TEXTURED SURFACE MODELING USING BÉZIER TRIANGLES

by

Jesse Chaim Reiter

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Textured Surface Modeling Using Bézier Triangles

Master of Science degree, 1996

Jesse Chaim Reiter

Graduate Department of Computer Science

University of Toronto

Abstract

Traditional surface modelers construct surfaces that are pieced together from a collection of plain rectangular surface elements. This approach suffers from two major drawbacks. First, arbitrary objects may not admit a tessellation using non-degenerate four-sided patches. Second, once constructed, these surfaces have a stark, plastic appearance. These problems can be overcome by constructing surfaces piecewise using triangularly shaped surface elements which are individually textured. This thesis presents a geometrically intuitive formulation of the triangular Bézier surface patch, and develops methods by which such a patch can be individually textured. The issues involved in the construction of a large, textured surface from a piecewise collection of textured Bézier triangles are also discussed. Finally, an interactive tool for modeling such surfaces is presented.

Acknowledgements

Now that the crunch is over, the lineups at slate and mom have subsided and the toner has been replenished, I would like to thank some people who made this work possible.

First, I am indebted to my supervisor Eugene Fiume for his time and encouragement invested in me. His patience and enthusiasm have been an inspiration throughout this process.

I would also like to thank my co-supervisor James Stewart for his patience in discussing the topic, and for his helpful comments during the writing of this thesis.

My great appreciation and thanks to all the people at DGP for providing such a wonderful and friendly environment, without whom this thesis would surely have been completed long ago.

Finally, I am forever indebted to my wife and friend Nicola Lisus for putting up with the endless hours of research and writing. Her presence and love have contributed much to this work, which would be much lacking without her helpful comments, and unwavering support.

Contents

List of Figures

Chapter 1

Introduction

The field of Computer Aided Geometric Design (CAGD) has developed considerably since its inception in the late 1950's. At that time, hardware had just become available that would permit the the milling of three dimensional objects from raw materials, a process which has come to be known as Computer Aided Manufacture (CAM). The use of computers in the manufacturing process made it necessary to develop a formulation by which the milled objects could be concisely described, and from which the objects could be precisely constructed. Properties of geometric objects had long since been studied in the form of parametric curves and surfaces. The theory of these was well understood within the realm of differential geometry, but not until this time were the results applied to Computer Aided Design (CAD). Throughout the 1960's major breakthroughs in the theory were spawned by people such as P. Bézier, P. de Casteljau, and S. Coons which brought CAGD by the mid 1970's to a field in its own right [Far93].

The fundamental problem of CAGD is the description of the geometry of three-dimensional objects in concise form. The geometric model must meet complex and often conflicting requirements bestowed upon it by the user for design considerations, and by the software for reasons of consistency and compactness. The widespread use of computer graphics technology in fields such as scientific research and entertainment has put additional pressure on geometric models, for surfaces in CG are expected to look "real". Plain geometric models are deemed unacceptable by many applications.

Thus in addition to providing clear geometric descriptions, a design tool must be equipped with the facilities that will enable texture to be added to the images.

In this thesis, a mathematical formulation for surfaces will be developed, which is geometrically intuitive and which lends itself well to texture mapping. Several methods for applying texture will also be developed. Finally, the implementation of an interactive modeling tool which offers the capacity to design fully textured three dimensional surfaces will be discussed.

1.1 Preliminaries

Strictly speaking, *points* in three-dimensional Euclidean space, and *vectors* in the three-dimensional real vector space $\mathbf{R}^3$ are distinct entities. Nonetheless, because they are essentially equivalent, as their identical notation suggests, the distinction between the two will be dropped. At times when the unique identification will be necessary, the distinction will be made explicit.

In what follows, let $\mathbf{x}_0, \ldots, \mathbf{x}_n \in \mathbf{R}^3$ be a collection of points in space, and let $\alpha_0, \ldots, \alpha_n \in \mathbf{R}$ be a collection of scalars. A *linear combination* of the points $\{\mathbf{x}_i\}$ is any weighted sum of the form

$$\mathbf{p} = \sum_{i=0}^{n} \alpha_i \mathbf{x}_i.$$

When the weights α_i sum to unity, the weighted sum of points is called an *affine combination*. A special kind of affine combination occurs when, in addition to summing to unity, the weights are all non-negative. In this case the combination is called *convex*.

Convex combinations are very important and will be used extensively in the ensuing chapters. Their significance stems from the following. Suppose that the set of points $\{\mathbf{x}_i\}$ is lying somewhere in space, and imagine that a rubber sheet is wrapped tightly around the set. The rubber sheet now encloses some volume or area called the *convex hull* of the set. The importance of convex combinations is that the convex hull of a collection of points is formed by the set of all their convex combinations. Thus a convex combination of points always lies within their convex hull.

Affine maps may be informally described as maps that respect the spatial characteristics of points. For example, affine maps send straight lines to straight lines, midpoints of lines to midpoints of lines, and circles to circles. More formally, affine maps can be defined in terms of affine combination. A map $\varphi : \mathbf{R}^3 \to \mathbf{R}^3$ is said to be *affine* if it preserves affine combinations of points, that is, if

$$\varphi\left(\sum_{i=0}^{n} \alpha_i \mathbf{x}_i\right) = \sum_{i=0}^{n} \alpha_i \varphi(\mathbf{x}_i), \tag{1.1}$$

when $\alpha_i \geq 0$ for all i, and $\sum_i \alpha_i = 1$. Although they respect some geometric attributes of points, affine maps do not necessarily preserve lengths and angles. Common examples of affine maps are rotations, translations, scalings, and parallel projections.

In geometric terms, a *simplex* is an elementary building block of some vector space, in the sense that it is comprised of the smallest set of points that can generate the space. For example, a 2-simplex in the plane is a triangle, a 3-simplex in $\mathbf{R}^3$ is a tetrahedron. An important property of affine maps is that every such map is uniquely determined by a simplex and its image. That is, if $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbf{R}^3$ is a non-degenerate tetrahedron, the affine map φ is completely determined by the tetrahedron $\varphi(\mathbf{a}), \varphi(\mathbf{b}), \varphi(\mathbf{c}), \varphi(\mathbf{d}) \in \mathbf{R}^3$.

1.2 Survey of Techniques

The basic problem dealt with in CAGD is that of constructing curves or surfaces from given sets of data points. This is essentially the same as fitting a curve or surface to a given set of points, a classic problem from the field of approximation theory. The solutions generally fall into one of two categories: those in which all the points are included or *interpolated*, and those in which they are only approximated. Several solutions to the problem of interpolative curve fitting will be discussed, and their extension to interpolative surface fitting mentioned.

The curve problem can be stated as follows. Given $n+1$ points $\mathbf{x}_0, \ldots, \mathbf{x}_n \in \mathbf{R}^3$ that are associated with $n+1$ parameter values $t_0, \ldots, t_n \in \mathbf{R}$, a polynomial $\mathbf{p}(t)$ is sought that will satisfy

$$\mathbf{p}(t_i) = \mathbf{x}_i. \tag{1.2}$$

There are infinitely many polynomials that satisfy Equation (1.2). One straightforward method of finding such a polynomial is by using *Lagrange interpolation*. Suppose a set of n functions $L_i^n(t)$ were provided such that $L_i^n(t_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta (*i.e.*, $\delta_{ij} = 1$ iff $i = j$, and $\delta_{ij} = 0$ otherwise). Then the polynomial $\mathbf{p}(t) = \sum_{i=0}^{n} L_i^n(t)\mathbf{x}_i$ clearly interpolates all the points. The functions $L_i^n(t)$ are called the *Lagrange polynomials* and are given by

$$L_i^n(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(t - t_j)}{(t_i - t_j)}.$$

Suppose, now, that the data does not consist only of physical points, but includes tangent vectors at the points as well. This leads to a different interpolation scheme that is more useful than the Lagrange approach. Consider the case when the data given consists of two points $\mathbf{x}_0, \mathbf{x}_2$ and their corresponding tangent vectors $\mathbf{x}_1, \mathbf{x}_3$. *Hermite interpolation* is used to define the cubic polynomial $\mathbf{p}(t) = \mathbf{A}t^3 + \mathbf{B}t^2 + \mathbf{C}t + \mathbf{D}$ that satisfies

$$\mathbf{p}(0) = \mathbf{x}_0, \qquad \dot{\mathbf{p}}(0) = \mathbf{x}_1, \qquad \mathbf{p}(1) = \mathbf{x}_2, \qquad \dot{\mathbf{p}}(1) = \mathbf{x}_3.$$

These four constraints are sufficient to uniquely resolve the four coefficients $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ and determine the cubic polynomial $\mathbf{p}$.

The advantage of the Lagrange and Hermite methods is that all the data is packaged and interpolated by one polynomial. The disadvantage is that the degree of this polynomial is directly proportional to the size of the data set $\{\mathbf{x}_i\}$. A large data set is therefore interpolated by a polynomial of a high degree. This leads to various problems of numeric instability due to the propagation of errors through the evaluation process on a finite precision machine. A more inherent problem is that polynomials of high degree tend to "wiggle" around and thus produce "bumpy" curves.

