more than *three* interior boundaries meet at any point.

We have arrived at two subdivision schemes for the quadrilateral case (Fig. 7.4). Subdivision 'a' produces the fewest number of patches, while subdivision 'b' produces one extra patch and requires somewhat more computation. However, it is more 'symmetric' and treats all the vertices and sides of the quadrilateral in the same way.

We use an approach equivalent to that of the triangular patch subdivision to obtain all the necessary control points of the interior boundaries and subpatches. Again, the interior control points $L_{i,i+1}$ and $M_{i,i+1}$ of the subpatches, closest to the boundary of the original patch can readily be found by Chiyokura's method once the control points S_i of the interior boundaries have been chosen. The second control points of these boundaries, P_i, can then be determined by Farin's formulas if the area constraint is taken care of. The two control points, Q_{ij} and V_{ij}, on the one innermost boundary of subdivision 'a' or on the four boundaries in subdivision 'b', as well as the central points Z_i of the triangles formed by the control points of the interior boundaries are then chosen in such a way, that the area ratio constraints are satisfied. Then the remaining control points of the subpatches $N_{i,i+1}$ can be found from two (subdivision 'a') or four separate systems (subdivision 'b') of three vector equations.

Just as we did for the triangular subdivision, we are now analyzing the number of available degrees of freedom that the two subdivision schemes have.

For the scheme 'a', the points P_i, N_{ij}, and Q_i have 3, and S_i and Z_i have 2 degrees of freedom each. This gives a total of 48. On the other hand, there are 5 interior boundaries and therefore 35 constraints. Thus, there are 13 degrees of freedom available for this method.

For the scheme 'b', again the points P_i, N_{ij}, Q_{ij}, and V_{ij} have 3 degrees of freedom each, while S_i and Z_i have only 2 for the total of 88. Eight interior boundaries give 56 constrains, which leaves 32 degrees of freedom in the system.

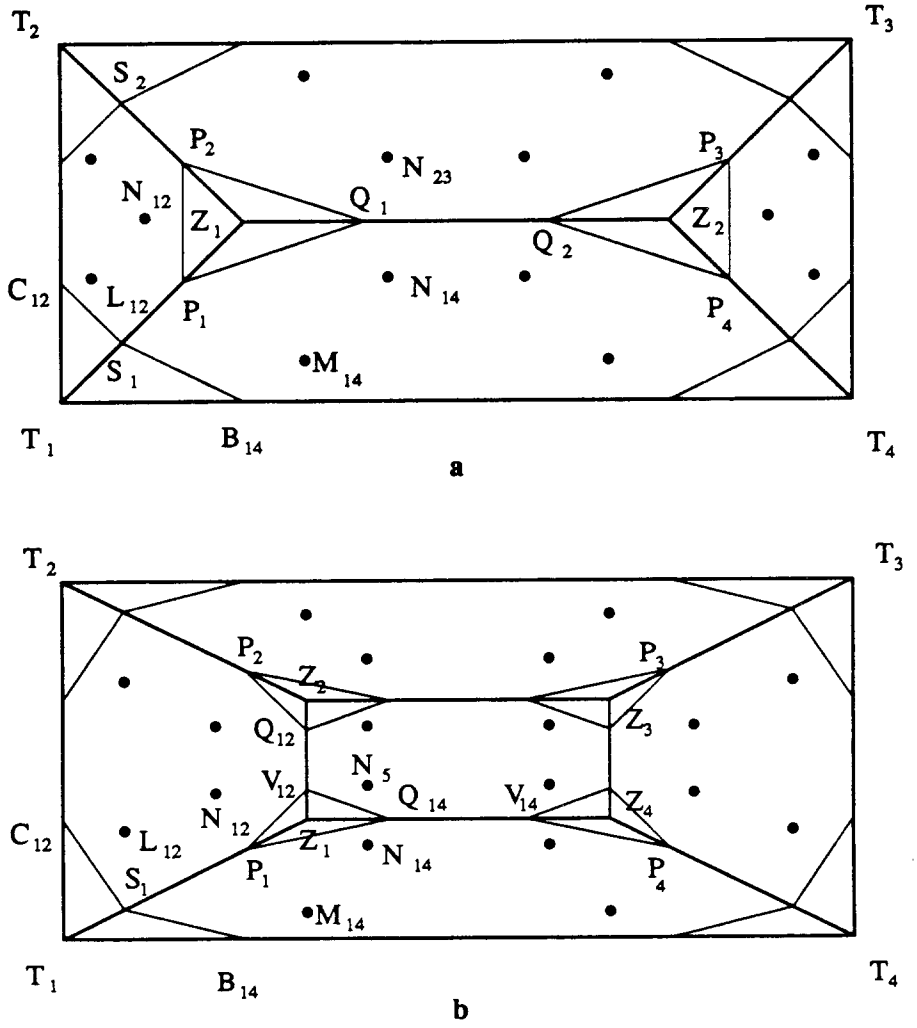


Figure 7.4: *Two ways of quadrilateral patch subdivision.*

The above analysis shows that in both schemes many control points can be chosen freely. However, a reasonable choice should be made. As we did with the triangular subdivision, we consider the case of the *symmetrical square patch* and make our selections based on this special case. Again, we assume that the control points of the outer boundaries divide them into three equal parts. For the subdivision 'a' we assume the line segment Z_1Z_2 to pass through the center of the square, to be of one third of the side length, and to be parallel to two sides of the square. For the subdivision 'b', $Z_1Z_2Z_3Z_4$ is a square with the side length equal to one third of the side length of the big square. The control points S_i divide the corresponding boundary in the ratio 1:2, if the corresponding control points on the exterior boundary divide it into three equal parts. Furthermore, in this symmetric case, Q_{ij} and V_{ij} should also divide the corresponding boundary into three equal parts (Fig. 7.5).

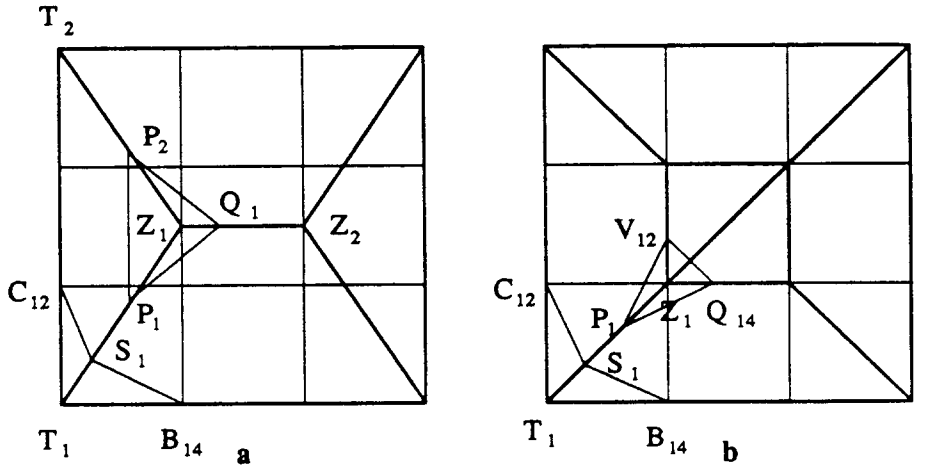


Figure 7.5: *Location of the control points for the regular square.*

7.3.2. Determination of the Control Points

7.3.2.1. Subdivision 'a'

From the discussion about the regular square it follows that the ratio of influences of the points C_{12} and B_{14} to S_1 should be 2:3, i.e.

$$\mathbf{S}_1 = \frac{3\,\mathbf{C}_{12} + 2\,\mathbf{B}_{14} - 3\gamma_1\,\mathbf{T}_1}{5 - 3\gamma_1} \tag{7.40}$$

or

$$\mathbf{C}_{12} = \gamma_1 \mathbf{T}_1 + (\frac{5}{3} - \gamma_1)\,\mathbf{S}_1 - \frac{2}{3}\,\mathbf{B}_{14} \tag{7.41}$$

for some number γ_1.

Since quartic control points have to be used for the boundary of the triangle T_1T_2Z and cubic control points for the boundary of the quadrilateral $T_1Z_1Z_2T_4$, then the above equation can be rewritten as follows in the spirit of (7.5):

$$\mathbf{C}_{12}^{q} = \alpha_1 \mathbf{T}_1 + (\frac{3}{2} - \alpha_1)\,\mathbf{S}_1 - \frac{1}{2}\,\mathbf{B}_{14}, \tag{7.42}$$

where

$$\alpha_1 = \frac{1}{4} + \frac{3}{4}\,\gamma_1. \tag{7.43}$$

Analogous equations hold for the other S_i; therefore, each point S_i has one degree of freedom. This reduces the total number of degrees of freedom for this scheme from 13 to 9.

Now, the point P_2^g has to be expressed in terms of P_1, Z_1, and Q_1. To satisfy the area ratio requirement, the coefficient of Q_1 should be equal to 1/2 (which is the coefficient of B_{14} in the previous equation, see Section 7.2.1):

$$\mathbf{P}_2^g = \alpha_3 \mathbf{P}_1 + (\frac{3}{2} - \alpha_1)\, \mathbf{Z}_1 - \frac{1}{2}\, \mathbf{Q}_1. \tag{7.44}$$

Since Z_1 should be symmetric with respect to P_1 and P_2, then α_3 must be equal to $-3/4$, so that

$$\mathbf{Z}_1 = \frac{3}{8}\, \mathbf{P}_1 + \frac{3}{8}\, \mathbf{P}_2 + \frac{2}{8}\, \mathbf{Q}_1. \tag{7.45}$$

This choice of Z_1 and Z_2 reduces the number of degrees of freedom by 4 to 5. However, at the same time, the area ratio constraint across all the 5 interior boundaries will be automatically satisfied, and in fact 10 degrees of freedom remain. Out of these, 6 can be used to pick the points Q_1 and Q_2 (Section 7.3.1); the remaining degrees of freedom can be used to select α_1 for the 4 inner boundaries starting at the T_i's.

Solving Farin's equations across the 3 interior boundaries T_1Z_1, T_2Z_1, and Z_1Z_2 results in the following expressions:

$$\mathbf{P}_1 = \frac{1}{4\,(3-2\alpha_1)}\, (12\, \mathbf{L}_{12} + 6\, \mathbf{M}_{14} - (8\, \alpha_1 + 9)\, \mathbf{S}_1 + 3\, \mathbf{T}_1), \tag{7.46}$$

$$\mathbf{N}_{12} = \frac{1}{432}\, (-108\, \mathbf{S}_1 - 108\, \mathbf{S}_2 + (48\alpha_1 - 32\beta_1 + 417)\, \mathbf{P}_1 + (-32\alpha_1 + 48\beta_1 + 417)\, \mathbf{P}_2 -$$

$$(16\alpha_1 + 16\beta_1 + 234)\, \mathbf{Q}_1 + 48\, \mathbf{Q}_2),$$

$$\mathbf{N}_{14} = \frac{1}{216}\, (-108\, \mathbf{S}_1 + 108\, \mathbf{S}_2 + (48\alpha_1 + 32\beta_1 + 303)\, \mathbf{P}_1 - (32\alpha_1 + 48\beta_1 + 321)\, \mathbf{P}_2 + \tag{7.47}$$

$$(-16\alpha_1 + 16\beta_1 + 282)\, \mathbf{Q}_1 - 48\, \mathbf{Q}_2),$$

$$\mathbf{N}_{23} = \frac{1}{216}\, (108\, \mathbf{S}_1 - 108\, \mathbf{S}_2 - (48\alpha_1 + 32\beta_1 + 321)\, \mathbf{P}_1 + (32\alpha_1 + 48\beta_1 + 303)\, \mathbf{P}_2 +$$

$$(16\alpha_1 - 16\beta_1 + 282)\, \mathbf{Q}_1 - 48\, \mathbf{Q}_2)$$

and analogously for the other control points. Note that α_1 refers to the boundary T_1Z_1, and β_1 to the boundary T_2Z_1.

Just as we chose the control points in the centers of gravity of the corresponding triangles in the triangular subdivision, we can pursue the regular square analogy further and set α_1 and β_1 to 0. In this case, the equations are simplified:

$$\mathbf{P}_1 = \frac{1}{4}\, (4\, \mathbf{L}_{12} + 2\, \mathbf{M}_{14} - 3\, \mathbf{S}_1 + \mathbf{T}_1), \tag{7.48}$$

$$\mathbf{N}_{12} = \frac{1}{144}\, (-36\, \mathbf{S}_1 - 36\, \mathbf{S}_2 + 16\, \mathbf{Q}_2 + 139\, \mathbf{P}_1 + 139\, \mathbf{P}_2 - 78\, \mathbf{Q}_1),$$

$$\mathbf{N}_{14} = \frac{1}{72}(-36\,\mathbf{S}_1 + 36\,\mathbf{S}_2 - 16\,\mathbf{Q}_2 + 101\,\mathbf{P}_1 - 107\,\mathbf{P}_2 + 94\,\mathbf{Q}_1), \tag{7.49}$$

$$\mathbf{N}_{23} = \frac{1}{72}(36\,\mathbf{S}_1 - 36\,\mathbf{S}_2 - 16\,\mathbf{Q}_2 - 107\,\mathbf{P}_1 + 101\,\mathbf{P}_2 + 94\,\mathbf{Q}_1),$$

and analogously for the other control points.

7.3.2.2. Subdivision 'b'

This subdivision scheme is symmetric in the sense that it treats all the vertices of the original face in the same way. It is natural to assume that the interior control points are also symmetric with respect to other control points. Therefore, S_i and Z_i should be on the medians of the corresponding triangles. For S_1 and Z_1 these conditions can be expressed as follows:

$$\mathbf{C}_{12} = \alpha_1\mathbf{T}_1 + (2-\alpha_1)\,\mathbf{S}_1 - \mathbf{B}_{14}, \tag{7.50}$$

$$\mathbf{V}_{12} = \alpha_3\mathbf{P}_1 + (2-\alpha_3)\,\mathbf{Z}_1 - \mathbf{Q}_{14}. \tag{7.51}$$

While α_1 can be chosen independently for each interior boundary T_iZ_i, α_3 should be the same for all 4 interior triangles to enforce area ratio constraint. Therefore, the total number of degrees of freedom should be reduced by 4+1+2+2+2 = 11 to 21. However, all 8 ratio constraints will be satisfied, raising the total number of degrees of freedom to 29. Out of these, 24 will be used to choose the remaining 8 control points Q_{ij} and V_{ij} (Section 7.3.1). The remaining 5 are determined by 4 α_1's and a α_3.

Solving Farin's equations across the boundaries T_1Z_1, Z_1Z_2, and Z_1Z_4 yields:

$$\mathbf{P}_1 = \frac{1}{2\,(2-\alpha_1)}(3\,\mathbf{L}_{12} + 3\,\mathbf{M}_{14} + (\alpha_3 - 2\,\alpha_1 - 2)\,\mathbf{S}_1 - \alpha_3\,\mathbf{T}_1), \tag{7.52}$$

$$\mathbf{N}_{12} = \frac{1}{6\alpha_3\,(2-\alpha_3)}(-(2\alpha_3^3 - 4\alpha_3^2)\,\mathbf{S}_1 - (2\alpha_3 - 4)\,\mathbf{Q}_{12} + (2\alpha_3 - 4)\,\mathbf{V}_{14} +$$

$$(2\alpha_3^3 - 10\alpha_3^2 + (2\alpha_1 + 8)\,\alpha_3)\,\mathbf{P}_1 - (3\alpha_3^2 + (\alpha_1 - 14)\,\alpha_3 + 12)\,\mathbf{V}_{12} + (3\alpha_3^2 - (\alpha_1 + 10)\,\alpha_3 + 12)\,\mathbf{Q}_{14}),$$

$$\mathbf{N}_{14} = \frac{1}{6\alpha_3\,(2-\alpha_3)}(-(2\alpha_3^3 - 4\alpha_3^2)\,\mathbf{S}_1 + (2\alpha_3 - 4)\,\mathbf{Q}_{12} - (2\alpha_3 - 4)\,\mathbf{V}_{14} + \tag{7.53}$$

$$(2\alpha_3^3 - 10\alpha_3^2 + (2\alpha_1 + 8)\,\alpha_3)\,\mathbf{P}_1 + (3\alpha_3^2 - (\alpha_1 + 10)\,\alpha_3 + 12)\,\mathbf{V}_{12} - (3\alpha_3^2 + (\alpha_1 - 10)\,\alpha_3 + 12)\,\mathbf{Q}_{14}),$$

$$\mathbf{N}_5 = \frac{1}{12 - 6\alpha_3}(-(2\alpha_3^3 - 4\alpha_3^2)\,\mathbf{S}_1 + (2\alpha_3 - 4)\,\mathbf{Q}_{12} + (2\alpha_3 - 4)\,\mathbf{V}_{14} +$$

$$(2\alpha_3^3 - 10\alpha_3^2 + (2\alpha_1 + 10)\,\alpha_3)\,\mathbf{P}_1 + (3\alpha_3^2 - (\alpha_1 + 10)\,\alpha_3 + 10)\,\mathbf{V}_{12} + (3\alpha_3^2 - (\alpha_1 + 10)\,\alpha_3 + 10)\,\mathbf{Q}_{14}).$$

Finally, if each S_i and Z_i is chosen in the center of the corresponding triangle, then the parameters α_1 and α_3 are set to -1 and

$$\mathbf{P}_1 = \frac{1}{6}(3\,\mathbf{L}_{12} + 3\,\mathbf{M}_{14} - \mathbf{S}_1 + \mathbf{T}_1), \tag{7.54}$$

$$\mathbf{N}_{12} = \frac{1}{3}(-\mathbf{S}_1 - \mathbf{Q}_{12} + \mathbf{V}_{14} + 3\,\mathbf{P}_1 + 5\,\mathbf{V}_{12} - 4\,\mathbf{Q}_{14}),$$

$$\mathbf{N}_{14} = \frac{1}{3}(-\mathbf{S}_1 + \mathbf{Q}_{12} - \mathbf{V}_{14} + 3\,\mathbf{P}_1 - 4\,\mathbf{V}_{12} + 5\,\mathbf{Q}_{14}), \tag{7.55}$$

$$\mathbf{N}_5 = \frac{1}{9}(3\,\mathbf{S}_1 - 3\,\mathbf{Q}_{12} - 3\,\mathbf{V}_{14} - 10\,\mathbf{P}_1 + 11\,\mathbf{V}_{12} + 11\,\mathbf{Q}_{14}),$$

and analogously for the other control points.

7.3.3. Selection of Control Points on Interior Boundaries

An analysis of the number of degrees of freedom, made in the previous Section, shows that the points Q_1 and Q_2 for the subdivision 'a', and the points Q_{ij} and V_{ij} for the subdivision 'b' can be chosen freely without destroying G^1 continuity across interior boundaries. However, a reasonable choice must be made. Improper selection of these points can easily result in waves or spikes on the surface.

In this Section, we describe a way to choose the control points so that the undesired effects are avoided. The main idea is to construct Bézier patches so that they would closely approximate a Gregory patch that would have been constructed as described in Chapter 6. The justification for this is the fact that interpolation with Gregory patches generally gives good results; one would like to combine the smoothness it produces with the piecewise polynomial surface given by Bézier patches.

We have tested two general approaches to the problem on the subdivision 'b'. In the first approach, available degrees of freedom were used to ensure that interior network joints (points Z_i) lie on the surface of the Gregory patch. In the second approach, normals at Z_i's were required to coincide with corresponding normals of the Gregory patch. Thus, interpolation of points and normals at 4 joints and 4 patch corners were compared. As a number of test cases clearly indicated, interpolation of the normals consistently produced more pleasing, smooth surfaces compared to point interpolation.

We now describe in detail the process of selection of the 4 interior joints Z_i and 8 control points Q_{ij} and V_{ij}. At present, this method is also implemented with good results in Ricoh Corp.'s DESIGNBASE modeling system [26].

In this scheme, the normals at Z_i are required to coincide with the normals $\mathbf{n}_i$ of the Gregory patch at parametric values (1/3,1/3), (1/3,2/3), (2/3,2/3), and (2/3,1/3) respectively. The computation proceeds as follows:

1. Compute the Gregory patch as described in Chapter 6. Evaluate it at 4 center points (with above parametric values) and compute its normals there, yielding 4 points Z_i^0 and 4 normals $\mathbf{n}_i$.

2. Compute the points P_i according to (7.52) or (7.54).

3. Select the joints Z_i on a line parallel to $\mathbf{n}_i$ through Z_i^0, so that the line segment $P_i Z_i$ lies in a plane with the normal $\mathbf{n}_i$ (Fig. 7.6):

$$\mathbf{Z}_i = \mathbf{Z}_i^0 - ((\mathbf{P}_i - \mathbf{Z}_i^0) \cdot \mathbf{n}_i)\, \mathbf{n}_i. \tag{7.56}$$

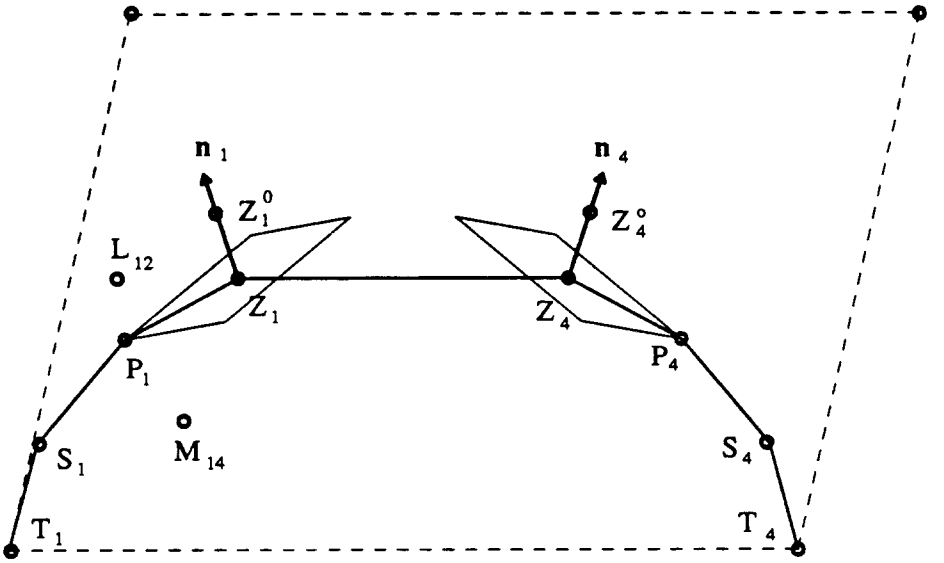


Figure 7.6: *Computation of interior control points.*

4. At this stage, the 4 center points Z_i can be viewed as additional vertices with prescribed normals $\mathbf{n}_i$ that need to be interpolated. Thus, any of the procedures from Chapter 5 to determine boundary curves can be applied here. Assume that the resulting control points of interior boundaries are Q_{ij}^0 and V_{ij}^0.

5. To guarantee G^1 continuity across interior boundaries, equations (7.50) and (7.51) need to be satisfied. Therefore, the control points are adjusted as follows:

$$\mathbf{Q}_{14} = \frac{1}{2}\,((2-\alpha_3)\,\mathbf{Z}_1 + \alpha_3\,\mathbf{P}_1 + \mathbf{Q}_{14}^0 - \mathbf{V}_{12}^0), \tag{7.57}$$

$$\mathbf{V}_{12} = \frac{1}{2}\,((2-\alpha_3)\,\mathbf{Z}_1 + \alpha_3\,\mathbf{P}_1 - \mathbf{Q}_{14}^0 + \mathbf{V}_{12}^0).$$

If Z_i are chosen in the center of gravity of corresponding triangles, then $\alpha_3 = -1$ and

$$\mathbf{Q}_{14} = \frac{1}{2}\,(3\,\mathbf{Z}_1 - \mathbf{P}_1 + \mathbf{Q}_{14}^0 - \mathbf{V}_{12}^0), \tag{7.58}$$

$$\mathbf{V}_{12} = \frac{1}{2}\,(3\,\mathbf{Z}_1 - \mathbf{P}_1 - \mathbf{Q}_{14}^0 + \mathbf{V}_{12}^0).$$

6. Finally, the remaining interior control points of the Bézier patches are computed according to formulas (7.53) or (7.55).

The same idea of 'closeness' to the Gregory patch can be applied to the subdivision 'a'. In this case, 2 points Z_1^0 and Z_2^0 and 2 normals $\mathbf{n}_1$ and $\mathbf{n}_2$ will correspond to the points and normals of the Gregory patch evaluated at the parametric values (1/2, 1/3) and (1/2, 2/3). Z_1 will then be located on a line parallel to $\mathbf{n}_1$ through Z_1^0 so that the angle between $\mathbf{n}_1$ and the normal to $P_1P_2Z_1$ is minimized; analogously for Z_2. To satisfy the area ratio constraint, Q_i can be determined from (7.45). Finally, the interior control points of the Bézier patches can be found from (7.47) or (7.49).

We believe, however, that subdivision 'b' is more practical. Unlike subdivision 'a', it does not mix triangular and quadrilateral patches. More importantly, it is symmetric in the sense that the application does not have to make arbitrary guesses in which direction to split a patch.

7.4. Results

The described subdivision schemes for triangular and quadrilateral patches have been tested on a number of examples. These examples include the test polyhedrons and meshes used for constructing boundary curves (Chapter 5), as well as pairs of quadrilateral and triangular patches used for illustrating the shape parameters (Chapter 6). Fig. 7.7 shows a pair of warped triangular and quadrilateral patches with the interior boundaries between the Bézier patches. Fig. 7.8 shows an interpolated icosahedron (a) and interpolated cube (b) with the net of isoparametric lines or each constructed Bézier patch.

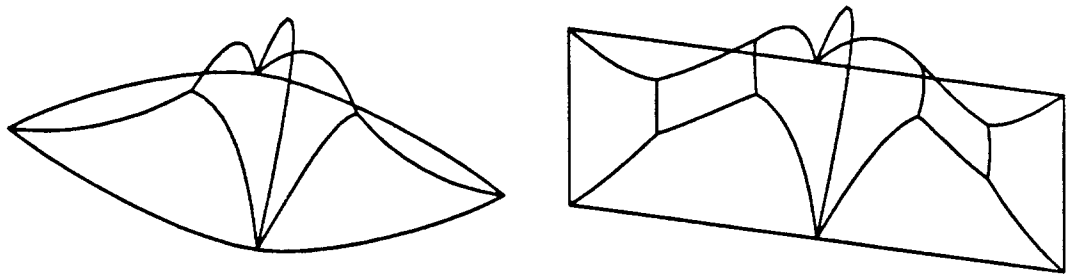
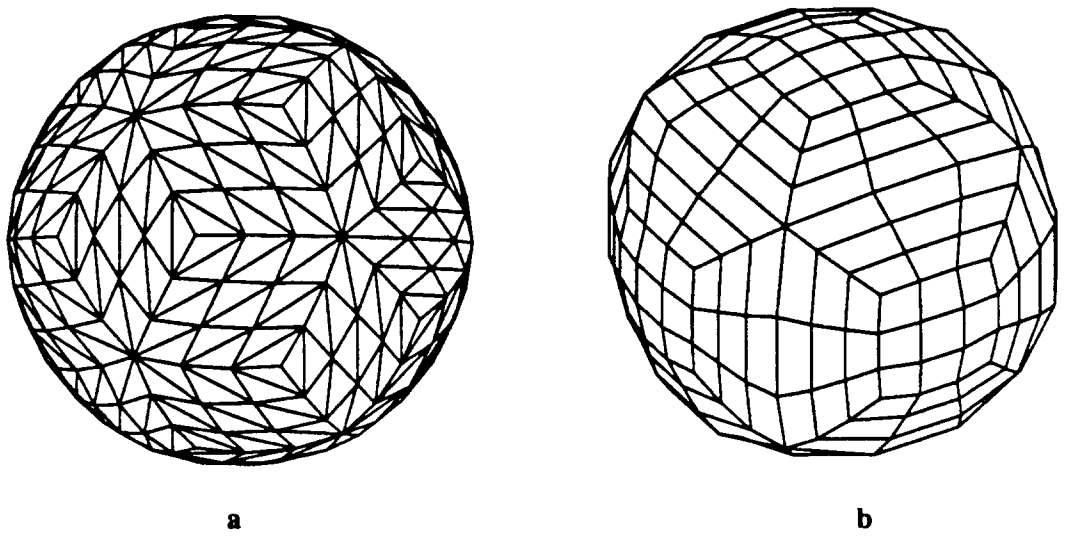


Figure 7.7: *A pair of warped triangular and quadrilateral patches with interior boundaries.*

The union of 3 or 5 Bézier patches interpolates the boundaries and the tangent planes at the vertices of the original triangular or quadrilateral Gregory patch. It is clear, however, that the surface composed of Bézier patches will differ from the surface composed of Gregory patches. This difference turns out to be quite small. For example, for the interpolated cube (Fig. 7.8b), the newly generated vertices lie 0.8% below the surface of the original Gregory patch. For the case of the interpolated icosahedron (Fig. 7.8a), the new vertices actually lie on the surface of the

Gregory patch.



a b

Figure 7.8: a. *Interpolated icosahedron.* **b.** *Interpolated cube.*

In general, it is difficult to calculate the deviation between the Bézier and Gregory patches due to the differences in their parametrizations. However, for the measure of smoothness of the Bézier surface, this deviation is not as important as the curvature variation across the internal boundaries. We use the tools from Chapter 5 to estimate the curvature variation between the adjacent Bézier subpatches.