These difficulties can be overcome by employing piecewise curves. Such curves are defined by a family of polynomials, each having a bounded domain. Every curve segment is typically formed by a cubic polynomial. Cubics are chosen because they are the curves of lowest degree that offer sufficient control over the curvature and smoothness properties of the interconnected segments.

A classic example of a curve defined by a piecewise family of polynomials is a *spline*. The *cubic spline* is constructed by defining a cubic curve segment for each pair of neighboring points $\mathbf{x}_i, \mathbf{x}_{i+1}$. If the polynomial is required to be C^2 at the endpoints of each segment, one has sufficient information to determine the interpolating spline up to the boundary conditions at its endpoints. Additional information is needed at the extremal points because no continuity constraints are enforced there. Details of this method can be found, for example, in [HL93]. Note that when $\Delta t = t_{i+1} - t_i = 1$ for all i, the cubic spline is just a piecewise Hermite interpolant.

Another type of spline that is extremely popular is known as the *B-spline*. This spline offers the same level of continuity as the cubic spline, but is considerably easier to compute. Using this formulation, the data are not necessarily interpolated, and it is possible to set the desired level of continuity between adjacent curve segments. Several types of B-splines have been studied; most notable are the non-uniform rational B-splines, also known as NURBS.

Splines are advantageous because each data point has a limited scope of influence and, therefore, local changes to a data point cause only local changes to the curve. Another significant advantage is that the composite curve is smooth throughout its entire range. The disadvantage of splines is that the curves are difficult to control and often lack an intuitive relationship to their data points.

The most common method of extending curves to surfaces is by defining two curve segments in orthogonal parametric directions. This leads to the development of rectangular, or tensor-product patches. These patches will be discussed in greater detail in Chapter 3, but suffice it to say now that this method of extension does not necessarily follow the most natural approach. It has been suggested by de Casteljau [dC92] that this is perhaps a result of human socialization factors more than anything else, as children grow up with a bias towards right-angled objects such as *dominoes*, tables, and doors. In any event, surfaces fitted to large data sets suffer from the same inherent drawbacks as do curves. High degree polynomials simply do not produce very useful surfaces. Therefore, as in the case of curves, the piecewise approach is adopted.

The extension of the Hermite interpolation method from curves to surfaces produces what has become knows as the *Coons patch*. Just as cubic Hermite curves required the position and tangent vectors of two points, the *bi*-cubic Coon's patch, being cubic in each of the two orthogonal parameters, requires this information for both curves. In a similar fashion, the cubic B-spline curve is extended to the bicubic B-spline patch. It is worthwhile mentioning that because a bicubic patch is represented by a bicubic polynomial which is uniquely determined by its sixteen coefficients, every such patch must resolve sixteen data-point constraints. All bicubic patches are therefore equivalent in the sense that the representation of a given patch in one form can always be translated to an equivalent representation in another.

1.3 Motivation

Although the global smoothness properties of B-splines seem to make them best suited for general surface representation, they are in fact detrimental in that they produce surfaces which are *too* smooth. Mathematically well behaved, B-spline surfaces tend to resemble large "blobs" of plasticine or putty, thus restricting their applications in general settings. They are well suited for modeling regular geometric objects such as spheres and tori, but deformed as they may be, objects constructed from one large B-spline are of limited practical value. Free-form modeling is best orchestrated through the use piecewise surfaces.

This work is motivated by the desire to develop a free-form sculpting tool that is geometrically based, and which frees the user from the need to deal with the underlying mathematics of the models. Other design tools have previously been developed with this purpose in mind. Indeed, most make use of piecewise surfaces whose tessellation elements are rectangular as those previously described. This is, however, rather curious because it is provably impossible to pave arbitrary objects with non-degenerate rectangular patches. Some surface elements must therefore be reduced to degenerate three-sided ones, thus opening the door to singularities and robustness problems. The question is then asked why triangular shaped surface elements are not used in the first place. It seams interesting that, despite its lack of superiority, the four-sided approach has prevailed.

Another issue deserving attention is that of texture mapping. No surface modeler would be complete without the realism that comes with the use of texture. The structure of triangular patches will be exploited for the benefit of piecewise texturing of the composite surface.

1.4 Overview

In this paper, a triangular surface element known as the Bézier triangle will be presented and will be used as a building block for modeling general surfaces. Various methods of boosting the realism of the models using texture mapping techniques will also be described.

In the next chapter, a geometrically intuitive mathematical formulation for the design of space curves will be developed. A variety of tools used for assisting their design will also be described. In Chapter 3 the scheme will be extended to triangular surface patches, and it will be explicitly shown how these can be attached to one another in a cohesive fashion. Several methods of sampling the surface and computing each surface point will then be presented, and their computational complexity discussed. In Chapter 4, the discussion will diverge to the issue of texture mapping. Two techniques by which texture can be applied to synthetic objects will be developed. The adaptation of these methods to piecewise surfaces will be discussed. In Chapter 5, several issues will be explored regarding the construction and the texturing of large surfaces comprised of tessellated triangular elements. Finally, in Chapter 6 the user interface to the modeler will be presented, and several implementation details discussed.

Chapter 2

Bézier Curves and the de Casteljau Algorithm

From a designer's standpoint, drawn objects are apprehended geometrically rather than mathematically. Points on an object are designated relative to others in terms of their functionality: A point can be a corner, an endpoint of a line, the midpoint of two others, and so on. The space in which these objects lie is arbitrary, as it possesses no preferred direction or coordinate system. The location of an object is determined with respect to its peers rather than to some fixed point of reference.

Analytic representation, conversely, is mathematically succinct and concise. It provides a means of specification from which the modeled object can be precisely reconstructed in any place, a task of paramount importance to the implementor. Unfortunately, this analytical clarity often obscures the underlying visual geometry.

The algorithm below bridges the two extremes, offering the requisite geometric intuition for the designer while preserving the analytical precision necessary for computation. It is one of the most fundamental in the theory of parameterized curves, yet surprisingly simple. The intuition driving the algorithm is supplied in the first section of this chapter. In the second, a formal description is provided, followed by a mathematical formulation. The final sections deal with some specific issues.

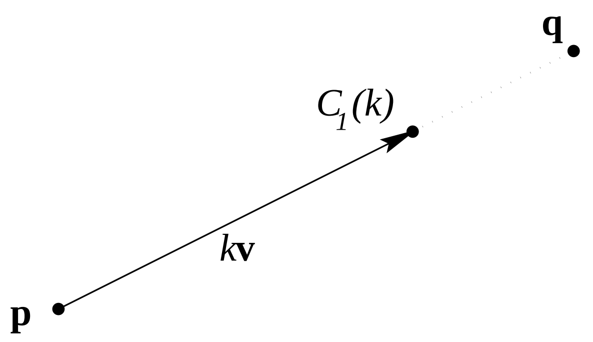

Figure 2.1: Linear interpolation of two points

2.1 Intuition

Consider two points in space, $\mathbf{p}$ and $\mathbf{q}$. The vector $\mathbf{v}$ leading from the former to the latter is given by $\mathbf{v} = \mathbf{q} - \mathbf{p}$. It may be scaled and its direction reversed through multiplication by a *scaling factor* or *scalar* $k \in \mathbf{R}$, producing the shortened or elongated version $k\mathbf{v} = k(\mathbf{q} - \mathbf{p})$. The magnitude and direction of this new vector is determined respectively by the magnitude and sign of k.

The scaled vector may be attached to the point $\mathbf{p}$, giving a parametric representation of the line connecting the two points:

$$C_1(k) = \mathbf{p} + k\mathbf{v} = \mathbf{p} + k(\mathbf{q} - \mathbf{p}) = k\mathbf{q} + (1 - k)\mathbf{p}. \tag{2.1}$$

For any given scalar k, the point $C_1(k)$ is found by standing on the point $\mathbf{p}$ and walking $|k|$ times the distance to $\mathbf{q}$, either directly towards or directly away from $\mathbf{q}$ (see Figure 2.1). When $k = 0$ this point is just $\mathbf{p}$; when $k = 1$ it is $\mathbf{q}$.

The line $C_1(k)$ is formed by all affine combinations of $\mathbf{p}$ and $\mathbf{q}$, since the coefficients of the points in Equation (2.1) sum to 1. One says that $\mathbf{p}$ and $\mathbf{q}$ *generate* the line C_1. When $0 \leq k \leq 1$, both k and $1 - k$ are non-negative and the combinations are convex: the point $C_1(k)$ lies between $\mathbf{p}$ and $\mathbf{q}$, lending an intuitive justification to the earlier definition of convexity in Chapter 1. In what follows, all combinations are assumed to be convex unless otherwise specified. The operation performed in (2.1) is called *linear interpolation*. It is said that $C_1(k)$ *interpolates* $\mathbf{p}$ and $\mathbf{q}$.

The direction and length of the line segment are determined completely by the

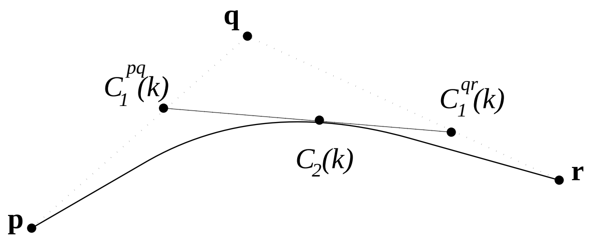

Figure 2.2: Repeated linear interpolation between three points

spatial location of its generating points. A designer can interactively control $C_1(k)$ by manipulating these. For this reason, $\mathbf{p}$ and $\mathbf{q}$ are called *control points* of the line segment. Furthermore, since the control points form the endpoints of the segment, they determine it uniquely. Hence we may speak of *the* control points of the segment.