Table 7.1 summarizes the difference in curvature variations between exterior and interior boundaries of the Bézier patches for the interpolated icosahedron and cube, as well as for several polyhedral models from Section 5.6.1. The four different measures used in the comparison are the same as in Section 5.6.1: maximum curvature value on all boundaries, maximum difference between these values on either boundary side, maximum integral $\int |\kappa_1 - \kappa_2| ds$ of the above differences, and, finally, the sum of these integrals.

As we see from this table, curvature variation across the internal boundaries is small compared to the variation across the external boundaries. The only exception to that is the interpolated cube, which is G^2 across external boundaries due to the symmetry. The interpolated icosahedron is G^2 across *all* boundaries also because of its symmetry. Thus, there is no penalty in overall surface smoothness for using the Bézier representation.

Boundary Type	Polyhedron					
	Icosahedron	Cube	Sphere	Torus	Franke surface	Random triang. mesh
Exterior	2.2	2.4	5.8	1.8	25.3	4.9
	0.0	0.0	3.8	1.8	14.7	3.6
	0.0	0.0	1.8	16.5	4.2	6.4
	0.0	0.0	65.8	131.8	54.6	73.9
Interior	2.2	1.7	5.8	0.7	38.9	8.6
	0.0	0.2	2.1	0.4	15.6	3.5
	0.0	0.1	0.4	1.7	1.0	4.3
	0.0	0.8	8.2	13.9	3.7	15.3

Table 7.1: *Curvature variation across different exterior and interior boundaries.*

7.5. Summary

The described procedure for filling a given mesh of cubic boundaries with Bézier patches is simple and efficient. It combines the advantages of the polynomial Bézier patches used in Farin's method with the locality of Chiyokura's approach. The triangular patch subdivision and the subdivision of a quadrilateral face into five quadrilateral patches (Fig. 7.4b) have been implemented in UNICUBIX and produce satisfactory results.

The system of Bézier patches that fit with G^1 continuity into a given frame of 3 or 4 cubic curves has several degrees of freedom. We have made some plausible assumptions how to use these degrees of freedom to come up with a simple sequence of computations to determine all the necessary control points. As a result, all control points of the subdivision boundaries become linear combinations of the originally given vertices and the control points of the cubic boundaries between them.

8

UniCubix: an Experimental Modeling System

In this chapter, an overview of an experimental modeling system UNICUBIX is given. This system was used for testing the methods for curve and surface construction described in the previous Chapters. First, a conceptual view of the system is given and each of the procedural steps is described. Second, a brief user guide is presented. Third, implementation details are discussed. Finally, hints for future enhancements of the system are suggested.

8.1. Conceptual Overview of UniCubix

In order to test various methods for generation of curves and surfaces, an experimental modeling system, UNICUBIX, was developed. It is a direct extension of Berkeley UNIGRAFIX modeling and rendering system [119]. In UNIGRAFIX, polyhedral objects are described by vertices positioned in space and by linear edges and polyhedral faces joining these vertices. UNICUBIX generalizes this paradigm by selectively replacing edges by boundary curves and faces by surface patches. The paradigm of the procedural, rule-based process of surface construction has been incorporated into the UNICUBIX system. As a result, although UNICUBIX is still an experimental system with rather limited capabilities, it has several unique features that are not available in traditional modeling systems.

The minimal information that must be provided to the system is a polyhedral approximation of the object. Rather than providing local rounding operations for individual edges [26, 77], UNICUBIX globally rounds the whole object, except for specifically marked edges or faces, eliminating the necessity of manually picking edges to be smoothed.

Another advantage of UNICUBIX is its flexibility. Many modeling systems can work exclusively with interpolating points, or with a curve mesh. There are no such restrictions in UNICUBIX; in fact, vertex normals and/or boundary curves can be added in a selective manner among the data points. If possible, the system will then adjust automatically to these constraints.

Finally, the available degrees of freedom are made available to the user in the form of shape parameters that have a clear interpretation of their effect on the shape of the surface. Thus, a designer can work with geometrically intuitive user-friendly handles rather than twiddling individual control points of surface patches [112].

UNICUBIX consists of two separate parts. The smaller part of the system is devoted to constructing G^1 and G^2 interpolating curves, while the bulk of UNICUBIX deals with G^1 interpolating surfaces.

8.1.1. Construction of Interpolating Curves

The starting point is a user-defined sequence of vertices to be interpolated. The system then computes vertex normals according to one of the methods in Chapter 3. Then, Bézier points are selected on defined tangent lines according to a specified velocity method. Finally, if G^2 curves are desired, each Bézier segment is split into two, and the corresponding control points are adjusted for curvature continuity (Chapter 4).

Below is an example specification of a rectangle in UNIGRAFIX / UNICUBIX format [120]:

```
v   A   0 0 0;
v   B   0 1 0;
v   C   5 1 0;
v   D   5 0 0;
w       (A B C D A);
```

The order of the points is defined by the above *wire* specification. The vertex A is present twice in it because a rectange is a closed polygon.

From the above description, UNICUBIX derives the G^1 interpolating curve by substituting each polygon edge with a cubic Bézier segment. This is achieved by selecting a normal direction and two velocities at each vertex of the defining polygon (Chapter 3). The actual method for choosing normals and velocities is specified by the user. Curvature continuous (G^2) curves can also be constructed by using two cubic segments per each polygon edge (Chapter 4).

8.1.2. Global Smoothing Procedures for Polyhedral Objects

The major part of UNICUBIX constructs interpolating surfaces through the vertices of supplied polyhedral objects. The starting point is a rough polyhedral approximation to the desired

curved surface consisting of vertices arranged into faces that determine its topology. Since triangular and quadrilateral patches are used, all faces in the defining mesh *must have either 3 or 4 vertices*. From this input the system derives the first smooth approximation by progressing through a sequence of procedural steps.

Defining Vertex Normals. A smooth surface interpolating a given set of points will have a certain tangent plane at each vertex. To define this tangent plane, a normal must be assigned to each vertex. UNICUBIX determines these normals by one of the methods described in Chapter 5. The actual method to be used is specified by the user.

It is also possible to predefine normals at several selected vertices. In this case, the system will not compute normals at these vertices, but will use the prescribed ones.

Since it is possible for a user to predefine whole boundary curves, some care must be taken to ensure that the computed normals and predefined curves are compatible to each other. Four cases must be considered:

1. Suppose that one boundary curve or two curves with the same tangent direction at a certain vertex are predefined. Any computed vertex normal will have to be orthogonal to this single tangent direction. If this is not the case, the vertex normal will be adjusted by projecting it onto the plane, perpendicular to the tangent direction.

2. If several boundary curves at a vertex are predefined that share the same tangent plane, this tangent plane will effectively define the vertex normal. Thus, the step of normal determination can be skipped in this case.

3. If predefined curves at a vertex *do not* share the same tangent plane, then no G^1 surface can be constructed. In this case, a warning will be issued and the default method for normal generation will be used. The tangent directions will be projected onto the tangent plane defined by the normal.

4. Similarly to the previous case, a warning will be issued if a predefined normal is not compatible to one or several predefined curves. Again, tangent directions will be projected onto the tangent plane defined by the prescribed normal.

Defining Curved Edges. With the vertex normals, and, therefore, tangent planes at all vertices defined, the original straight edges of the defining mesh are replaced by curved cubic edges by computing their Bézier points. The user specifies which of the methods for determination of tangent directions and velocities are to be used. Again, no computation will take place for those edges that are assigned predefined boundaries.

Defining Gregory or Bézier Patches. Next, UNICUBIX computes a single Gregory patch for each original face, using Chiyokura's method (Chapter 6) or several G^1 Bézier patches (Chapter 7). The user can also set shape parameters that would affect the shape of the surface near one or several boundaries.

8.1.3. Selective Smoothing

Since many objects are not overall smooth, UNICUBIX has to be able to specify edges that will remain straight, or that will show discontinuity of tangent planes, or faces that will remain flat after the global smoothing process. In UNICUBIX, this issue is addressed by having 4 different types of edges and 2 types of faces.

Edge Types. The following edge types are available in UNICUBIX:

1. *Curved Borders and Edges.* Unless explicitly defined, edges of this type will be replaced by system-generated Bézier curves.
2. *Straight Borders or Edges.* Edges of this type will remain straight lines between the two endpoints. The Bézier points will lie on the edge at a distance of 1/3 and 2/3 of the length of the edge from any of its endpoints.

The difference between borders and edges is that the G^1 continuity is enforced across borders only. Unlike borders, edges are visible as geometric features of the final object. The default edges type is *curved border.*

Face Types. There are 2 face types in UNICUBIX: *flat faces* and *patches.* If the face is flagged to be flat, it will remain flat even after the application of smoothing operations. Flat faces are treated as polygons: no interior control points are computed for them and no subdivision is necessary. Although flat faces may have curved edges, all the control points of these edges will be projected onto the plane of the face. The other type, patch, is the default; such faces will be replaced by Gregory or Bézier patches.

8.2. A Brief User Guide to UniCubix

An ideal setting for the modeling system would consist of an interactive editor, with which the user can manipulate the basic elements of the picture (vertices, normals, boundaries, etc). The designer would work with a mouse and a keyboard to specify and/or change various geometric parameters that define the surface. At present time, however, the implemented prototype for UNICUBIX, *uci*, has no mouse support and allows interaction through type-in commands only. A typical session, then, would consist of reading in a polyhedral description of the object, smoothing and viewing it and then possibly modifying its shape. We start with describing the format for the object description.

8.2.1. Description of Smooth Objects

Whether the curved object is created with a text editor or by an automated process, a format to capture and store the result is required [120]. For that purpose, the UNIGRAFIX descriptive language has been extended to include different types of edges and faces. Below are the added UNICUBIX statements for edge specifications:

curved border:	**bc**	$[ID]\,(v_1\ v_2\ b_{1x}\ b_{1y}\ b_{1z}\ b_{2x}\ b_{2y}\ b_{2z})$;
curved edge:	**ec**	$[ID]\,(v_1\ v_2\ b_{1x}\ b_{1y}\ b_{1z}\ b_{2x}\ b_{2y}\ b_{2z})$;
straight border:	**bl**	$[ID]\,(v_1\ v_2)$;
straight edge:	**be**	$[ID]\,(v_1\ v_2)$;

These statements are optional. Unless explicitly defined, an edge is assumed to be of the type curved border and its Bézier points are computed automatically by the system.

The UNIGRAFIX description of the faces remains essentially unchanged:

$$\mathbf{f}\ [ID]\,(v_1\ v_2\ \cdots\ v_n);$$

In this case, the face will be replaced by a surface patch. If, however, the identifier is replaced by F, then this face will remain flat.

Below is an example of a four-sided pyramid in UNICUBIX format:

```
v   Tip   0 0 4;
v   A     0 1 0;
v   B     1 0 0;
v   C     0 -1 0;
v   D     -1 0 0;
f   Bot   (A B C D);
f   a     (B A Tip)
f   b     (C B Tip)
f   c     (D C Tip)
f   d     (A D Tip)
```

Since there are no special conditions on edges and faces, this description is also valid in UNIGRAFIX. All edges will be replaced by curved borders and all faces by patches after a smoothing operation is performed. This would produce an egg-shaped surface.

If, however, a cone is to be produced from the above description, then the face **Bot** has to be flagged flat, and edges connecting the base of the pyramid with its tip have be declared straight:

```
F    Bot    (A B C D);
bl          (A Tip);
bl          (B Tip);
bl          (C Tip);
bl          (D Tip);
```

8.2.2. Reading in an Object Description

The above UNICUBIX specification of an object to be smoothed is usually contained in a file. So, the first action in *uci* after the system prompt *uc>* is displayed is to read in the file:

uc> smooth [-b <real>] [-n <int>] [-t <int>] [-v <int>] <filename> [xform options]

The **-b** option specifies the global bulge to be applied to all boundary curves. The options **-n**, **-t**, and **-v** select the normal, tangent, and velocity method respectively used for generating boundary curves that are not explicitly given. For valid numbers corresponding to various methods, and for description of transformation options, look at the **xform** command of the UNICUBIX manual (Appendix 2).

An alternative to **smooth** is **read** command, which only reads in polyhedral descriptions and does not create boundary curves. However, boundary curves can be specified explicitly in the UNICUBIX format.

8.2.3. Creating Interpolating Curves

We now describe the smaller part of the system that constructs an interpolating curve from a wire specification. This is achieved with the command **gcurve**:

uc> gcurve [-m <int> <int>] [-e <int> <int>] [-b <real>] [-k]

The actual procedural method to be used for normals and velocities is specified by the **-m** option. The **-e** option controls the rules for normals and velocities and the endpoints (for open curves only). Bulge can be altered with the **-b** option. Finally, G^2 curves will be constructed if the **-k** option is selected.

The resulting curve can then be displayed with the **curvedge** command (see below). For interpolating curves only (and not surfaces), the **-c** *<int>* option *must* be selected. The integer value (0 or 1) controls whether the defining polygon will not or will be displayed together with the curve itself. If the **-k** option of the **curvedge** command is selected, a *curvature plot*, rather then the curve itself, is constructed.

We now proceed to describe UNICUBIX commands that control the construction of interpolating surfaces.

8.2.4. Displaying and Modifying Boundary Curves

Boundary curves can be displayed by the **curvedge** command:

```
uc> curvedge [-s <int>] [-n] [-i] [-c <int>] [-k]
```

The number of linear segments used for display can be selected by the **-s** option; the default is 10 segments per edge. If the **-n** option is specified, the control net of Bézier points is attached to the curve. Finally, the **-i** option is used if interior boundaries of Bézier subdivision are to be displayed; in this case, the **patch -b** command has to be executed previously (see the next Section).

Vertex normals can be displayed by the **normals** command:

```
uc> normals [-l <real>] [-c]
```

The default length of normals is 1.0, unless specified by the **-l** option. Normals can be cleared from the scene by the **-c** option.