Consider, now, the case of three points in space: $\mathbf{p}, \mathbf{q}$, and $\mathbf{r}$. As before, $\mathbf{p}$ and $\mathbf{q}$ determine a parameterized line segment C_1^{pq}. Another parameterized line segment, C_1^{qr}, is determined by $\mathbf{q}$ and $\mathbf{r}$. A given scalar k' fixes two points, $C_1^{pq}(k')$ and $C_1^{qr}(k')$, one on each segment (see Figure 2.2). These two points determine another line segment which may also be parameterized by k, producing the final point $C_2(k)$ given by

$$\begin{aligned} C_2(k) &= C_1^{pq}(k) + k(C_1^{qr}(k) - C_1^{pq}(k)) \\ &= kC_1^{qr}(k) + (1-k)C_1^{pq}(k) \\ &= k(k\mathbf{r} + (1-k)\mathbf{q}) + (1-k)(k\mathbf{q} + (1-k)\mathbf{p}) \\ &= k^2\mathbf{r} + 2k(1-k)\mathbf{q} + (1-k)^2\mathbf{p}. \end{aligned} \tag{2.2}$$

Notice that $C_2(k)$ is a degree-2 polynomial, a quadratic. It traces out a curve (a parabola) defined by convex combinations of three points, and is therefore planar.

Before extending these curves to a general, non-planar setting, their structure is examined more closely. It is readily seen that whereas the first and final control points are always interpolated as in the linear case, the middle one is not necessarily included. Next, by introducing a change of notation, one can write the control points

$\mathbf{p}, \mathbf{q}, \mathbf{r}$ as $\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}$ respectively, yielding the *monomial form* of (2.2)

$$C_2(k) = \sum_{i=0}^{2} \binom{2}{i} k^i (1-k)^{2-i} \mathbf{p}_i, \tag{2.3}$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

are the binomial coefficients. This equation shall be revisited later in Section 2.3.

This process of curve construction is easily generalized to an arbitrary number of control points. Every pair of successive control points produces a new, interpolated point for the given parameter value. Each of these is given as a linear function of k. When a new control point is added, the number of these interpolated points equals the number of control points that existed before the new one was added. Thus, if the previous curve involved a polynomial of degree n combining the control points, the addition of one point will produce a polynomial of degree n combining *linear polynomials*, a curve of degree $n+1$. Points constructed by interpolating two control points are called *level-1* points; those constructed from two level-1 points are called *level-2* points, and so on. Because two points yield a line, simple induction shows that $n+1$ control points produce a curve of degree n. Furthermore, when the control points are not co-planar, as is possible with four points or more, then neither is the curve.

In this fashion, curves of arbitrary degrees may be constructed. To produce a curve of degree n, all the designer must do is lay down $n+1$ control points. This curve is then completely determined by the number and location of these. By interactively moving them in space, the designer is afforded an intuitive and effective means of shape-design.

2.2 The de Casteljau Algorithm

The curve creation algorithm described above was developed by Paul de Casteljau in 1959 ([dC59] as referenced in [Far93]). Due to historical circumstances, however,

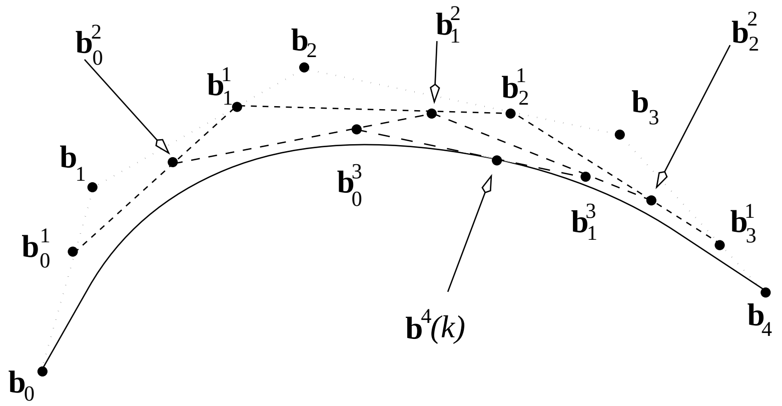

Figure 2.3: The de Casteljau algorithm on a quartic curve

curves created in this manner are called *Bézier curves*, named after P. Bézier whose work was published first. Given the intuition, the algorithm is developed in formal terms.

The de Casteljau algorithm for curves:

Let $\mathbf{b}_0, \ldots, \mathbf{b}_n \in \mathbf{R}^3$ *be a set of* $n+1$ *points, and let* $k \in \mathbf{R}$ *be a parameter. Put* $\mathbf{b}_i^0 = \mathbf{b}_i$*, and set*

$$\mathbf{b}_i^r(k) = k\mathbf{b}_{i+1}^{r-1}(k) + (1-k)\mathbf{b}_i^{r-1}(k), \tag{2.4}$$

where $r = 1, \ldots, n$*, and* $i = 0, \ldots, n-r$*. Then the Bézier curve of degree* n *is given by* $\mathbf{b}^n(k) = \mathbf{b}_0^n(k)$*.*

The control points $\mathbf{b}_i$ are also known as *Bézier points*. The polygon formed by the Bézier points is called the *Bézier polygon* or *control polygon* of the Bézier curve $\mathbf{b}^n$. Note that all intermediate points formed in the process, the ones with a non-zero superscript r, are strictly dependent on the parameter k. In particular, to determine the point with parameter value k on $\mathbf{b}^n$, all intermediate points must be explicitly evaluated for the specific k. Figure 2.3 depicts the de Casteljau algorithm generating a quartic Bézier curve.

The basic operation used by this algorithm is linear interpolation. An important property of linear interpolation is that it is invariant under affine maps. The invariance holds both for points *and* for the parameter. Since the de Casteljau algorithm consists of nothing more than repeated linear interpolation, it follows that it, too, is invariant under affine maps. Consequently, in order to move a Bézier curve from one place to another, it is sufficient to transform the control polygon and re-evaluate the curve at the new location; a process considerably cheaper than transforming the continuum of single points along the curve. Furthermore, the domain $[0,1]$ of the curve can be moved to any interval $[u,v]$ without affecting the final product, by using the affine map $k : t \mapsto (t-u)/(v-u)$ sending $t \in [u,v]$ to $k \in [0,1]$. Another consequence of the repeated linear interpolation is that $\mathbf{b}^n$ lies completely inside the convex hull of its control polygon. This follows from the fact that every intermediate point generated is a convex combination of two others which are in the convex hull of the polygon. At no step of the way are points created outside this hull.

A final property of the algorithm is that the first and final control points are interpolated into the curve. This is easily verified by inspecting (2.4): For $k = 0$, $\mathbf{b}^n(k) = \mathbf{b}_0$; and for $k = 1$, $\mathbf{b}^n(k) = \mathbf{b}_n$.

2.3 Bernstein Polynomials and the Bernstein Form

The de Casteljau algorithm is attractive because it provides an intuitive feeling for the Bézier curves. Its recursive nature, however, is not conducive to smooth theoretical development, and a more compact representation is desired.

Refer back to the quadratic curve of Equation (2.3): The coefficients of the points *look* very similar to the terms occurring in the binomial theorem. Indeed, they are exactly those occurring in the expansion of $[k+(1-k)]^2$. This leads to the consideration of the general integer exponent. Expanding the LHS of the equation $[k+(1-k)]^n = 1$ yields

$$[k + (1-k)]^n = \sum_{i=0}^{n} \binom{n}{i} k^i (1-k)^{n-i}. \tag{2.5}$$

The individual terms in the summation, when considered as functions of k, are known as the *Bernstein polynomials of degree* n. The i-th such polynomial is thus defined as

$$B_i^n(k) = \binom{n}{i} k^i (1-k)^{n-i}. \tag{2.6}$$

With $k \in [0,1]$ it is immediately apparent from Equation (2.5) that the set of n-th degree Bernstein polynomials forms a partition of unity.