The construction of boundary curves is affected by the following commands:

```
uc> geometry [-n <int>] [-t <int>] [-v <int>]
uc> bulge <real>
uc> vnormal <vid> [<real> <real> <real>]
```

The **geometry** command selects methods for normal, tangent, and velocity determination. The **bulge** command sets the global bulge value for all curves. Finally, the **vnormal** command prints out or sets the normal at a specified vertex. In this case, all previously computed tangent directions at selected vertex will be projected onto the newly defined tangent plane. Velocities will remain unaffected.

8.2.5. Creating and Modifying Surface Patches

Once the boundary curves have been computed, surface patches need to be defined. The command **patch** computes and attaches suitable control points to each non-flat face:

```
uc> patch [-b] [-B] [-p]
```

By default, Gregory patches are created. If the **-b** option is specified, 3 Bézier patches for each original triangular face and 5 Bézier patches for each original quadrilateral face are created

instead. The **-B** option actually adds junction vertices and interior curves of Bézier subdivision to the polyhedral description of the object. Finally, the **-p** option creates a net of control points that is attached to the face.

Gregory and Bézier patches can be modified locally by setting shape parameters (bulge, tilt, shear):

```
uc> shape [-e <vid> <vid>] <real> <real> <real>
```

The **-e** options selects an edge on which the selected shape parameters are set. If this option is not specified, shape parameters are set on all boundary curves.

Finally, the **net** command evaluates all patches at parametric values $(i/n, j/n)$, $0 \le i,j \le n$ for quadrilateral patches, and at $(i/n, j/n, k/n)$, $0 \le i,j,k \le n$, $i+j+k=n$ for triangular patches, where n is the evaluation granularity:

```
uc> net [-s <int>] [-t] [-f]
```

The default granularity is 5, unless specified otherwise by the **-s** option. By default, triangular facets are created for triangular faces, and quadrilateral facets for quadrilateral faces. If the **-t** option is selected, triangular facets are created for all faces; this is useful for shading programs that can deal only with planar facets. The **-f** option creates a net on the original faces, rather then on the interpolating patches. Note that the **net** command destroys the control point information. Thus, if a different granularity is desired, the **patch** command has to be executed again before the **net** command.

Once the smooth object has been created, it is possible that some changes to it will have to be performed. This can be achieved by the modification commands, such as **geometry** or **shape**. However, these commands do not compute interior control points of the patches. Thus, the **patch** command needs to be rerun before any **geometry** or **shape** commands show the resulting effects.

8.2.6. Other Commands

There is a number of *uci* commands that can be useful for debugging purposes. For example, the **kurvature** command prints out curvature information of the patches. The **source** command can be used to read in the UNICUBIX command file and to execute statements from it (batch mode). Thus, the desired modifications to the original UNIGRAFIX object can be stored in the command file. The **copy** command copies the backup data structure into the current structure (see the next Section). Usually, this operation is performed automatically if the current structure was changed by, for example, **curvedge** or **net** commands.

Other *uci* commands are taken directly from *ugi* [120], an interactive display tool for UNIGRAFIX scenes. Useful commands include **clear** for clearing current scene, and **write** for writing the current scene into a file. The **write** command can also be used to output patch information in

a format suitable for other programs. These programs include a contouring tool for Gregory patches, used in Ricoh Corp. DESIGNBASE [26], and a surface evaluation package, developed at the University of Washington, Seattle [87]. The reader is referred to the UNICUBIX manual for more information.

8.3. Implementation Details

8.3.1. Data Structure

The core UNICUBIX data structure is inherited from UNIGRAFIX. However, to store information about curved objects (such as vertex normals and Bézier or Gregory control points), extra fields were added to the appropriate structures. The major structures in UNICUBIX are VERTEX, EDGE, and FACE.

8.3.1.1. Major Data Structures

The VERTEX structure contains vertex coordinates, a pointer to the list of edges (ELIST) meeting at this vertex, and a pointer to the list of faces (FLIST) using the vertex. In addition, there is a pointer to a VECT structure that holds the 3 coordinates of the normal.

The EDGE structure points to its 2 endpoint VERTEX structures, and to the list of faces (FLIST) using it. Two VECT pointers are provided for the two interior Bézier points of the curved boundary, as well as the pointer to the SHPAR structure that contains 3 shape parameters (bulge, tilt, and shear). The flag field of the structure contains information about the edge type (STREDGE, CUREDGE, STRBORDER, CURBORDER).

The FACE structure points to its contour list (in UNICUBIX, this list can have only one element, as no holes are allowed). The contour list (FCONTLIST) points to the contour structure (CONTOUR) that, in turn, points to the ELIST of edges of the current face. In addition, the FACE structure contains an array of 20 pointers to the control points of a Gregory patch, and 5 arrays of pointers to the control points of Bézier subdivision patches. There is also an array of 4 VERTEX pointers to the face corners, and an array of 4 SHPAR pointers to the shape parameters of the face edges. Finally, the flag field of the structure differentiates between face types (PLANAR or CURPATCH).

The other data structures for transformations, color, illumination, etc. were taken directly from UNIGRAFIX without any changes. The reader is referred to the appropriate UNIGRAFIX documentation [120] for their description.

8.3.1.2. Data Structure Modification

At any stage of the program execution, *two* independent data structures are always maintained: a *current structure*, and its *backup copy*. When an object is first read in, both data

structures are initialized to contain this object. However, all the modifications are performed on the current structure only. Thus, it is always possible to restore the original object by copying the backup structure into the current structure. The **copy** command does exactly that. This avoids the necessity to read in the file description after a change has been made.

However, the restoration of the original object is performed *automatically* if the current structure has been changed so that no further modification of it is possible. For example, the **net** command deletes the original object and replaces it with the polyhedral representation of the evaluated surface. Similarly, the **curvedge** command replaces the original object with a wireframe representing the curved edges. So, if, for example, a **patch** command is issued after **curvedge**, the original structure is first implicitly restored, and then the **patch** command itself is executed on the restored structure.

Other UNICUBIX commands that modify the structure are:

- **patch -p** adds a wireframe that connects the control points of the face to its vertices;
- **patch -B** deletes the original faces and replaces them with Bézier subdivision subfaces;
- **normals** adds a wire representing the vertex normal to each vertex of the object.

The remaining boundary and patch modification commands just operate on certain fields of the structure, such as pointers to control points. If the structure has been previously modified by one of the above commands, the original structure is restored.

- **gcurve** computes two control points for each edge and adds the proper pointers to the structure;
- **geometry** sets global variables corresponding to a chosen method of boundary curve construction;
- **bulge** recomputes control points on boundary curves using the provided bulge coefficient;
- **vnormal** changes the normal at the specified vertex and recomputes the Bézier points of the edges meeting at this vertex;
- **patch** [**-b**] computes interior control points of Gregory or Bézier patches;
- **shape** sets the specified shape parameters on one or all edges.

8.3.2. Representation of Gregory and Bézier patches

As mentioned above, a curved object can be represented as union of Gregory or Bézier patches. If the Bézier option is specified, each triangular face is represented with 3 triangular patches, while quadrilateral faces are represented with 5 quadrilateral patches. Fig. 8.1 shows relative positions of Gregory control points in triangular and quadrilateral patches. The numbers near control points correspond to the indices in the VECT array of control points that is associated with each face.

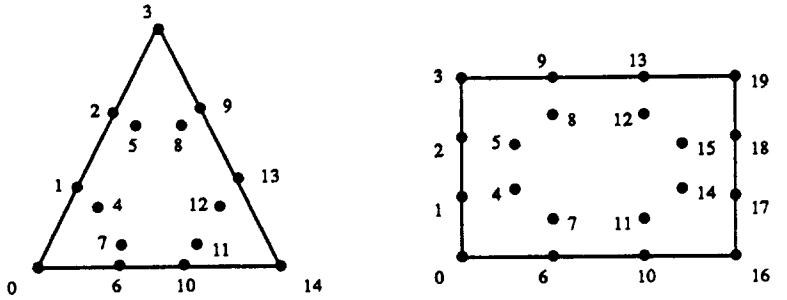


Figure 8.1: *Relative positions of Gregory control points.*

Fig. 8.2 shows Bézier subpatches that represent an original face. Again, the numbers near control points are the indices of the corresponding VECT array; the subpatch number is indicated by roman numerals.

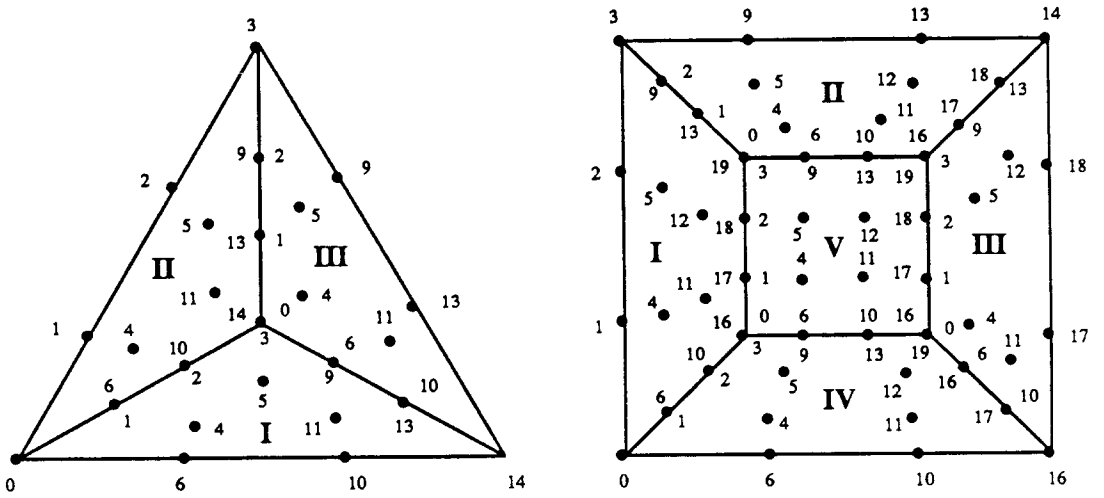


Figure 8.2: *Relative positions of Bézier control points.*

8.3.3. Surface Evaluation

In order to be able to view the final constructed surface, each surface patch has to be evaluated on a certain grid of points in the parameter space of the patch. In the current implementation of UNICUBIX, this is done by direct evaluation. Triangular Gregory patches are evaluated by the formula (2.54) at parametric values $(i/n, j/n, k/n)$, $0 \le i,j,k \le n$, $i+j+k=n$, where n is the evaluation of granularity (as specified in the **net** command). Analogously, quadrilateral Gregory patches are evaluated by (2.45) at parametric values $(i/n, j/n)$, $0 \le i,j \le n$).

Similarly, each Bézier subpatch can also be evaluated separately of the other subpatches that correspond to a face. However, the resulting net representation creates an illusion of one extra vertex in the middle of the face in the case of triangular subdivision, and 4 extra vertices in the case of quadrilateral subdivision. To get rid of this unpleasant (purely visual) effect, we construct the grid of parameter values of the enclosing face, and then map each grid point to the parameter space of the subface. Then the Bézier patch is evaluated. We illustrate this process for triangular and quadrilateral subdivisions.

Triangular Subdivision. Suppose that some point P has the barycentric coordinates (r,s,t) in the parameter space of the big triangle and the barycentric coordinates (u,v,w) in the parameter space of the subtriangle it lies in (Fig. 8.3a). Assuming that the center vertex has coordinates (1/3, 1/3, 1/3), it easily follows that P has the following representations in the parameter space of the subtriangle it is in:

I. $u = r - s,\ v = 3s,\ w = t - s, \quad [s < r,\ s < t]$;

II. $u = 3r,\ v = s - r,\ w = t - r, \quad [r < s,\ r < t]$;

III. $u = r - t,\ v = s - t,\ w = 3t, \quad [t < r,\ t < s]$.

So, for each triple (r,s,t) of the original grid, parameter values of the proper subtriangle are computed and the corresponding Bézier patch evaluated at (u,v,w) by the formula (2.38).

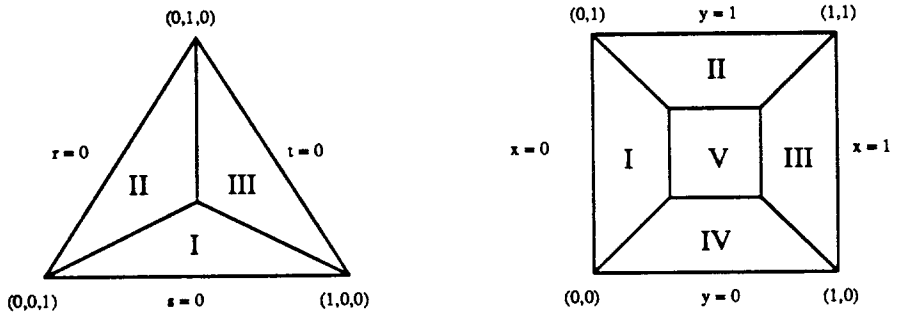


Figure 8.3: *Parameter space regions for triangles* (a) *and for quadrilaterals* (b).

Quadrilateral Subdivision. Analogously, suppose that some point P has the coordinates (x,y) in the parameter space of the big quadrilateral. We assume that the coordinates of the 4 center points in Fig. 8.3b are (1/3,1/3), (1/3,2/3), (2/3,1/3), and (2/3,2/3). Some computation shows that P can be represented as follows in the parameter space (u,v) of the subquadrilateral it is in:

I. $u = 3x,\ v = \dfrac{y - x}{1 - 2x}, \quad [x < y,\ x + y < 1,\ x < \dfrac{1}{3}]$;

II. $u = \frac{x+y-1}{2y-1}$, $v = 3y-2$, $[x < y,\ x+y > 1,\ y > \frac{2}{3}]$;

III. $u = 3x-2$, $v = \frac{x+y-1}{2x-1}$, $[x > y,\ x+y > 1,\ x > \frac{2}{3}]$;

IV. $u = \frac{x-y}{1-2y}$, $v = 3y$, $[x > y,\ x+y < 1,\ y < \frac{1}{3}]$;

V. $u = 3x-1$, $v = 3y-1$, $[\frac{1}{3} < x, y < \frac{2}{3}]$.

The proper Bézier patch is then evaluated at (u,v) according to the formula (2.34).

8.4. Discussion

The implemented prototype for UNICUBIX, *uci*, is a modeling system that allows to describe curved objects defined by cubic boundary curves. The system is still in an experimental stage, and, consequently, has rather restricted capabilities. The greatest drawback of the system is the fact that it has a very limited user interface (type-in commands only). An interactive editor with a mouse support is a must. Such an editor would introduce real time interaction to the system.

Another disadvantage of the system is the fact that it is computationally unefficient because of time-consuming direct evaluations of the points on curve segments and surface patches. There are two possible approaches to improve efficiency. First, one could use a fast evaluation algorithm, such as forward differencing. However, a possibly better approach would be to eliminate the step of curve/surface evaluation altogether from the system. This should be possible if *uci* is run on a modern workstation with a graphics library that supports Bézier patches and general NURBS. Then, control points of a curve or a patch would be directly passed to the library for rendering.

In spite of the above disadvantages, UNICUBIX is the foundation for a needed addition to the UNIGRAFIX system that would bring it closer to a useful CAD modeler. In our research it has proved to be a good tool to understand the relationship between various geometric parameters that define the construction process and the shape of the final interpolating surface.

9

Conclusions

In this work, we have described local methods to construct tangent continuous (G^1) and curvature continuous (G^2) curves, and tangent continuous surfaces that interpolate given data points. The curves are built of cubic Bézier segments, joined together with the required degree of geometric continuity, while surfaces are represented as a union of quadrilateral bicubic and/or quartic triangular Bézier or Gregory patches.

The *procedural* approach was used in both curve and surface construction. This approach is 'smarter' than traditional algebraic ones in the sense that it is based on geometric reasoning. Thus, a curve or a surface is derived from a sequence of procedural steps that may include special rules. These rules generally take into account a special situation of a particular piece of a curve or a surface which can greatly improve the visual quality of a resulting object.

The system of curve segments or surface patches that produces overall G^1 continuity has several degrees of freedom. It is our belief that in a practical modeling system these degrees of freedom should be represented not as control points or weights of a particular patch, but as high-level user-friendly handles, or *shape parameters*. Shape parameters affect the shape of a curve or a surface in an intuitive manner that does not depend on the underlying functional representation and are transparent even to a nonmathematical user.

The procedural approach also makes the system very flexible in the sense that a variety of input data can be processed. For example, smooth G^1 surfaces can be constructed just from a set of points in $\mathbf{R}^3$, or from a fully-defined mesh of boundary curves. If certain shape parameters are not available from the input, our procedural method assigns *default* values to these parameters. The proper assignment of default values is of great importance to the visual quality of the final curve or surface. The user, however, has an option to change them for the application's particular needs.

9.1. Relationship to Other Data Interpolation Methods

The problem of interpolating a set of points with a smooth curve or a surface is a classical problem of approximation theory. Traditional approaches often use a single function, such as a

polynomial, to interpolate all the points. The resulting solution, while being C^∞ continuous, may exhibit undesirable oscillations between the data points.

Alternatively, the spline-based methods represent the interpolant as a union of continuous (typically polynomial) non-overlapping pieces that connect at the interpolation points. In order to define these spline pieces unambiguously, the *topological structure* of the desired interpolating surface must be specified. We assume that it is either explicitly given in the form of a topology polyhedron on the points to be interpolated or that a suitable polyhedron can readily be found by some straightforward triangulation procedure on the data points [109]. As a result of using a spline representation, the oscillations between the data points are reduced at the cost of decreasing the degree of continuity of the interpolant.

In CAGD, the parametrization of the interpolating curve or a surface is not important. Thus, the continuity requirements can be relaxed even further by enforcing *geometric*, rather that parametric continuity at the data points. This relaxation of continuity requirements leads to the availability of several parameters that need to be assigned certain values in order to construct a particular solution. The choice of these parameters is of crucial importance. Most current interpolation schemes either assign some ad hoc values to these parameters or leave them to be defined by the user. As a result, undesirable 'overshoots' or 'bulges' may occur between the data points, decreasing overall visual quality of the interpolant.

In this work, we have attempted to conduct a thorough study of how these parameters should be defined. We have proposed several procedural rules that assign default values to the available degrees of freedom with the goal to improve certain qualities of the resulting curves or surfaces. The same approach also allowed us to represent G^2 interpolating curves with 2 cubics per segment (Chapter 4), and G^1 surfaces with 3 or 5 Bézier patches per face (Chapter 7).

We have also studied ways to make the available degrees of freedom user-friendly. We have extracted a set of *shape parameters* that can be used to modify the curve or surface. The shape parameters have a very intuitive effect on the shape of the primitive and are clearly superior to 'twiddling' control points or assigning values to some obscure coefficients.

We believe that the procedural approach is a very powerful tool in constructing smooth interpolants. It offers good visual quality, user-friendly local and global shape parameters, and a simple representation of the underlying splines.

9.2. Major Contributions

The following major contributions have been presented in this thesis:

1. We have studied the problem of interpolating a set of points in $\mathbf{R}^2$ or $\mathbf{R}^3$ with G^1 Bézier curves. We have established default rules for finding curve *normals* at interpolation points, and *velocities* that determine positions of interior Bézier control points on tangent lines. These rules consistently produce curves of high visual quality (measured in terms of

curvature variation) even for highly irregularly spaced data points. Moreover, *special rules*, such as those concerning collinear segments, have been developed.

2. We have presented a local method to construct interpolating curvature continuous (G^2) planar curves represented as a union of cubic Bézier segments, with two segments per span between adjacent vertices. In addition to data points, curve normals, velocities, and curvature values at interpolation points may be specified.

3. We have experimented with several approaches to construct cubic boundary curves from a given polyhedral model. The boundaries are completely defined by three types of *geometric parameters*: *vertex normals*, *tangent directions*, and *velocities*. We found that naive methods for determining these parameters not only produce surfaces of poor visual quality, but sometimes fail to produce a G^1 surface at all. Therefore, we have developed a method that is based on constructing several G^1 curves through the interpolation point. The geometric parameters are then derived from these curves. The resulting *Opposite Edge method* proves to be very robust; moreover, a substantial improvement in surface quality (again measured in terms of curvature variation) has been realized.

4. We have studied the conditions for G^1 continuity between adjacent Bézier or Gregory patches, and derived 3 shape parameters, *bulge*, *tilt*, and *shear* that control the shape of the patches near their common boundary without changing this boundary or destroying G^1 continuity.

5. We have described a process of interpolating a single Gregory patch with several Bézier patches so that patch boundaries and cross-boundary derivatives are interpolated exactly. This is important for efficiency and compatibility purposes. A quartic triangular Gregory patch is replaced by 3 quartic triangular Bézier patches, while a bicubic quadrilateral Gregory patch can be replaced with either 2 quadrilateral and 2 triangular Bézier patches, or with 5 quadrilateral Bézier patches.

9.3. Open Problems

The following issues require more research:

1. The current rules for determination of normals and velocities for interpolating curves depend only on the nearest neighbors of the current vertex. However, in order to check if a special rule needs to be applied, *second nearest* neighbors have to be examined. Given that we have to look at the second nearest neighbors anyhow, this additional information could be used to formulate better default rules for normals and velocities. The tradeoffs between these more sophisticated rules and their increased computational cost should be studied.

2. Our G^2 curve interpolation method does not extend naturally to $\mathbf{R}^3$ due to an additional constraint imposed by osculating planes. The next logical step is to research whether the osculating planes can always be chosen locally so that pairs of G^2 cubics can be

constructed. If the answer is negative (and we believe that it is), interpolation with quintic splines should be studied. In particular, default rules need to be developed for choosing the two 'innermost' control points of a quintic segment.

3. At present, our surface interpolation scheme can only deal with triangular and/or quadrilateral topology of underlying faces. In Chapter 7, we presented a method to construct a G^1 surface over a regular n-gon. An extension to arbitrary faces would certainly be desirable. The major difficulty here is the fact that n-sided patches suitable for inclusion into a network of Bézier patches are of very high degree [85].

4. It would be interesting to study whether the basic idea of Chiyokura's G^1 blending could be extended to form G^2 surfaces. To preserve local control, *quintic* Gregory patches would have to be defined. These patches should allow independent specifications of the first and second cross-boundary derivatives at the patch borders and should be derivable from quintic Bézier patches by splitting each interior control point.

Bibliography

1. J. Adams, "Cubic Spline Fitting with Controlled End Conditions," *Computer Aided Design*, vol. 6, no. 1, pp. 2-9, January 1974.

2. R. Barnhill, "Smooth Interpolation over Triangles," in *Computer Aided Geometric Design*, ed. R. Riesenfeld, Academic Press, Inc., Orlando, Florida, March 1974.

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

4. B. Barsky, "Local Control of Bias and Tension in Beta-splines," *ACM Transactions on Graphics*, vol. 2, pp. 109-134, 1983.

5. B. Barsky, *Computer Graphics and Geometric Modeling Using Beta-splines*, Springer-Verlag, Tokyo, 1988.

6. B. Barsky, "Introducing the Rational Beta-spline," *Proceedings ICEGDG*, vol. 1, pp. 16-27, Vienna, 1988.

7. B. Barsky and T. DeRose, "Geometric Continuity for Parametric Curves," Technical Report No. UCB/CSD 84/205, Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, October 1984.

8. B. Barsky and T. DeRose, "Geometric Continuity of Parametric Curves: Three Equivalent Characterizations," *IEEE Computer Graphics and Applications*, vol. 9, no. 6, pp. 60-68, November 1989.

9. B. Barsky and T. DeRose, "Geometric Continuity of Parametric Curves: Constructions of Geometrically Continuous Splines," *IEEE Computer Graphics and Applications*, vol. 10, no. 1, pp. 60-68, January 1990.

10. R. Bartels, J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Inc, Los Altos, California, 1987.

11. P. Bézier, *The Mathematical Basis of the UNISURF CAD System*, Butterworths & Co Ltd., London, England, 1986.

12. L. Bickhardt, "Interpolative Surface Definition with Automatic Determination of Tangent and Twist Vectors," *Eurographics '79*, pp. 212-218, 1979.

13. W. Boehm, "On Cubics: A Survey," *Computer Graphics and Image Processing*, vol. 19, pp. 201-226, Academic Press, Inc., 1982.

14. W. Boehm, "Curvature Continuous Curves and Surfaces," *Computer Aided Geometric Design*, vol. 2, no. 6, pp. 313-323, 1985.

15. W. Boehm, "Multivariate Spline Methods in CAGD," *Computer Aided Design*, vol. 18, no. 2, pp. 102-104, March 1986.

16. W. Boehm, "Rational Geometric Splines," *Computer Aided Geometric Design*, vol. 4, pp. 67-78, 1987.

17. W. Boehm, G. Farin, and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design*, vol. 1, no. 1, pp. 1-60, 1984.

18. C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978.

19. C. de Boor, K. Höllig, and M. Sabin, "High Accuracy Geometric Hermite Interpolation," *Computer Aided Geometric Design*, vol. 4, pp. 269-278, 1987.

20. J. Brewer and D. Anderson, "Visual Interaction with Overhauser Curves and Surfaces," *Computer Graphics*, vol. 11, no. 2, pp. 132-137, July 1977.

21. K. Brodlie, "Methods for Drawing Curves," in *Fundamental Algorithms for Computer Graphics*, ed. R. Earhshaw, pp. 303-323, Springer-Verlag, Berlin, 1985.

22. P. Brunet, "Increasing the Smoothness of Bicubic Spline Surfaces," *Computer Aided Geometric Design*, vol. 2, pp. 157-164, 1985.

23. E. Catmull and R. Rom, "A Class of Local Interpolating Splines," in *Computer Aided Geometric Design*, ed. R. Riesenfeld, pp. 317-326, Academic Press, Inc., Orlando, Florida, March 1974.

24. P. Charrot and J. Gregory, "A Pentagonal Surface Patch for Computer Aided Geometric Design," *Computer Aided Geometric Design*, vol. 1, pp. 87-94, 1984.

25. H. Chiyokura, "Localized Surface Interpolation Method for Irregular Meshes," *Advanced Computer Graphics*, pp. 3-19, Springer-Verlag, Tokyo, 1986.

26. H. Chiyokura, *Solid Modeling with DESIGNBASE*, Addison-Wesley, 1988.

27. H. Chiyokura and F. Kimura, "Design of Solids with Free-Form Surfaces," *Computer Graphics*, vol. 17, no. 3, pp. 289-298, 1983.

28. A. Cline, "Scalar- and Planar-Valued Curve Fitting Using Splines Under Tension," *Communications of the ACM*, vol. 17, no. 4, pp. 218-220, April 1974.

29. S. Coons, *Surfaces for Computer Aided Design*, Design Division, Mechanical Engineering Department, M.I.T., Cambridge, Massachusetts, 1964.

30. S. Coons, "Surfaces for Computer Aided Design of Space Forms," MAC-TR-41, Project MAC, Massachusetts Institute of Technology, 1967.

31. F. Cros and P. Brock, "A Method for Providing Full Interactive Control of the Shape of 3-D Curves and Surfaces," *Eurographics '88*, pp. 443-455, North-Holland Publishing Company, Amsterdam, The Netherlands, September 1988.

32. T. DeRose, "Geometric Continuity: A Parametrization Independent Measure of Continuity for Computer Aided Geometric Design," Ph.D. Thesis (Report No. UCB/CSD 86/255), University of California, Berkeley, California, 1985.

33. T. DeRose, "Necessary and Sufficient Conditions for Tangent Plane Continuity of Bézier Surfaces," *Computer Aided Geometric Design*, vol. 7, pp. 165-179, 1990.

34. T. DeRose and B. Barsky, "Geometric Continuity, Shape Parameters, and Geometric Constructions for Catmull-Rom Splines," *ACM Transactions on Graphics*, vol. 7, no. 1, pp. 1-41, January 1988.

35. L. Degen, "Explicit Continuity Conditions for Adjacent Bézier Surface Patches," *Computer Aided Geometric Design*, vol. 7, pp. 181-189, 1990.

36. R. Delbourgo and J. Gregory, "C^2 Rational Quadratic Spline Interpolation to Monotonic Data," *IMAJ on Numerical Analysis*, vol. 3, pp. 141-152, 1983.

37. N. Dyn and C. Michelli, "Shape Preserving Parametric Representation of Curves with Local Control for CAGD," IBM Res. Rep., Mathematical Sciences Dept., IBM T.J. Watson Research Center, Yorktown Heights, N.Y., January 1985.

38. G. Farin, "Designing C^1 Surfaces Consisting of Triangular Cubic Patches," *Computer Aided Design*, vol. 14, no. 5, pp. 253-256, September 1982.

39. G. Farin, "A Construction for Visual Continuity of Polynomial Surface Patches," *Computer Graphics and Image Processing*, vol. 20, pp. 272-282, 1982.

40. G. Farin, "Visually C^2 Cubic Splines," *Computer Aided Design*, vol. 14, no. 3, pp. 137-139, May 1982.

41. G. Farin, "Smooth Interpolation to Scattered 3D Data," in *Surfaces in Computer Aided Geometric Design*, ed. W. Boehm, pp. 1-24, North Holland, Amsterdam, The Netherlands, 1983.