One property of these polynomials is that each B_i^n is symmetric in k on the unit interval $[0,1]$: $B_i^n(k) = B_i^n(1-k)$. Of greater significance is that the polynomials satisfy the following recursion [Far93]:

$$B_i^n(k) = kB_{i-1}^{n-1}(k) + (1-k)B_i^{n-1}(k), \tag{2.7}$$

where $B_0^0(k) \equiv 1$, and $B_i^n(k) \equiv 0$ when $i \notin \{0, \ldots, n\}$. The proof of this last property is straightforward using the combinatorial identity $\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}$. To obtain the result, one merely writes

$$\begin{aligned} B_i^n(k) &= \binom{n}{i} k^i (1-k)^{n-i} \\ &= \binom{n-1}{i-1} k^i (1-k)^{n-i} + \binom{n-1}{i} k^i (1-k)^{n-i} \\ &= kB_{i-1}^{n-1}(k) + (1-k)B_i^{n-1}(k). \end{aligned}$$

Recursion (2.7) can be used to produce the monomial form of Equation (2.4). Toward this end, a stronger claim is proven, namely that

$$\mathbf{b}_i^r(k) = \sum_{j=0}^{r} B_j^r(k)\mathbf{b}_{i+j}, \qquad \begin{array}{l} r = 0, \ldots, n \\ i = 0, \ldots, n-r \end{array}. \tag{2.8}$$

The proof consists of noting first that the result is trivial when $r = 0$. Proceeding by induction on r, it is assumed that the result holds true for $r-1$. Thus

$$\mathbf{b}_i^r(k) \quad = \quad k\mathbf{b}_{i+1}^{r-1}(k) + (1-k)\mathbf{b}_i^{r-1}(k)$$

$$
\begin{aligned}
&= k\sum_{j=0}^{r-1} B_j^{r-1}(k)\mathbf{b}_{i+j+1} + (1-k)\sum_{j=0}^{r-1} B_j^{r-1}(k)\mathbf{b}_{i+j} \\
&= k\sum_{j=-1}^{r-1} B_j^{r-1}(k)\mathbf{b}_{i+j+1} + (1-k)\sum_{j=0}^{r} B_j^{r-1}(k)\mathbf{b}_{i+j} \qquad (\star) \\
&= \sum_{j=-1}^{r-1} \binom{r-1}{j} k^{j+1}(1-k)^{r-(j+1)}\mathbf{b}_{i+j+1} + \sum_{j=0}^{r} \binom{r-1}{j} k^{j}(1-k)^{r-j}\mathbf{b}_{i+j} \\
&= \sum_{j=0}^{r} \binom{r-1}{j-1} k^{j}(1-k)^{r-j}\mathbf{b}_{i+j} + \sum_{j=0}^{r} \binom{r-1}{j} k^{j}(1-k)^{r-j}\mathbf{b}_{i+j} \\
&= \sum_{j=0}^{r} \binom{r}{j} k^{j}(1-k)^{r-j}\mathbf{b}_{i+j} \\
&= \sum_{j=0}^{r} B_j^{r}(k)\mathbf{b}_{i+j},
\end{aligned}
$$

where the line $(\star)$ is justified since $B_{-1}^{r-1}(k), B_r^{r-1}(k) \equiv 0$ by definition. By setting $r = n$ in Equation (2.8) the following form is obtained:

The Bernstein form of the Bézier curve:

$$\mathbf{b}^n(k) = \sum_{i=0}^{n} B_i^n(k)\mathbf{b}_i. \tag{2.9}$$

Because the only terms involving the parameter k in (2.9) are the Bernstein polynomials, themselves symmetric in k, it follows that the entire curve is symmetric in k on the interval $[0,1]$: $\mathbf{b}^n(k) = \mathbf{b}^n(1-k)$. This means that the order in which the curve is traversed is immaterial to its shape, and hence the ordering of the Bézier points may be reversed at will.

2.4 Degree Elevation

The process of curve design generally starts with the designer setting down an initial curve of a certain degree, and then interactively reshaping it by manipulating the control points. The freedom of shape is, of course, restricted by the degree of the curve and thus by the number of control points. The greater the degree of the curve, the more freedom the designer has in defining its shape. The freedom, however, bears with it an increase in complexity, as maneuvering the curve becomes more difficult with the additional control points. Typically, the designer will start with a low degree curve (usually a cubic), and will add control points by raising the degree only when greater control is necessary.

When raising the degree, the ideal situation would be to obtain an identical curve. In other words, if $\{\mathbf{b}_i\}_0^n$ is the initial set of Bézier points and $\{\hat{\mathbf{b}}_i\}_0^{n+1}$ is the new set, we would like to have, initially, $\hat{\mathbf{b}}^{n+1}(k) = \mathbf{b}^n(k)$. Using equation (2.9) and multiplying the RHS by $[k + (1-k)] = 1$, we obtain

$$\begin{aligned}\sum_{i=0}^{n+1} B_i^{n+1}(k)\hat{\mathbf{b}}_i &= \sum_{i=0}^{n} B_i^n(k)\mathbf{b}_i \\ \sum_{i=0}^{n+1}\binom{n+1}{i}k^i(1-k)^{n+1-i}\hat{\mathbf{b}}_i &= [k+(1-k)]\sum_{i=0}^{n}\binom{n}{i}k^i(1-k)^{n-i} \\ &= \sum_{i=0}^{n}\binom{n}{i}[k^{i+1}(1-k)^{n+1-(i+1)} + k^i(1-k)^{n+1-i}]\mathbf{b}_i.\end{aligned}$$

Two polynomials are equivalent if and only if all respective coefficients agree. Considering the last equation as a polynomial[1] in $k(k+1)$, the coefficients of $k^i(1-k)^{n+1-i}$ on both sides are equated. The following relation between the Bézier points is obtained:

$$\binom{n+1}{i}\hat{\mathbf{b}}_i = \binom{n}{i}\mathbf{b}_i + \binom{n}{i-1}\mathbf{b}_{i-1}. \tag{2.10}$$

Upon rearrangement, this yields the condition

$$\hat{\mathbf{b}}_i = (1 - \frac{i}{n+1})\mathbf{b}_i + \frac{i}{n+1}\mathbf{b}_{i-1}. \tag{2.11}$$

Thus the vertices of the new control polygon are obtained by linearly interpolating, or *resampling*, those of the old polygon over a uniform knot spacing of $i/(n+1)$. Accordingly, the new polygon lies in the convex hull of the old one.

Using operator notation, one can denote the process of degree elevation by an operator $\mathcal{E}$ acting on the control polygon $\mathbf{P}$, which sends the degree n control polygon

[1] See Section 2.6

$\mathbf{P}$ to the degree $n+1$ polygon $\mathcal{E}\mathbf{P}$. Along a similar scheme, one may use the operator $\mathcal{B}$ to associate a control polygon with its Bézier curve: $\mathbf{b}^n = \mathcal{B}\mathbf{P}$. Following the approach of J. Zhou [Zho85], Farin [Far93] shows that through the process of repeated degree elevation, the sequence of control polygons corresponding to $\mathbf{b}^n$ converges to their Bézier curve:

$$\lim_{r\to\infty} \mathcal{E}^r\mathbf{P} = \mathcal{B}\mathbf{P}.$$

He notes, however, that this convergence is too slow to be of any practical value.

A final thought on degree elevation is that raising the degree of a curve introduces a certain amount of redundancy, as the raised curve is initially defined by more information than is required. The inverse of this process, degree *reduction*, is not generally possible. For example, a quadratic curve with an extremal point cannot possibly be rewritten linearly. Approximation of the curve, however, is possible although the results are generally unsatisfactory. A detailed account of degree reduction can be found in [WW88].

2.5 Derivatives

Recall that the i-th Bernstein polynomial of degree n is defined in equation (2.6) as $B_i^n(k) = \binom{n}{i} k^i(1-k)^{n-i}$. Its derivative is readily computed:

$$\begin{aligned}
\frac{d}{dk}B_i^n(k) &= \frac{d}{dk}\left[\binom{n}{i}k^i(1-k)^{n-i}\right] \\
&= \frac{n!}{i!(n-i)!}[ik^{i-1}(1-k)^{n-i} - (n-i)k^i(1-k)^{n-i-1}] \\
&= \frac{n(n-1)!}{(i-1)!(n-i)!}k^{i-1}(1-k)^{n-i} - \frac{n(n-1)!}{i!(n-i-1)!}k^i(1-k)^{n-i-1} \\
&= n[B_{i-1}^{n-1}(k) - B_i^{n-1}(k)].
\end{aligned}$$

The derivative of the Bézier curve $\mathbf{b}^n$ can now be expressed as

$$\frac{d}{dk}\mathbf{b}^n(k) = \sum_{i=0}^{n} \frac{d}{dk}B_i^n(k)\mathbf{b}_i$$

$$
\begin{aligned}
&= n\sum_{i=0}^{n}[B_{i-1}^{n-1}(k) - B_i^{n-1}(k)]\mathbf{b}_i \\
&= n[\sum_{i=1}^{n} B_{i-1}^{n-1}\mathbf{b}_i - \sum_{i=0}^{n-1} B_i^{n-1}(k)\mathbf{b}_i] \qquad (\star) \\
&= n[\sum_{i=0}^{n-1} B_i^{n-1}(k)\mathbf{b}_{i+1} - \sum_{i=0}^{n-1} B_i^{n-1}(k)\mathbf{b}_i] \\
&= n\sum_{i=0}^{n-1}[\mathbf{b}_{i+1} - \mathbf{b}_i]B_i^{n-1}(k),
\end{aligned}
$$

where the line $(\star)$ is justified by the definition of B_i^n for $i \notin \{0, \ldots, n\}$.

The derivative is thus given by the expression

$$\frac{d}{dk}\mathbf{b}^n(k) = n\sum_{i=0}^{n-1}[\mathbf{b}_{i+1} - \mathbf{b}_i]B_i^{n-1}(k). \tag{2.12}$$

That is, the derivative of a Bézier curve is itself a Bézier curve whose control points are given by forward differencing those of the original curve. This derivative curve is sometime called a *hodograph* ([SWS95], [BFK84]).

2.6 The Barycentric Form of the Bézier Curve

The affine combination of two points $\mathbf{p}$ and $\mathbf{q}$ was earlier defined as $\mathbf{c}(k) = k\mathbf{p}+(1-k)\mathbf{q}$ for any $k \in \mathbf{R}$. The same point may be equivalently expressed as a *barycentric combination* $\mathbf{c}(u, v) = u\mathbf{p} + v\mathbf{q}$ where $u + v$=1. Typically, the parameter is written as a tuple $\mathbf{u} = (u, v)$, giving the point $\mathbf{c}(\mathbf{u})$. The parameters u and v are called the *barycentric coordinates* of the point $\mathbf{c}$. Note that although the latter form looks bivariate, it is in fact only *uni*variate because of the condition that the parameters sum to unity.

The difference between the two forms is purely notational. The significance of the barycentric form will become clear in the next chapter when the Bézier curves are extended to surfaces. For comparative reasons, the results of this chapter are now presented in barycentric form.

By setting $u = 1 - k$, $v = k$, and $\mathbf{b_i} = \mathbf{b}_{ij} = \mathbf{b}_i$ in equation (2.9) the barycentric form of the curve is obtained:

$$\mathbf{b}^n(\mathbf{u}) = \sum_{|\mathbf{i}|=n} B^n_{\mathbf{i}}(\mathbf{u})\mathbf{b}_{\mathbf{i}}, \tag{2.13}$$

where $\mathbf{i} = (i, j)$, $|\mathbf{i}| = i + j$,

$$B^n_{\mathbf{i}}(\mathbf{u}) = B^n_{ij}(u, v) = \binom{n}{i, j} u^i v^j$$

and

$$\binom{n}{i, j} = \binom{n}{\mathbf{i}} = \frac{n!}{i!j!} = \binom{n}{i}.$$

Equation (2.4) may also be rewritten producing an alternate form of the de Casteljau algorithm

$$\mathbf{b}^r_{\mathbf{i}}(\mathbf{u}) = \mathbf{b}^r_{i,j}(\mathbf{u}) = u\mathbf{b}^{r-1}_{i+1,j}(\mathbf{u}) + v\mathbf{b}^{r-1}_{i,j+1}(\mathbf{u}), \tag{2.14}$$

with $r = \{0, \ldots, n\}$, and $|\mathbf{i}| = i + j = n - r$.

Having written $\mathbf{u} = \mathbf{u}(k) = (1-k, k)$, the derivative of the curve is easily computed using the chain rule:

$$\begin{aligned} \frac{d}{dk}\mathbf{b}(\mathbf{u}(k)) &= \frac{\partial \mathbf{b}}{\partial u}\frac{du}{dk} + \frac{\partial \mathbf{b}}{\partial v}\frac{dv}{dk} \\ &= \frac{\partial \mathbf{b}}{\partial v} - \frac{\partial \mathbf{b}}{\partial u}. \end{aligned} \tag{2.15}$$

The *directional derivative* of $\mathbf{x} = (x_1, x_2)$ in direction (α, β) is defined as $D_{\alpha,\beta}(\mathbf{x}) = \alpha\partial\mathbf{x}/\partial x_1 + \beta\partial\mathbf{x}/\partial x_2$. Thus by putting $\mathbf{d} = (-1, 1)$, Equation (2.15) can be expressed as a directional derivative:

$$\frac{d}{dk}\mathbf{b}(\mathbf{u}(k)) = D_{\mathbf{d}}\mathbf{b}(\mathbf{u}). \tag{2.16}$$

Finally, invoking the de Casteljau algorithm (2.14) with parameter value $\mathbf{d}$ produces

$$\mathbf{b}^1_{\mathbf{i}}(\mathbf{d}) = -\mathbf{b}_{i,j+1}(\mathbf{d}) + \mathbf{b}_{i+1,j}(\mathbf{d}),$$

which, together with (2.16) and (2.12), yields the barycentric form of the derivative:

$$D_{\mathbf{d}}\mathbf{b}(\mathbf{u}) = n \sum_{|\mathbf{i}|=n-1} \mathbf{b}^1_{\mathbf{i}}(\mathbf{d}) B^{n-1}_{\mathbf{i}}(\mathbf{u}).$$

Chapter 3

Triangular Bézier Patches

Numerous methods have been formulated for the sake of extending the notion of parametric curves to surfaces. Just as in the case with curves, large surfaces are most easily constructed piecemeal from a collection of individual surface elements, colloquially known as *patches*. Each patch is individually specified in an attempt to reduce the overall complexity of the model. The seaming of adjacent patches is kept tight and non-overlapping in order to produce a smooth and coherent surface appearance. Jagged edges are avoided in favor of smooth curves along the patch boundaries, further reducing the occurrence of cracks.

The most common forms of surface patches are bounded by four space curves, and called *rectangular surface patches*[1]. Basically, such a patch may be visualized as a swath created by tracing the movement of a curve in space whose shape is allowed to deform along the way. The initial and final curves form two opposite "edges" of the patch; the remaining two edges are the loci of the curve endpoints (see Figure 3.1). Surface patches are therefore generally smooth, or *fair*, and tend not to self-intersect. They are thus two-dimensional manifolds. One common method of formulating rectangular patches is using Bézier curves. Because of the affine invariance of these curves, the sweeping motion of the curve may be achieved by moving its control polygon. But because the sweeping curve is permitted to undergo free-form

[1]This is an unfortunate misnomer, as the word *rectangular* refers not to the shape of the patch but to that of its parameter domain.

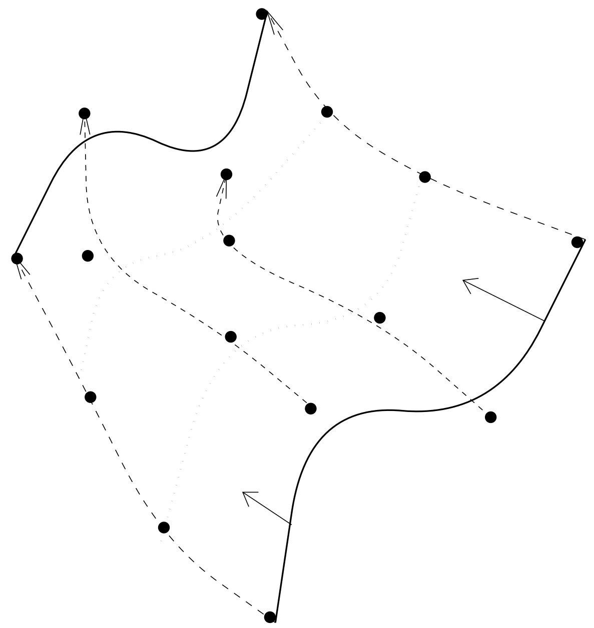

Figure 3.1: A bicubic Bézier patch

deformations, each of its control points may be assigned an individual trajectory. These trajectories may themselves be specified by Bézier curves. Since the surface is parameterized by two variables, one can be used to parameterize the sweeping curve, and the other to parameterize the trajectories. Setting one parameter fixes a point on each trajectory, thus producing the control points for the curve; the second parameter then fixes a point on the curve and hence in the interior of the patch. This method which bears his name was first developed by Bézier at Renault. Because of the characteristic separability of the two parameters, patches generated in this method are also called *tensor-product Bézier patches.* Analogous extension of the Hermite and B-spline curves lead to the Hermite (Coons) and B-spline patches respectively. In practice, cubic curves are most often used because no lower degree curve permits the prescription of both the positions and the tangent vectors at the endpoints. Such patches are also called *bicubic patches*, accounts of which may be found in [FvDFH92], [WW92], and [HL93].

Although common, rectangular patches suffer from inherent difficulties arising from their topology, namely the inability to precisely tessellate an arbitrary object with non-degenerate four-sided surface elements.[2] A more natural mathematical construct for this is a triangle. Most of the difficulties can be avoided by using surfaces constructed over non-degenerate triangular domains, and indeed triangles were the first surface types considered by de Casteljau in his work [dC63]. Interestingly, although both de Casteljau and Bézier separately came up with the same formulation for curves, their respective extensions to surfaces are quite different.

This chapter will discuss the extension of the methods used for curves to triangular surface patches. To begin, the notion of *barycentric coordinates* is generalized, and the intuition behind the surface construction is provided. Next, the *de Casteljau algorithm for surfaces* is defined, and the previous notion of *Bernstein polynomials* is extended to produce the *Bernstein form* of Bézier triangles. A mathematical tool known as *blossoming* is then introduced, simplifying the subsequent development of additional modeling issues. Finally, a comparison of sampling techniques is provided.