42. G. Farin, "Triangular Bézier-Bernstein Patches," *Computer Aided Geometric Design*, vol. 3, no. 2, 1986.

43. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, Inc, 1988.

44. G. Farin, G. Rein, N. Sapidis, and A. Worsey, "Fairing Cubic B-spline Curves," *Computer Aided Geometric Design*, vol. 4, no. 1-2, pp. 91-104, 1987.

45. G. Farin and N. Sapidis, "Fairing Curves," *IEEE Computer Graphics and Applications*, pp. 52-57, March 1989.

46. I. Faux and M. Pratt, *Computational Geometry for Design and Manufacture*, John Wiley & Sons, New York, 1979.

47. G. Fletcher and D. McAllister, "Natural Bias Approach to Shape Preserving Curves," *Computer Aided Design*, vol. 18, no. 1, pp. 48-52, 1986.

48. N. Fog, "Creative Definition and Fairing of Ship Hulls Using a B-spline Surface," *Computer Aided Design*, vol. 16, no. 4, pp. 225-229, 1984.

49. R. Franke, "A Critical Comparison of Some Methods for Interpolation of Scattered Data," Technical Report NPS-53-79-003, Naval Postgraduate School, 1979.

50. R. Franke and A. Forrest, "The Twisted Cubic Curve: A Computer Aided Geometric Design Approach," *Computer Aided Design*, vol. 12, no. 4, pp. 165-172, July 1980.

51. F. Fritsch, "Energy Comparisons of Wilson-Fowler Splines with Other Interpolating Splines," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, pp. 185-201, SIAM, Philadelphia, PA, 1987.

52. F. Fritsch and R. Carlson, "Monotone Piecewise Cubic Interpolation," *SIAM Journal on Numerical Analysis*, vol. 17, pp. 238-246, 1980.

53. J. Glass, "Smooth Curve Interpolation: A Generalized Spline-Fit Procedure," *BIT*, vol. 9, pp. 277-293, Copenhagen, Denmark, 1966.

54. T. Goodman, "Properties of β-splines," *Journal on Approximation Theory*, vol. 44, pp. 132-153, 1985.

55. T. Goodman and K. Unsworth, "Shape Preserving Interpolation by Parametrically Defined Curves," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 1-13, 1988.

56. T. Goodman and K. Unsworth, "Shape Preserving Interpolation by Curvature Continuous Parametric Curves," *Computer Aided Geometric Design*, vol. 5, pp. 323-340, 1988.

57. W. Gordon, "Free-form Surface Interpolation Through Curve Networks," Technical Report, General Motors Research Laboratories, 1969.

58. W. Gordon, "Spline-blended Surface Interpolation Through Curve Networks," *Journal of Mathematics and Mechanics*, vol. 18, no. 10, pp. 931-952, 1969.

59. W. Gordon and R. Riesenfeld, "B-spline Curves and Surfaces," in *Computer Aided Geometric Design*, ed. R. Riesenfeld, pp. 95-126, Academic Press, Inc., Orlando, Florida, March 1974.

60. T. Gossing, "Bulge, Shear, and Squash: A Representation for the General Conic Arc," *Computer Aided Design*, vol. 13, no. 2, pp. 81-84, March 1981.

61. J. Gregory, "Smooth Interpolation Without Twist Constraints," in *Computer Aided Geometric Design*, ed. R. Riesenfeld, Academic Press, Inc., Orlando, Florida, March 1974.

62. J. Gregory, "C^1 Rectangular and Non-Rectangular Surface Patches," in *Surfaces in Computer Aided Geometric Design*, ed. W. Boehm, pp. 25-33, North Holland, Amsterdam, The Netherlands, 1983.

63. J. Gregory, "Shape Preserving Spline Interpolation," *Computer Aided Design*, vol. 18, no. 1, pp. 53-57, 1986.

64. J. Gregory and P. Charrot, "A C^1 Triangular Interpolation Patch for Computer Aided Geometric Design," *Computer Graphics and Image Processing*, vol. 13, pp. 80-87, 1980.

65. H. Hagen, "Geometric Spline Curves," *Computer Aided Geometric Design*, vol. 2, pp. 223-227, 1985.

66. K. Harada and E. Nakamae, "Applications of the Bézier Curve to Data Interpolation," *Computer Aided Design*, vol. 14, no. 1, pp. 55-60, January 1982.

67. G. Herron, "Smooth Closed Surfaces with Discrete Triangular Interpolants," *Caomputer Aided Geometric Design*, vol. 2, pp. 2997-306, 1985.

68. G. Herron, "Techniques for Visual Continuity," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, pp. 163-174, SIAM, Philadelphia, PA, 1987.

69. M. Higashi, K. Kaneko, and M. Hosaka, "Generation of High-Quality Curve and Surface with Smoothly Varying Curvature," *Eurographics '88*, pp. 79-92, North-Holland Publishing Company, Amsterdam, The Netherlands, September 1988.

70. M. Hosaka and F. Kimura, "Non-four-sided Patch Expression with Control Points," *Computer Aided Geometric Design*, vol. 1, pp. 75-86, 1984.

71. J. Hoschek, "Detecting Regions with Undesirable Curvature," *Computer Aided Geometric Design*, vol. 1, no. 2, pp. 183-192, 1984.

72. M. Hourdaquin and P. Coheignoux, "Specifying Arbitrary Planar Smooth Curves for Fast Drawing," *Eurographics '79*, pp. 193-211, 1979.

73. T. Jensen, "Assembling Triangular and Rectangular Patches and Multivariate Splines," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, pp. 203-220, SIAM, Philadelphia, PA, 1987.

74. A. Jones, "Shape Control of Curves and Surfaces through Constrained Optimization," in *Geometric Modeling: Algorithms and Trends*, ed. Gerald E. Farin, pp. 265-279, SIAM, Philadelphia, PN, 1987.

75. M. Jordon and F. Schindler, "Curves under Tension," *Computer Aided Geometric Design*, vol. 1, pp. 291-300, 1984.

76. J. Kahmann, "Continuity of Curvature Between Adjacent Bezier Patches," in *Surfaces in Computer Aided Geometric Design*, ed. W. Boehm, pp. 65-75, North Holland, Amsterdam, The Netherlands, 1983.

77. F. Kimura, "Geomap III: Designing Solids with Free-form Surfaces," *IEEE Computer Graphics and Applications*, vol. 4, no. 6, pp. 58-72, June 1984.

78. J. Kjellander, "Smoothing of Cubic Parametric Splines," *Computer Aided Design*, vol. 15, no. 3, pp. 175-179, 1983.

79. D. Kochanek and R. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *SIGGRAPH'84 Conference Proceedings*, vol. 18, no. 3, pp. 33-41, ACM, Minneapolis, MN, July 1984.

80. E. Lee and G. Forsythe, "Variational Study of Nonlinear Spline Curves," *SIAM review*, vol. 15, no. 1, pp. 120-133, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, January 1973.

81. Y. Liang, X. Ye, and S. Fang, "G^1 Smoothing Solid Objects by Bicubic Bezier Patches," *Eurographics '88*, pp. 343-355, North-Holland Publishing Company, Amsterdam, The Netherlands, September 1988.

82. D. Liu, "GC^1 Continuity Conditions for Smoothness between Adjacent Rational Bezier Surface Patches," *Computer Aided Geometric Design*, vol. 7, pp. 151-163, 1990.

83. C. Long, "Special Bézier Quartics in 3-dimensional Curve Design and Interpolation," *Computer Aided Design*, vol. 19, no. 2, pp. 77-84, March 1987.

84. L. Longhi, "Interpolating Patches Between Cubic Boundaries," Master's Project Report, Computer Science Division, University of California, Berkeley, California, 1985.

85. C. Loop and T. DeRose, "A Multisided Generalization of Bézier Surfaces," *ACM Transcations on Graphics*, vol. 8, no. 3, pp. 204-234, July 1989.

86. N. Lott and D. Pullin, "Method for Fairing B-spline Surfaces," *Computer Aided Design*, vol. 20, no. 10, pp. 597-604, December 1988.

87. M. Lounsbery, C. Loop, S. Mann, D. Meyers, J. Painter, T. DeRose, and K. Sloan, "A Testbed for the Comparison of Parametric Surface Methods," in *Curves and Surfaces in Computer Vision and Graphics*, ed. R. de Figueiredo, vol. 1251, pp. 94-105, SPIE Proceedings, Santa Clara, California, February 1990.

88. J. Manning, "Continuity Conditions for Spline Curves," *The Computer Journal*, vol. 17, pp. 181-186, British Computer Society, London, England, 1974.

89. H. McLaughlin, "Shape Preserving Planar Interpolation: An Algorithm," *IEEE Computer Graphics and Applications*, vol. 3, no. 3, pp. 58-67, 1983.

90. L. Montefusco, "An Interactive Procedure for Shape Preserving Cubic Spline Interpolation," *Computers and Graphics*, vol. 11, no. 4, pp. 389-392, 1987.

91. M. Mortenson, *Geometric Modeling*, John Wiley & Sons, New York, 1985.

92. G. Nielson, "Some Piecewise Polynomial Alternatives to Splines Under Tension," in *Computer Aided Geometric Design*, ed. R. Riesenfeld, pp. 209-235, Academic Press, Inc., Orlando, Florida, March 1974.

93. G. Nielson, "A Transfinite, Visually Continuous, Triangular Interpolant," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, pp. 235-246, SIAM, Philadelphia, PA, 1987.

94. S. Ohlin, "Splines for Engineers," *Eurographics '87*, pp. 555-565, North-Holland Publishing Company, Amsterdam, The Netherlands, 1987.

95. A. Overhauser, "Analytic Definition of Curves and Surfaces by Parabolic Blending," Technical Report No. SL 68-40, Ford Motor Company Scientific Laboratory, May 1968.

96. T. Pal, "Intrinsic Spline Curve with Local Control," *Computer Aided Design*, vol. 10, no. 1, pp. 19-29, January 1978.

97. E. Passow and J. Roulier, "Monotone and Convex Spline Interpolation," *SIAM Journal on Numerical Analysis*, vol. 14, pp. 904-909, 1977.

98. J. Peters, "Local Generalized Hermite Interpolation by Quartic C^2 Space Curves," *ACM Transcations on Graphics*, vol. 8, no. 3, pp. 235-242, July 1989.

99. J. Peters, "Smooth Mesh Interpolation with Cubic Patches," *submitted to CAGD*, 1989.

100. J. Peters, "Smooth Interpolation of a Mesh of Curves," *submitted to CAGD*, 1989.

101. J. Peters, "Local Smooth Surface Interpolation: A Classification," *Computer Aided Geometric Design*, vol. 7, pp. 191-195, 1990.

102. B. Pham, "Conic Beta-splines with Local Tension Control for Interactive Curve Fitting," *Eurographics '88*, pp. 67-78, North Holland Publishing Company, Amsterdam, the Netherlands, 1988.

103. L. Piegl, "Defining C^1 Curves Containing Conic Segments," *Computers and Graphics*, vol. 8, no. 2, pp. 177-182, 1984.

104. L. Piegl, "Curve Fitting Algorithm for Rough Cutting," *Computer Aided Design*, vol. 18, no. 2, pp. 79-83, March 1986.

105. L. Piegl and W. Tiller, "Curve and Surface Construction Using Rational B-splines," *Computer Aided Design*, vol. 19, no. 9, pp. 485-498, Novermber 1987.

106. B. Piper, "Visually Smooth Interpolation with Triangular Bézier Patches," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, pp. 221-233, SIAM, Philadelphia, PA,

1987.

107. M. Powell, *Approximation Theory and Methods*, Cambridge University Press, Cambridge, 1981.

108. V. Pratt, "Techniques for Conic Splines," *Computer Graphics*, vol. 19, no. 3, pp. 151-159, 1985.

109. F. Preparata and M. Shamos, *Computational Geometry, An Introduction*, Springer-Verlag, New York, 1985.

110. G. Renner and V. Pochop, "A New Method for Local Smooth Interpolation," *Eurographics '81*, pp. 137-150, North Holland Publishing Company, Amsterdam, the Netherlands, 1981.

111. T. Reunding, "Bézier Patches on Cubic Grid Curves — An Application to the Programming Design of a Yacht Hull Surface," *Computer Aided Geometric Design*, vol. 6, no. 1, pp. 11-21, February 1989.

112. R. Riesenfeld, "Summary of the Concepts of the Alpha-1 CAGD System," *Detroit Engineer*, pp. 8-11, 1982.

113. M. Sabin, "Non-Rectangular Surface Patches Suitable for Inclusion in a B-spline Surface," *Eurographics '83*, pp. 57-69, North-Holland Publishing Company, Amsterdam, The Netherlands, 1983.

114. M. Sabin, "Some Negative Results in n-sided Patches," *Computer Aided Design*, vol. 18, pp. 38-44, 1986.

115. R. Sarraga, "G^1 Interpolation of Generally Unrestricted Cubic Bézier Curves," *Computer Aided Geometric Design*, vol. 4, pp. 23-41, 1987.

116. F. Schmitt and W. Du, "Bézier Patches with Local Shape Control Parameters," *Eurographics '87*, pp. 261-274, North-Holland Publishing Company, Amsterdam, The Netherlands, 1987.

117. L. Schumaker, "On Shape Preserving Quadratic Spline Interpolation," *SIAM Journal on Numerical Analysis*, vol. 20, pp. 854-864, 1983.

118. D. Schweikert, "An Interpolation Curve Using a Spline in Tension," *Journal of Mathematics and Physics*, vol. 45, pp. 312-317, 1966.

119. C. Séquin, "Berkeley UNIGRAFIX: A Modular Rendering and Modeling System," *Proceedings of the Second USENIX Computer Graphics Workshop*, pp. 38-53, 1985.

120. C. Séquin, "The Berkeley UNIGRAFIX Tools," Tech. Report No. UCB/CSD 86/281, Computer Science Division, University of California, Berkeley, California, 1985.

121. C. Séquin, "Procedural Spline Interpolation in UNICUBIX," *Proceedings of the Third USENIX Computer Graphics Workshop*, pp. 68-83, Monterey, California, 1986.

122. T. Sederberg, "Implicit and Parametric Curves and Surfaces for Computer Aided Geometric Design," Ph.D. Thesis, Purdue University, 1983.

123. T. Sederberg, "Planar Piecewise Algebraic Curves," *Computer Aided Geometric Design*, vol. 1, pp. 241-255, December 1984.