3.1 Barycentric Coordinates

Any three non-collinear points in space define a plane. Given the points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ in $\mathbf{R}^3$, the plane $\mathbf{P}$ that they span may be described as the set of points whose distance from the plane is nil. In set theoretic notation $\mathbf{P}$ is defined as:

$$\mathbf{P} := \{\mathbf{d} \in \mathbf{R^3} \quad | \quad \mathbf{d} \cdot [(\mathbf{a} - \mathbf{b}) \times (\mathbf{c} - \mathbf{b})] = 0\}.$$

This definition of the plane is certainly not unique, as $\mathbf{P}$ is spanned by any three non-collinear points on it. However given a specific triangle, every point $\mathbf{p} \in \mathbf{P}$ can be expressed as a weighted sum of the vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

$$\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}, \tag{3.1}$$

[2]The insistence on non-degeneracy is essential to ensure algorithm robustness.

with the additional constraint that

$$u + v + w = 1. \tag{3.2}$$

The coefficients $\mathbf{u} = (u, v, w)$ are called the *barycentric coordinates* of $\mathbf{p}$ with respect to the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. The term *barycentric* arises from a physical context and means *center of mass* (*barys*, in Greek, means heavy), as the center of mass (or gravity) of three objects with masses u, v, w lying, respectively, at the points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ is at $\mathbf{p}$. To generalize slightly, the requirement that all components of $\mathbf{u}$ be non-negative is eliminated, although this is usually the case. Points inside the triangle are inside the convex hull of its vertices, and indeed have non-negative coordinates. The remaining points on $\mathbf{P}$ have at least one negative coordinate. Generally the distinction between the point $\mathbf{p}$ and its corresponding barycentric coordinates $\mathbf{u}$ is dropped, and the two are used synonymously.

Given three non-collinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$, the barycentric coordinates of a fourth co-planar point $\mathbf{p}$ with respect to the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ can always be determined ([Far93]). Since all four points lie on a plane isomorphic to $\mathbf{R}^2$, equation (3.1) provides two constraints, while Equation (3.2) provides a third. The three linear equations are sufficient to uniquely determine the three unknowns u, v, and w. These coordinates can also be determined geometrically by:

$$u = \frac{\mathcal{A}(\mathbf{p}, \mathbf{b}, \mathbf{c})}{\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{c})} \qquad v = \frac{\mathcal{A}(\mathbf{a}, \mathbf{p}, \mathbf{c})}{\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{c})} \qquad w = \frac{\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{p})}{\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{c})} \tag{3.3}$$

where $\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is the signed area of the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$, and is given analytically ([Far93], [GS93], [HL93]) by

$$\mathcal{A}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} \begin{vmatrix} 1 & 1 & 1 \\ \mathbf{a}_x & \mathbf{b}_x & \mathbf{c}_x \\ \mathbf{a}_y & \mathbf{b}_y & \mathbf{c}_y \end{vmatrix}.$$

In Section 2.1, linear interpolation of two points was defined and utilized to pa-

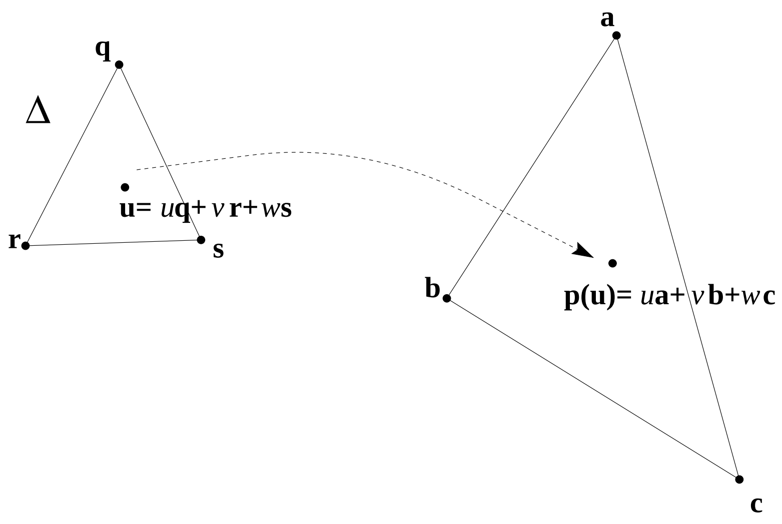

Figure 3.2: Spatial linear interpolation

rameterize the line between the two points. A similar process will now be used to define the *spatial linear interpolation* of three points, thus pararameterizing the triangle – and hence the plane – spanned by the three points. Contrary to the former case where the parameter k varied between 0 and 1, barycentric coordinates are used here. By varying the tuple $\mathbf{u} = (u, v, w)$ over any triangle $(\mathbf{q}, \mathbf{r}, \mathbf{s})$, a parameterization of a different triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is obtained (Figure 3.2). The triangle over which the parameter is varied is called the *domain-triangle* or *base-triangle* or, more formally, the *domain simplex* ([dB87, LD89]) and is denoted by Δ. Note that the actual location and shape of Δ is completely irrelevant to the linear interpolation, as the concern is solely with the barycentric parameterization. This fact will be used extensively below.

3.2 Intuition

To begin, consider three points in space: $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{R}^3$. Fixing a barycentric tuple $\mathbf{u}_0 = (u_0, v_0, w_0) \in \Delta$ with non-negative u_0, v_0, w_0 will determine a point $\mathbf{p}$ in the interior of the triangle:

$$\mathbf{p}(\mathbf{u_0}) = u_0\mathbf{a} + v_0\mathbf{b} + w_0\mathbf{c}.$$

Varying the parameter $\mathbf{u}$ over the entire domain will trace out the complete area inside the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$.

The spatial location of the triangle is, as before, completely determined by the location of its corner points $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and is not affected by the domain simplex. For this reason these points are called *control points* of the triangle. Notice that when one of the parameters in the barycentric tuple is zero, the corresponding control point does not affect the location of the parameterized point. When two parameters are zero, the parameterized point coincides with the third control point. In this way all three control points are interpolated into the triangle, forming its corners. Thus they uniquely determine the triangle, and hence may be called *the* control points. By moving them, the area, shape, aspect ratio, and location of the triangle may be set.

Now consider the case where the control points of the triangle are themselves parameterized. Each such point must lie in a separate triangle. This requires three triangles which are formed by six points in space: $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f} \in \mathbf{R^3}$. The set of points $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ forms one triangle; the set $\{\mathbf{b}, \mathbf{d}, \mathbf{e}\}$ forms another; and the set $\{\mathbf{c}, \mathbf{e}, \mathbf{f}\}$ yet another (see Figure 3.3). By linearly interpolating the points of each set with a fixed parameter $\mathbf{u} = \mathbf{u_0}$, three distinct points $\mathbf{p}^1, \mathbf{p}^2, \mathbf{p}^3$ are produced, one from each set:

$$
\begin{aligned}
\mathbf{p}^1(\mathbf{u}_0) &= u_0\mathbf{a} + v_0\mathbf{b} + w_0\mathbf{c}, \\
\mathbf{p}^2(\mathbf{u}_0) &= u_0\mathbf{b} + v_0\mathbf{d} + w_0\mathbf{e}, \qquad \text{and} \\
\mathbf{p}^3(\mathbf{u}_0) &= u_0\mathbf{c} + v_0\mathbf{e} + w_0\mathbf{f}.
\end{aligned}
$$

These three points are now also interpolated using the same $\mathbf{u_0}$ to produce the final point $\mathbf{p}$:

$$
\begin{aligned}
\mathbf{p}(\mathbf{u}_0) &= u_0\mathbf{p}^1(\mathbf{u}_0) + v_0\mathbf{p}^2(\mathbf{u}_0) + w_0\mathbf{p}^3(\mathbf{u}_0) \\
&= u_0(u_0\mathbf{a} + v_0\mathbf{b} + w_0\mathbf{c}) + v_0(u_0\mathbf{b} + v_0\mathbf{d} + w_0\mathbf{e}) + \\
&\quad\; w_0(u_0\mathbf{c} + v_0\mathbf{e} + w_0\mathbf{f}) \\
&= u_0^2\mathbf{a} + v_0^2\mathbf{d} + w_0^2\mathbf{f} + 2u_0v_0\mathbf{b} + 2u_0w_0\mathbf{c} + 2v_0w_0\mathbf{e}. \qquad (3.4)
\end{aligned}
$$

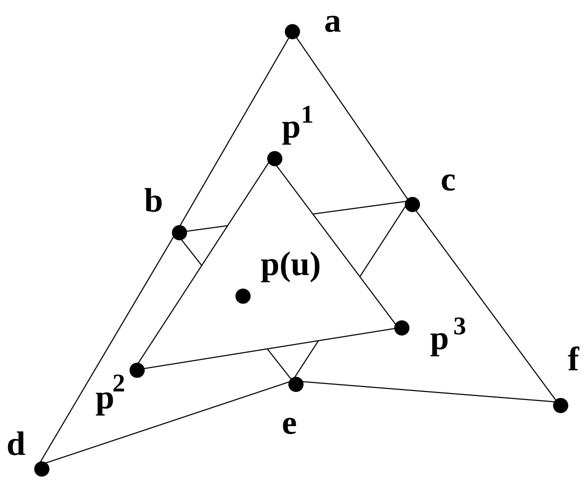

Figure 3.3: Quadratic triangular Bézier patch

The term *degree of a patch* is used to refer to the degree of the polynomial which defines the patch. Thus said, the patch defined by equation (3.4) is quadratic.

This process of surface construction can be generalized by parameterizing all the control points that produce a given patch. If the initial patch was of degree n, the resultant patch will be generated by a degree n polynomial that blends linear combinations of control points, thus producing a patch of degree $n + 1$.

A point produced by spatial linear interpolation must lie in the same plane as the points defining it. But when more than three control points are involved, the planes determined by the triangles do not necessarily coincide, so there is no guarantee on the planarity of the surface obtained. The surface is nevertheless bounded by three curves, and is hence called a *Triangular Bézier patch* or just a *Bézier triangle*.

A number of things must be noted at this point. First, when one of the parameters of a barycentric tuple is 0, *none* of the control points corresponding to that parameter contribute to the outcome of the interpolation, and what is produced from the others is a Bézier *curve*. Thus the boundary curves of a Bézier triangle are Bézier curves. Next, because each of the intermediary points lies in the convex hull of the three forming it, the patch lies completely inside the convex hull of the control points.

Patches generated in this fashion are completely determined by their set of control points. Moving the points in space affects the spatial attributes of the patch as well as its location. Patches of arbitrary degree may be created by simply setting down an appropriate number of control points. As is the case with curves, however, although a high degree patch offers greater shape control than one of a lesser degree, the proliferation of control points increases the complexity involved in controlling the shape. Typically a designer will begin with a cubic patch and will add control points only when necessary. Just as with curves, by interactively manipulating the control points, the designer is provided an intuitive means of sculpting and moulding the surface.

3.3 The de Casteljau Algorithm

Before formalizing the algorithm just described, some additional notation is introduced. A *multi-index* $\mathbf{i}$ is a tuple $(i_0, \ldots, i_m)$ of $m+1$ non-negative integers. Following [LD89, Far86], the norm of a multi-index $\mathbf{i}$ is defined to be the sum of its components, thus $|\mathbf{i}| = \sum_{j=0}^{m} i_j$. In what follows we restrict ourselves to the case where $m = 2$, and where the norm of the index equals some fixed integral value q. To simplify the notation, the components of the multi-index $\mathbf{i}$ will often be denoted by distinct letters (i, j, k). The number of multi-indices which obey these restrictions is equal to the number of ways to partition $[q]$ – the set of integers occurring in the interval $[0, q]$ – into three possibly empty parts, which is $\binom{q+2}{2} = (q+1)(q+2)/2$. The *addition* of multi-indices is performed componentwise. Three directional incrementor indices $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ will be used to simplify the notation. They are defined as $\mathbf{e}_1 = (1, 0, 0)$, $\mathbf{e}_2 = (0, 1, 0)$, and $\mathbf{e}_3 = (0, 0, 1)$.

For reasons that will become apparent shortly, multi-indices are used to label the control points of the Bézier triangles. The individual control points are also known as *Bézier points*; the collection of Bézier points associated with a Bézier patch is called its *Bézier mesh* or *Bézier net*. The mesh is written in a triangular structure. For a multi-index norm of n this structure is written as:

$$\begin{array}{ccccccc} & & & \mathbf{b}_{0n0} & & & \\ & & \mathbf{b}_{0,n-1,1} & & \mathbf{b}_{1,n-1,0} & & \\ & \mathbf{b}_{0,n-2,2} & & \mathbf{b}_{1,n-2,1} & & \mathbf{b}_{2,n-2,0} & \\ & \cdots & & \cdots & & \cdots & \\ \mathbf{b}_{00n} & & \cdots & & \cdots & & \mathbf{b}_{n00}. \end{array}$$

The points are labeled in such a way that the index i corresponds in some sense (that will soon become clear) to the closeness of the point to the rightmost vertex of the structure. The index j corresponds to the closeness of the topmost vertex, and the index k corresponds to the closeness of leftmost one. Thus, the points along the boundaries of the structure all have a 0's in the corresponding index position. For example, in the cubic case the Bézier net is:

$$\begin{array}{ccccccc} & & & \mathbf{b}_{030} & & & \\ & & \mathbf{b}_{021} & & \mathbf{b}_{120} & & \\ & \mathbf{b}_{012} & & \mathbf{b}_{111} & & \mathbf{b}_{210} & \\ \mathbf{b}_{003} & & \mathbf{b}_{102} & & \mathbf{b}_{201} & & \mathbf{b}_{300}. \end{array}$$

The de Casteljau algorithm for surfaces:

Let $\{\mathbf{b_i} \in \mathbf{R}^3\}_{|\mathbf{i}|=n}$ *be a set of* $\binom{n+2}{2}$ *control points, and let* $\mathbf{u} \in \Delta$ *be a parameter. Put* $\mathbf{b}_\mathbf{i}^0 = \mathbf{b_i}$, *and set*

$$\mathbf{b}_\mathbf{i}^r(\mathbf{u}) = u\mathbf{b}_{\mathbf{i}+\mathbf{e}_1}^{r-1}(\mathbf{u}) + v\mathbf{b}_{\mathbf{i}+\mathbf{e}_2}^{r-1}(\mathbf{u}) + w\mathbf{b}_{\mathbf{i}+\mathbf{e}_3}^{r-1}(\mathbf{u}), \tag{3.5}$$

where $r = 1, \ldots, n$, *and* $|\mathbf{i}| = n - r$. *Then the Bézier triangle of degree* n *is given by* $\mathbf{b}^n(\mathbf{u}) = \mathbf{b}_\mathbf{0}^n(\mathbf{u}) = \mathbf{b}_{000}^n(\mathbf{u})$.

The triangular mesh generated for level r is of one degree less than the generating

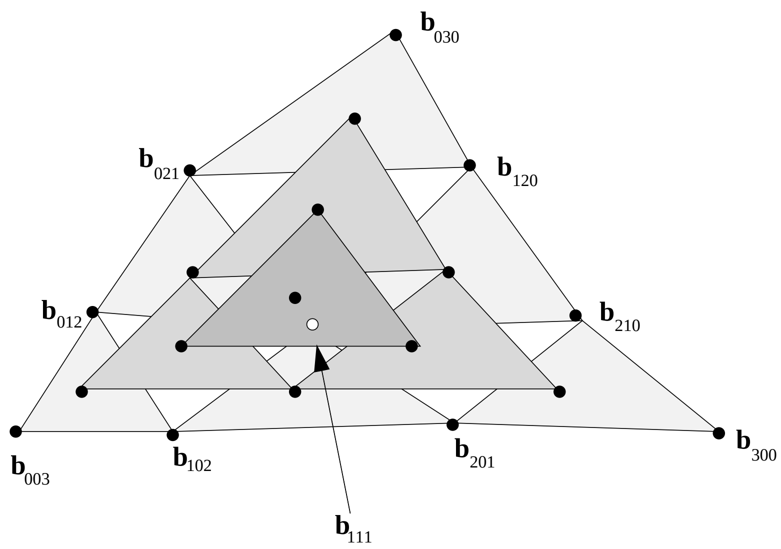

Figure 3.4: The de Casteljau algorithm

mesh of level $r-1$. Thus at level $r = n$ only one point $\mathbf{b}_\mathbf{0}^n(\mathbf{u})$ remains. By applying (3.5) to every point $\mathbf{u} \in \Delta$, the complete Bézier surface is obtained (figure 3.4).

From the *de Casteljau* algorithm, a number of properties of the Bézier triangles become evident. First, because the process of linear interpolation is affinely invariant as are the barycentric coordinates of points within the domain triangle, the patch itself is invariant under affine transformations of both the mesh and the base triangle. This means that in order to reposition (reparameterize) a given patch it is sufficient to reposition (reparameterize) its Bézier points (domain simplex). A second useful fact is the convex hull property: because every $\mathbf{b}_\mathbf{i}^r$ is a convex combination of the previous $\mathbf{b}_\mathbf{i}^{r-1}$'s, it lies within their convex hull. Therefore every surface point $\mathbf{b}^n$, and hence the convex hull of the entire Bézier triangle, lies with the convex hull of its control net. Finally, all points along a boundary curve of a patch have 0 as one of their parameters u, v, or w. Thus *patch*–boundary curves are determined completely by the control points along the *net*–boundary, and hence are regular Bézier curves.

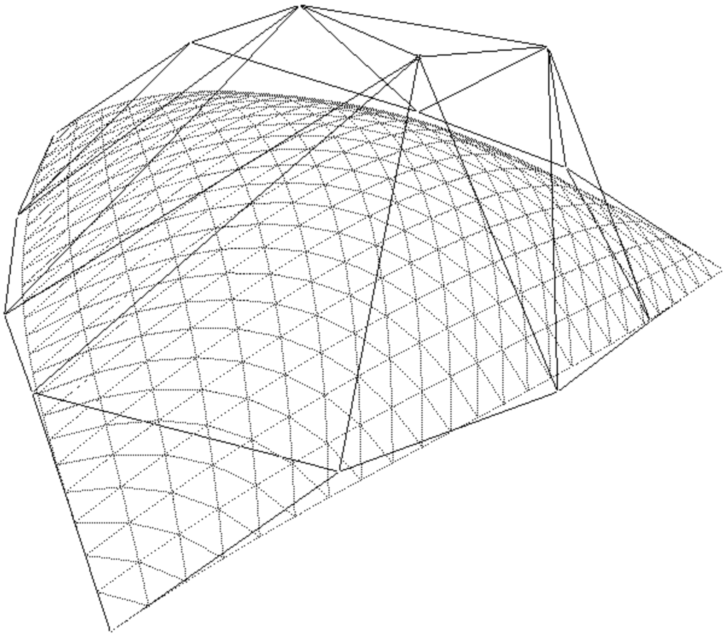

Figure 3.5: A quartic Bézier patch with its control net

3.4 Generalized Bernstein Polynomials and the Bernstein Form

Bernstein polynomials were introduced in Section 2.3 as being the summands occurring in the binomial expansion of $[k+(1-k)]^n$. The definition is now extended by considering the expansion of $|\mathbf{u}|^n$ for a barycentric triple $\mathbf{u}$. On the one hand, $|\mathbf{u}|^n = 1^n = 1$. On the other,

$$\begin{aligned}
|\mathbf{u}|^n &= (u+v+w)^n \\
&= \sum_{i=0}^{n} \binom{n}{i} u^i (v+w)^{n-i} \\
&= \sum_{i=0}^{n} \sum_{j=0}^{n-i} \binom{n}{i} \binom{n-i}{j} u^i v^j w^{n-i-j} && (3.6) \\
&= \sum_{|(i,j,k)|=n} \frac{n!}{i!j!k!} u^i v^j w^k \\
&= \sum_{|\mathbf{i}|=n} \binom{n}{\mathbf{i}} u^i v^j w^k. && (3.7)
\end{aligned}$$