124. T. Sederberg, "Piecewise Algebraic Surface Patches," *Computer Aided Geometric Design*, vol. 2, pp. 53-59, September 1985.

125. M. Stone and T. DeRose, "A Geometric Characterization of Parametric Cubic Curves," *ACM Transactions on Graphics*, vol. 8, no. 3, pp. 147-163, July 1989.

126. M. Veron, G. Ris, and J. Musse, "Continuity of Biparametric Surface Patches," *Computer Aided Design*, vol. 8, no. 4, pp. 267-273, 1976.

Appendix1: Control Points and Premultiplication Coefficients

As discussed in Sections 6.2.3 and 6.3.1.3, it is possible to express the internal control points of the adjacent Gregory of Bézier patches in terms of the coefficients of the premultiplying scalar functions. The expressions for the tangent vector and the two cross-boundary derivatives are (Section 6.2.1):

$$D\Gamma(v) = 3\sum_{i=0}^{2} \mathbf{s}_i\, B_i^2(v), \quad D\Phi(v) = 3\sum_{i=0}^{3} \mathbf{a}_i\, B_i^3(v), \quad D\Psi(v) = 3\sum_{i=0}^{3} \mathbf{b}_i\, B_i^3(v).$$

We premultiply $D\Gamma(v)$ with a cubic polynomial $\sum_{i=0}^{3} \gamma_i\, B_i^3(v)$, and $D\Phi(v)$ and $D\Psi(v)$ with quadratic polynomials $\sum_{i=0}^{2} \alpha_i\, B_i^2(v)$ and $\sum_{i=0}^{2} \beta_i\, B_i^2(v)$. The resulting sum is a polynomial of degree 5 in v that should be identically 0 for all values of v. This yields 6 equations for various degrees of v.

Two of the above equations would represent planarity constraints at the endpoints:

$$\alpha_0\,\mathbf{a}_0 + \beta_0\,\mathbf{b}_0 + \gamma_0\,\mathbf{s}_0 = 0,$$

$$\alpha_2\,\mathbf{a}_3 + \beta_2\,\mathbf{b}_3 + \gamma_3\,\mathbf{s}_2 = 0.$$

Assuming that the coefficients α_0, α_2, β_0, β_2, γ_0, and γ_3 satisfy the above constraints, the internal vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are expressed as follows:

$$\mathbf{a}_1 = -\frac{1}{3\alpha_0^2\beta_2^2 - 12\alpha_0\alpha_1\beta_1\beta_2 - 12\alpha_1\alpha_2\beta_0\beta_1 + 12\alpha_1^2\beta_0\beta_2 + 12\alpha_0\alpha_2\beta_1^2 - 6\alpha_0\alpha_2\beta_0\beta_2 + 3\alpha_2^2\beta_0^2} \times$$

$$\Big[((3\alpha_0\beta_2^2 - 12\alpha_1\beta_1\beta_2 - 3\alpha_2\beta_0\beta_2 + 12\alpha_2\beta_1^2)\gamma_1 + \beta_0(6\alpha_1\beta_2 - 6\alpha_2\beta_1)\gamma_2 + (\alpha_2\beta_0^2 - \alpha_0\beta_0\beta_2)\gamma_3)\mathbf{s}_0$$

$$+ ((2\alpha_0\beta_2^2 - 8\alpha_1\beta_1\beta_2 - 2\alpha_2\beta_0\beta_2 + 8\alpha_2\beta_1^2)\gamma_0 + \beta_0(12\alpha_1\beta_2 - 12\alpha_2\beta_1)\gamma_1 + (6\alpha_2\beta_0^2 - 6\alpha_0\beta_0\beta_2)\gamma_2 + (4\alpha_0\beta_0\beta_1 - 4\alpha_1\beta_0^2)\gamma_3)\mathbf{s}_1$$

$$+ (\beta_0(2\alpha_1\beta_2 - 2\alpha_2\beta_1)\gamma_0 + (3\alpha_2\beta_0^2 - 3\alpha_0\beta_0\beta_2)\gamma_1 + (6\alpha_0\beta_0\beta_1 - 6\alpha_1\beta_0^2)\gamma_2)\mathbf{s}_2$$

$$+ (2\alpha_0\alpha_1\beta_2^2 - 8\alpha_1^2\beta_1\beta_2 + 8\alpha_1\alpha_2\beta_1^2 - 2\alpha_2^2\beta_0\beta_1)\mathbf{a}_0 + (\beta_0(2\alpha_1\beta_2^2 - 4\alpha_2\beta_1\beta_2) + 2\alpha_0\beta_1\beta_2^2 - 8\alpha_1\beta_1^2\beta_2 + 8\alpha_2\beta_1^3)\mathbf{b}_0$$

$$+ (\alpha_0(4\alpha_1\beta_0\beta_1 + \alpha_2\beta_0^2) - \alpha_0^2\beta_0\beta_2 - 4\alpha_1^2\beta_0^2)\mathbf{a}_3 + (\alpha_0(4\beta_0\beta_1^2 - \beta_0^2\beta_2) - 4\alpha_1\beta_0^2\beta_1 + \alpha_2\beta_0^3)\mathbf{b}_3\Big],$$

$$a_2 = -\frac{1}{3\alpha_0^2\beta_2^2 - 12\alpha_0\alpha_1\beta_1\beta_2 - 12\alpha_1\alpha_2\beta_0\beta_1 + 12\alpha_1^2\beta_0\beta_2 + 12\alpha_0\alpha_2\beta_1^2 - 6\alpha_0\alpha_2\beta_0\beta_2 + 3\alpha_2^2\beta_0^2} \times$$

$$\Big[((3\alpha_3\beta_1^2 - 12\alpha_2\beta_2\beta_1 - 3\alpha_1\beta_3\beta_1 + 12\alpha_1\beta_2^2)\gamma_2 + \beta_3(6\alpha_2\beta_1 - 6\alpha_1\beta_2)\gamma_1 + (\alpha_1\beta_3^2 - \alpha_3\beta_3\beta_1)\gamma_0)s_3$$

$$+ ((2\alpha_3\beta_1^2 - 8\alpha_2\beta_2\beta_1 - 2\alpha_1\beta_3\beta_1 + 8\alpha_1\beta_2^2)\gamma_3 + \beta_3(12\alpha_2\beta_1 - 12\alpha_1\beta_2)\gamma_2 + (6\alpha_1\beta_3^2 - 6\alpha_3\beta_3\beta_1)\gamma_1 + (4\alpha_3\beta_3\beta_2 - 4\alpha_2\beta_3^2)\gamma_0)s_2$$

$$+ (\beta_3(2\alpha_2\beta_1 - 2\alpha_1\beta_2)\gamma_3 + (3\alpha_1\beta_3^2 - 3\alpha_3\beta_3\beta_1)\gamma_2 + (6\alpha_3\beta_3\beta_2 - 6\alpha_2\beta_3^2)\gamma_1)s_1$$

$$+ (2\alpha_3\alpha_2\beta_1^2 - 8\alpha_2^2\beta_2\beta_1 + 8\alpha_2\alpha_1\beta_2^2 - 2\alpha_1^2\beta_3\beta_2)a_3 + (\beta_3(2\alpha_2\beta_1^2 - 4\alpha_1\beta_2\beta_1) + 2\alpha_3\beta_2\beta_1^2 - 8\alpha_2\beta_2^2\beta_1 + 8\alpha_1\beta_2^3)b_3$$

$$+ (\alpha_3(4\alpha_2\beta_3\beta_2 + \alpha_1\beta_3^2) - \alpha_3^2\beta_3\beta_1 - 4\alpha_2^2\beta_3^2)a_0 + (\alpha_3(4\beta_3\beta_2^2 - \beta_3^2\beta_1) - 4\alpha_2\beta_3^2\beta_2 + \alpha_1\beta_3^3)b_0\Big],$$

$$b_1 = -\frac{1}{3\alpha_0^2\beta_2^2 - 12\alpha_0\alpha_1\beta_1\beta_2 - 12\alpha_1\alpha_2\beta_0\beta_1 + 12\alpha_1^2\beta_0\beta_2 + 12\alpha_0\alpha_2\beta_1^2 - 6\alpha_0\alpha_2\beta_0\beta_2 + 3\alpha_2^2\beta_0^2} \times$$

$$\Big[((3\beta_0\alpha_2^2 - 12\beta_1\alpha_1\alpha_2 - 3\beta_2\alpha_0\alpha_2 + 12\beta_2\alpha_1^2)\gamma_1 + \alpha_0(6\beta_1\alpha_2 - 6\beta_2\alpha_1)\gamma_2 + (\beta_2\alpha_0^2 - \beta_0\alpha_0\alpha_2)\gamma_3)s_0$$

$$+ ((2\beta_0\alpha_2^2 - 8\beta_1\alpha_1\alpha_2 - 2\beta_2\alpha_0\alpha_2 + 8\beta_2\alpha_1^2)\gamma_0 + \alpha_0(12\beta_1\alpha_2 - 12\beta_2\alpha_1)\gamma_1 + (6\beta_2\alpha_0^2 - 6\beta_0\alpha_0\alpha_2)\gamma_2 + (4\beta_0\alpha_0\alpha_1 - 4\beta_1\alpha_0^2)\gamma_3)s_1$$

$$+ (\alpha_0(2\beta_1\alpha_2 - 2\beta_2\alpha_1)\gamma_0 + (3\beta_2\alpha_0^2 - 3\beta_0\alpha_0\alpha_2)\gamma_1 + (6\beta_0\alpha_0\alpha_1 - 6\beta_1\alpha_0^2)\gamma_2)s_2$$

$$+ (2\beta_0\beta_1\alpha_2^2 - 8\beta_1^2\alpha_1\alpha_2 + 8\beta_1\beta_2\alpha_1^2 - 2\beta_2^2\alpha_0\alpha_1)b_0 + (\alpha_0(2\beta_1\alpha_2^2 - 4\beta_2\alpha_1\alpha_2) + 2\beta_0\alpha_1\alpha_2^2 - 8\beta_1\alpha_1^2\alpha_2 + 8\beta_2\alpha_1^3)a_0$$

$$+ (\beta_0(4\beta_1\alpha_0\alpha_1 + \beta_2\alpha_0^2) - \beta_0^2\alpha_0\alpha_2 - 4\beta_1^2\alpha_0^2)b_3 + (\beta_0(4\alpha_0\alpha_1^2 - \alpha_0^2\alpha_2) - 4\beta_1\alpha_0^2\alpha_1 + \beta_2\alpha_0^3)a_3\Big],$$

$$b_2 = -\frac{1}{3\alpha_0^2\beta_2^2 - 12\alpha_0\alpha_1\beta_1\beta_2 - 12\alpha_1\alpha_2\beta_0\beta_1 + 12\alpha_1^2\beta_0\beta_2 + 12\alpha_0\alpha_2\beta_1^2 - 6\alpha_0\alpha_2\beta_0\beta_2 + 3\alpha_2^2\beta_0^2} \times$$

$$\Big[((3\beta_3\alpha_1^2 - 12\beta_2\alpha_2\alpha_1 - 3\beta_1\alpha_3\alpha_1 + 12\beta_1\alpha_2^2)\gamma_2 + \alpha_3(6\beta_2\alpha_1 - 6\beta_1\alpha_2)\gamma_1 + (\beta_1\alpha_3^2 - \beta_3\alpha_3\alpha_1)\gamma_0)s_3$$

$$+ ((2\beta_3\alpha_1^2 - 8\beta_2\alpha_2\alpha_1 - 2\beta_1\alpha_3\alpha_1 + 8\beta_1\alpha_2^2)\gamma_3 + \alpha_3(12\beta_2\alpha_1 - 12\beta_1\alpha_2)\gamma_2 + (6\beta_1\alpha_3^2 - 6\beta_3\alpha_3\alpha_1)\gamma_1 + (4\beta_3\alpha_3\alpha_2 - 4\beta_2\alpha_3^2)\gamma_0)s_2$$

$$+ (\alpha_3(2\beta_2\alpha_1 - 2\beta_1\alpha_2)\gamma_3 + (3\beta_1\alpha_3^2 - 3\beta_3\alpha_3\alpha_1)\gamma_2 + (6\beta_3\alpha_3\alpha_2 - 6\beta_2\alpha_3^2)\gamma_1)s_1$$

$$+ (2\beta_3\beta_2\alpha_1^2 - 8\beta_2^2\alpha_2\alpha_1 + 8\beta_2\beta_1\alpha_2^2 - 2\beta_1^2\alpha_3\alpha_2)b_3 + (\alpha_3(2\beta_2\alpha_1^2 - 4\beta_1\alpha_2\alpha_1) + 2\beta_3\alpha_2\alpha_1^2 - 8\beta_2\alpha_2^2\alpha_1 + 8\beta_1\alpha_2^3)a_3$$

$$+ (\beta_3(4\beta_2\alpha_3\alpha_2 + \beta_1\alpha_3^2) - \beta_3^2\alpha_3\alpha_1 - 4\beta_2^2\alpha_3^2)b_0 + (\beta_3(4\alpha_3\alpha_2^2 - \alpha_3^2\alpha_1) - 4\beta_2\alpha_3^2\alpha_2 + \beta_1\alpha_3^3)a_0\Big].$$

Appendix 2: Uci Manual

NAME

uci – interactive UNICUBIX environment

SYNOPSIS

uci [arguments, options]

DESCRIPTION

Uci is an editing, viewing, and designing tool for UNICUBIX objects. It provides most of the *ugi* capabilities for objects with curved edges and curved surface patches.
The term *current scene* in this manual refers to all UNICUBIX objects that were either read in or created by one of the modifiers since the beginning of the session or since the last **clear** command.
The following options can be used:

-fi *filename*

Use file filename to find *uci* commands. Each command with its arguments must appear on a single line. This enables setting initial parameters, or even running a whole session as a batch job. Lines beginning with a '#' are ignored.

-e Echo the commands from the command-file. Comment lines are echoed too.

After processing the commands from the input file, *uci* will prompt you with *uc>*.
You may now enter commands. Command names are single words, which may be abbreviated to any unique prefix. The commands can be divided into seven logical groups: I/O, Display, Scene, Curve Construction, Patch Boundary Construction, Patch Construction, and Miscellaneous. The following sections describe each group.
Uci records the session in a log file called *uci.log* in the current directory.

I/O Commands

• **read** *filename1* [xform-options1] *filename2* [xform-options2] ...

Read is used to read in UNICUBIX scenes from files, optionally transforming them, and adding them to the *current scene*. Each set of transformations applies only to the filename preceding it, i.e. transformations do not accumulate. See description of *xform* for details of the options. Since everything becomes part of the same scene, name conflicts may occur between two objects of the same type and same name from different files. *Uci* handles those conflicts correctly in most cases. When writing such a scene to a file use the **newlabels**

command first.
If the *filename* is missing, standard input is read. Type your UNICUBIX statements in, and end input by typing **q**.
Similarly, the special filename << tells *uci* to read the next lines as UNICUBIX statements, until the **q** statement. This is useful in writing *uci* command files (following the spirit of csh scripts).

EXAMPLE: **read** cube -rx 20 -tx 3 cube -rx -20 -tx -3

- **smooth** [options] *filename1* [options1] *filename2* [options2] ...

  Reads in a scene from a file in UNIGRAFIX or UNICUBIX format and replaces edges by cubic curves and faces by curved patches unless specified otherwise.

  Smooth computes unspecified control points of curved borders or edges. All faces are converted into patches, except for specified flat faces **F**. Edges are replaced by curved borders, unless specified otherwise.

  -b *factor*
  Change the bulge of edges (and thus patches) to *factor*.

  -n *normethod* **-t** *tanmethod* **-v** *velmethod*
  Use the specified method for determining normals, tangent directions, and velocities. Refer to command **geometry** for a list of available methods.

  remaining options
  as in **read**.

  EXAMPLE: **smooth** -b 0.5 -n 2 -t 1 cube -ry 5 -tx 20 -sy 2

- **write** [options] *filename*

  Writes the current scene to the named file in UNICUBIX format. If *filename* is not specified, standard output is used.

  -h Writes the scene in a hierarchical format. This is the default option.

  -f Writes a *flattened* scene (no hierarchy).

  -C Writes a *compact* scene: vertices are written only if actually used by a face or a wire.

  -i Writes illumination sources to *filename*.il .

  -v Writes viewing parameters to *filename*.vp .

  -d Prints the names of loaded definitions.

  -ae Attach plane equations to top level faces.

-ai Attach illumination values to top level faces.

-s Writes all previous commands to a *uci* command file. This file can then be used with the **-fi** option to *uci* to reproduce the current session. (In effect, the logfile is copied to the specified filename, with a time stamp).

-g Write out Gregory patches for Ricoh Corp. contouring program. Only quadrilateral patches are written out. The program *gc* can then be used to produce the object contours, perpendicular to one of the coordinate axes in the UNIGRAFIX wireframe format. The format of the *gc* command is:

gc [**-a** *axis*] [**-d** *dh*] [**-l** *length*] *filename*

where *axis* can be 1 (X), 2 (Y), or 3 (Z). The contours will be perpendicular to this axis. Default: 3 (Z axis). The distance between the contours is controlled by *dh*. Default: 0.5. Finally, *length* is the maximum length of a line segment in a contour. Default: 1.0. The file *filename* (produced by *uci* **write -g** command) contains the coordinates of the control points of the Gregory or Bézier patches. This is a sample dialog to obtain a wireframe:

% uci
uc> smooth *file1*
uc> patch
uc> write -g *file2*
uc> quit
% gc -dh 0.1 -l 0.2 *file2* > *file3*
% ugplot < *file3*

Here *file1* is an input *uci* file, *file2* contains the coordinates of the control points of Gregory or Bézier patches, and *file3* is a UNIGRAFIX wireframe file.

-b Write out Bézier patches for UW Seattle Tess program. Triangular Bézier patches only are written. The output of this command can then be used by the *Tess* program:

% uci
uc> smooth *file1*
uc> patch -b
uc> write -b *file2*
uc> quit
% Tess < *file2*

Here *file1* is an input *uci* file, and *file2* contains the coordinates of the control points of the triangular Bézier patches.

Only one output format (either scene description or set of commands) can be written to *filename*, so only the last one specified will have effect. (To write out the scene both as a scene file and as a *uci* command file, use the **write** command twice).

EXAMPLE: **write** -i -f sceneOut

- **source** [options] *filename*

Reads in *uci* commands from the named file and executes them in a batch manner. This command is very useful for storing the modifications of the original object.

-e Echo each command before execution.

EXAMPLE: **source** uciCmdLog

Display Commands

- view [options]

Sets the *viewing parameters* for the current scene. Once set, the parameters remain in effect for the rest of the session, unless reset by another **view** command. They can be temporarily overridden by **display** options (see below) for a certain scene display. If you change from a perspective view (**-ep**) to an orthogonal view (**-ed**) or vice-versa, all parameters that are not relevant to the current view are saved but ignored.
All **display** options can be specified. An additional option is:

-p Prints the current viewing parameters. (Note that the set of viewing parameters is updated only *after* the **view** command, so **-p** will not reflect any new parameters set in the current **view**).

EXAMPLE: **view** -v -sa -ab -ed 3 2 -7 -vr 22

- **display** [options]

Displays the current scene on the requested device. All *ugdisp* options can be used, except for **-fi** for input file, since input is the current scene. Viewing parameters are added to the parameters that were set by **view** (see above), and override them in case of conflict. They take effect only for this one display. See *Ugdisp (UG)* man page for option details.

EXAMPLE: **display** -sg -st

- **illuminate** [options]

Modifies the set of illum sources and the illumination of the current scene. Two shading models can be used: uniform shading for each face, or smooth shading of the whole scene (using *Gouraud* shading). In addition, *fog* options can be specified to *fade* to a white or black background.

Options to modify the set of illum sources:

-a *id intensity* [x y z]

Add an illumination source. The new source may be *directional* or *ambient* in which case the direction vector is not specified. *intensity* should be in the range (0,1].

-r *id* Remove the specified source.

-p Print the list of current illum sources.

Options to modify illumination of the scene:

-i Illuminate the scene with uniform face shading.

-g Illuminate the scene with Gouraud shading.

-fw x y z *radius1 radius2*

Fade against white background in the interval *radius1-radius2* from the *fog point* **x y z.**

-fb x y z *radius1 radius2*

Fade against black background in the interval *radius1-radius2* from the *fog point* **x y z.**

The two shading models can coexist in the data structure since *uniform* data is kept in faces, and *smooth* data is kept in vertices. Thus, once any shading model was calculated (either implicitly by a **display** command, or explicitly by an **illuminate** command) it remains and can be used. To update any model after an illum source was added/removed, **-i** or **-g** should be called.

EXAMPLE: **illuminate** -a moon 0.4 -3 10 -15 -p -i

Scene commands

• **xform** [options]

Transforms the whole scene. (with optional transformation of illum sources). Hierarchical structure is retained, and if the scene is written with the **-h** option (see **write**) these transformations are appended to the end of the xform-lists of top-level instances and arrays.

-tx, -ty, -tz *amount*

Translate scene by *amount* in the specified direction.

-rx, -ry, -rz *angle*

Rotate scene around specified axis by *angle* (in degrees).
(positive angles cause counter-clockwise rotation when viewed in direction of positive axis).

-sx, -sy, -sz *factor*
Scale the scene by *factor* in the appropriate dimension.

-sa *factor*
Scale the scene by *factor* in all three dimensions.

-mx, -my, -mz
Mirror specified coordinates about the origin.

-ma Mirror all coordinates about the origin.

-M3 *3x3 matrix*
Use *one to nine numbers* as transformation matrix.

-M4 *4x4 matrix*
Use *one to sixteen numbers* as transformation matrix.

-xl Transform coordinates of light sources as well.

-px Print the list of specified transformations.

-pm Print the total transformation matrix.

EXAMPLE: **xform** -sa .7 -ry 23 -tx 10 -M3 1 0 2 -1 9 1 -xl -pm

• **newlabels** [options]

Gives new sequential labels to scene objects. Used mainly to convert very long names to short ones, and to avoid naming conflicts before writing a scene to a file.

-h Keep hierarchical structure. Objects in definitions retain their original labels, while top-level objects get new labels. This is the default option. As in **write**, **-h** is not allowed if a modifier like **curvedge** or **patch -p** was called.

-f Scene is flattened, and hierarchy is lost. All objects get new labels.

• **clear** [options]

Clears the current scene, and allows you to start a new scene.

-s Clear only scene description.

-t Like **-s** but only *top level* objects are cleared. All loaded definitions remain, and can be referenced and instantiated by future **read** commands.

-i Clear only illumination sources.

-v Clear only viewing parameters.

If no options were specified, the default is to clear everything. Your verification is then requested.

Curve Construction Commands

• **gcurve** [options]

Constructs interpolating curves from a single specified wire. Expects that every vertex (except for the endpoints) has exactly two neighbors.

-m *number number*

Select a procedural method for normals and velocities. Available normal methods are:

1. Angle bisector (default)
2. Angle bisector with special rule for collinear segments
3. Catmull–Rom method (directly weighted normals)
4. Inversely weighted normals
5. Inversely weighted edges
6. Directly weighted edges
7. Directly weighted normals for small angles and inversely weighted edges for large angles (top of the diamond)
8. Inversely weighted normals for small angles and directly weighted edges for large angles (bottom of the diamond)

Available velocity methods are:

1. Edge length
2. Edge projection
3. Average of edge length and projection
4. Catmull–Rom method (half chord length)
5. Average of side lengths
6. Average of edge projections
7. Average of methods 1 and 4 (default)
8. Average of methods 1 and 5
9. Average of methods 1 and 6

10. Average of methods 2 and 4
11. Average of methods 2 and 5
12. Average of methods 2 and 6
13. Average of methods 3 and 4
14. Average of methods 3 and 5
15. Average of methods 3 and 6

-e *number number*

Select a method for normals and velocities at the endpoints (for open curves only). Available endpoint normal methods are:

1. Symmetric method (default)
2. Extension of previous Bézier control point
3. Side difference method

Available velocity endpoint methods are:

1. Symmetric method (default)
2. Average of edge length and projection

-b *factor*

Set global bulge to specified factor.

-k Construct curvature continuous curve.

EXAMPLE: **gcurve** -m 3 4 -e 3 2 -k

Commands for Construction of Boundaries of Surface Patches

- **bulge** *factor*

Change the bulge of cubic curves to the given factor. Default is 1.0; *factor* outside of [0,1] may produce loops and self-intersections.

EXAMPLE: **bulge** 1.2

- **vnormal** *vertexid [x y z]*

Print the normal at selected vertex or, if the vector is supplied, set the normal to it. All the tangent directions at selected vertex are projected onto newly defined tangent plane.

Velocities remain unaffected.

EXAMPLE: **vnormal** A 0 0.5 -0.5

- **geometry** [options]

Sets the methods for determination of normals, tangent directions, and velocities of the boundary curves of interpolating patches.

-n *number*
Select a method for vertex normal determination.

-t *number*
Select a method for determination of tangent directions.

-v *number*
Select a method for determination of velocities.

Available methods for vertex normals are:

1. Opposite edge method, based on G1 curves (default)
2. Average of face normals, weighted by face angle at the vertex
3. Average of face normals, weighted by face area
4. Average of face normals
5. Average of face normals, weighted by inverse face area
6. Average of face normals, weighted by product of face angle and face area
7. Average of face normals, weighted by product of face angle and inverse face area

Available methods for tangent directions are:

1. Opposite edge method, based on G1 curves (default)
2. Projection method
3. Planar boundaries

Available methods for velocity are:

1. Edge length velocity (default)
2. Opposite edge method, based on G1 curves
3. Generalized average of Catmull–Rom and edge length methods

EXAMPLE: **geometry** -n 2 -t 2

- **curvedge** [options]

Approximates the cubic curves in the current scene with piecewise linear wires trains. Faces are deleted.

-s *number*
Number of linear segments to represent each edge. Default: 10

-p Add representation of Bézier points of edges linked to their vertices.

-i Create interior curves if Bézier subdivision was used (command **patch -b** has to be executed before).

-c *number*
This option must be specified if an interpolating curve is to be constructed. If *number* is 0, then the original wire will be deleted; otherwise, the defining polygon will remain in the structure together with the curve.

-k This option is valid for interpolating curves only (**gcurve** command has to be executed before; **-c** must also be specified). Constructs curvature plot of the curve rather than the curve itself.

EXAMPLE: **curvedge** -s 6 -p

- **normals** [options]

To each vertex it adds a wire representation of a normal assigned to the vertex.

-l *number*
Length of wires representing the normals. Default: 1.0.

-c Clear previously displayed normals.

-i This option must be specified for interpolating curves (constructed with the **gcurve** command).

EXAMPLE: **normals** -l .7

Patch Construction Commands

- **patch** [options]

Computes and stores control points of Gregory or Bézier patches for non-planar faces. Default is to compute Gregory patches.

-b Creates 5 Bézier patches for each quadrilateral face, and 3 patches for each triangular face.

-B Adds new vertices and curved borders corresponding to interior joints and boundaries of Bézier subdivision.

-p Creates a wire representation of control points linked to their edges.

EXAMPLE: **patch** -b

- **shape** [options] *bulge tilt shear*

Sets shape parameters on all edges or on a selected edge by specifying bulge, tilt, and shear coefficients.

-e *vert1 vert2*

Set shape parameters on the edge between *vert1* and *vert2* only.

EXAMPLE: **shape** -e A B 1 5 0

- **net** [options]

Create a faceted face representation of non-planar patches in UNIGRAFIX format. By default, quadrilateral facets are created for quadrilateral faces, and triangular facets for triangular faces.

-s *number*

Number of segments each edge is subdivided into for rendering. Default: 5. Thus, quadrilateral and triangular patches will be represented with s^2 facets.

-t Create triangular facets even for quadrilateral faces. This is useful for rendering routines that can deal with planar facets only.

-f Construct a net on the original faces of the polyhedron, rather than on the interpolating patches.

EXAMPLE: **net** -s 4 -t

Miscellaneous Commands

- **kurvature** [options]

Print out curvature variation along patch boundaries. Patches have to be Bézier. This is useful for comparison of various methods for boundary curve construction.

-s *number*

Number of equally spaced sample points along the boundary to evaluate the curvature. Default: 10.

-b *number*

Select boundary type. The following types are available (default is 1):

1. Exterior boundaries only
2. Interior boundaries only
3. Both exterior and interior boundaries

EXAMPLE: **kurvature** -b 2

- **help** *command-name*

Command-name is a unique substring of one of the commands. Description of the command and its options is printed. Without any argument it prints the menu of commands.

EXAMPLE: **help** help

- **quit**

Ends the session.

- **!** *string*

C-shell escape. *String* contains a csh command.

FILES

~ug/ug2/new/uci
~ug/ug2/src/uci

SEE ALSO

ugi (UG), ugdisp (UG)

DIAGNOSTICS

Checks input files for syntax errors.

BUGS

Yet to be reported.

AUTHORS

Lucia Longhi, Nachshon Gal, Leon Shirman