The individual terms in the sum are called the *generalized Bernstein polynomials of degree n*. Each such polynomial is subscripted by its appropriate multi-index $\mathbf{i}$:

$$B_{\mathbf{i}}^n(\mathbf{u}) = \binom{n}{\mathbf{i}} u^i v^j w^k. \tag{3.8}$$

The polynomial $B_{\mathbf{i}}^n$ is defined to be identically 0 whenever one of the elements occurring in the multi-index is negative. Note that although the $B_{\mathbf{i}}^n$'s *look* trivariate, they are in fact only *bi*variate since $w = 1 - u - v$.

As before, it is immediately apparent that the set of generalized Bernstein polynomials of degree n form a partition of unity. The cardinality of each set is $\binom{n+2}{2}$, which equals the number of control points called for by the de Casteljau algorithm to generate a Bézier triangle of degree n. The collection of polynomials forming each such set is clearly linearly independent, and thus spans an $\binom{n+2}{2}$-dimensional linear subspace of $\mathcal{P}(n,n)$, the space of bivariate polynomials of degree n. Each element $\mathbf{x}$ in this linear subspace can be expressed uniquely as a linear combination of these basis functions:

$$\mathbf{x}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u}).$$

This fact will be used to construct a compact representation for the Bézier triangles.

Using the combinatorial identity $\binom{n}{\mathbf{i}} = \binom{n}{i}\binom{n-i}{j}$, which follows from lines (3.6) and (3.7) above, in conjunction with the identity $\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$, the identity

$$\binom{n}{\mathbf{i}} = \binom{n-1}{i,j,k-1} + \binom{n-1}{i,j-1,k} + \binom{n-1}{i-1,j,k},$$

is derived (details in Appendix A.1). From this, the recursion:

$$\begin{aligned} B_{\mathbf{i}}^n(\mathbf{u}) &= \binom{n}{\mathbf{i}} u^i v^j w^k \\ &= uB_{\mathbf{i}-\mathbf{e}_1}^{n-1}(\mathbf{u}) + vB_{\mathbf{i}-\mathbf{e}_2}^{n-1}(\mathbf{u}) + wB_{\mathbf{i}-\mathbf{e}_3}^{n-1}(\mathbf{u}), \end{aligned} \tag{3.9}$$

is obtained.

In the vein of (2.9), the similarity between recursions (3.9) and (3.5) suggests that

Equation (3.5) can be written in terms of the Bernstein polynomials. This can be done using a stronger claim, namely [Far93]:

$$\mathbf{b}_{\mathbf{i}}^{r}(\mathbf{u}) = \sum_{|\mathbf{j}|=r} \mathbf{b}_{\mathbf{i}+\mathbf{j}} B_{\mathbf{j}}^{r}(\mathbf{u}), \qquad \begin{array}{l} r = 0, \dots n \\ |\mathbf{i}| = n - r. \end{array} \tag{3.10}$$

The proof consists of using the definition of $\mathbf{b}_{\mathbf{i}}^{r}$ in (3.5) and inducting on r. This gives:

$$\begin{aligned} \mathbf{b}_{\mathbf{i}}^{r+1} &= u\mathbf{b}_{\mathbf{i}+\mathbf{e}_1}^{r} + v\mathbf{b}_{\mathbf{i}+\mathbf{e}_2}^{r} + w\mathbf{b}_{\mathbf{i}+\mathbf{e}_3}^{r} \\ &= \sum_{|\mathbf{j}|=r} \left[u\mathbf{b}_{\mathbf{i}+\mathbf{j}+\mathbf{e}_1} B_{\mathbf{j}}^{r} + v\mathbf{b}_{\mathbf{i}+\mathbf{j}+\mathbf{e}_2} B_{\mathbf{j}}^{r} + w\mathbf{b}_{\mathbf{i}+\mathbf{j}+\mathbf{e}_3} B_{\mathbf{j}}^{r}\right] \\ &= \sum_{|\mathbf{j}|=r+1} \left[u\mathbf{b}_{\mathbf{i}+\mathbf{j}} B_{\mathbf{j}-\mathbf{e}_1}^{r} + v\mathbf{b}_{\mathbf{i}+\mathbf{j}} B_{\mathbf{j}-\mathbf{e}_2}^{r} + w\mathbf{b}_{\mathbf{i}+\mathbf{j}} B_{\mathbf{j}-\mathbf{e}_3}^{r}\right] \qquad (\star) \\ &= \sum_{|\mathbf{j}|=r+1} \mathbf{b}_{\mathbf{i}+\mathbf{j}} B_{\mathbf{j}}^{r+1}, \end{aligned}$$

where the line $(\star)$ is justified by the definition of $B_{\mathbf{i}}^{r}$ for illegal multi-indices. By setting $r = n$ in Equation (3.10) the following form is obtained:

The Bernstein form of a triangular Bézier patch:

$$\mathbf{b}^{n}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^{n}(\mathbf{u}). \tag{3.11}$$

The scheme used to index the control points is now clear. Each summand in equation (3.11) is of the form $\binom{n}{ijk} u^i v^j w^k \mathbf{b}_{ijk}$, so each control point is labeled according to its coefficient in the sum. Each label corresponds to the multi-exponent of uvw and the trinomial coefficient scaling it in the sum.

3.5 Polar Forms and the Blossoming Principle

Though Equation (3.11) significantly tidies up the formulation of Bézier patches, it is still somewhat of a black box. A more intuitive representation, which will be of use later, is provided using polar forms. Lyle Ramshaw [Ram88] has applied these forms to what he termed the *Blossoming Principle*, essentially supplying a notational tool which simplifies further development. It is presented now.

A map $f : R^n \to R^m$ is called *symmetric* if $f(x_1, \ldots, x_n) = f(x_{\pi(1)}, \ldots, x_{\pi(n)})$ for all permutations $\pi \in \Sigma_n$, the symmetric group of permutations of order n. f is called *multiaffine* if it is multilinear with respect to affine combinations: $f(\alpha x + \beta y, x_2, \ldots, x_n) = \alpha f(x, x_2, \ldots, x_n) + \beta f(y, x_2, \ldots, x_n)$ when $\alpha + \beta = 1$. Based on the classical principle in mathematics that polynomials in a single variable of degree n are equivalent to polynomials in n variables that are linear in each, [Ram88] and [GS93] show that polynomials $F : R^2 \to R^t$ of degree n and symmetric multiaffine maps $f : (R^2)^n \to R^t$ are equivalent, so that given any map of one type there exists a unique one of the other satisfying $F(\mathbf{u}) = f(\mathbf{u}, \ldots, \mathbf{u})$. Ramshaw called this the *Blossoming principle*, and called f the *polarization* or *blossom* of F. In classical multilinear algebra, f is called the *polar form* of F. For convenience, the multiplicative notation is often adopted when specifying the parameters of the polar form, writing $f(\mathbf{u}_1 \cdots \mathbf{u}_n)$ in lieu of $f(\mathbf{u}_1, \ldots, \mathbf{u}_n)$ which, in light of the symmetry, is further contracted to $f(\mathbf{u}_i^{\alpha_i} \cdots \mathbf{u}_j^{\alpha_j})$ when arguments are repeated. For the sake of clarity, however, the standard and the multiplicative notations will be mixed at times. Both the polynomial and the polar form of a surface will be denoted with the same letter, and will be distinguished by their respective arities and bracketing.

The de Casteljau algorithm is now revisited with the aid of the polar form. The blossoming principle guarantees the existence of the polar form of a Bézier triangle. Since the patch point $\mathbf{b}^n(\mathbf{u})$ is expressed both as a polynomial in $\mathbf{u}$ (equation (3.11)) and as an n-step algorithm with parameter $\mathbf{u}$ (equation (3.5)), the uniqueness of the polar form implies that the latter represents the polarization of the former. Equation (3.5) is, therefore, the blossom $\mathbf{b}[\mathbf{u} \cdots \mathbf{u}] = \mathbf{b}[\mathbf{u}^n]$ of the polynomial surface $\mathbf{b}^n(\mathbf{u})$. The de Casteljau algorithm calls for n recursive steps to be taken with a fixed parameter $\mathbf{u}$. The argument used by each such step corresponds to one of the parameters in the polar form. Because all the recursive steps make use of the same value, the argument to the blossom is an n-fold repetition of $\mathbf{u}$.

There is nothing mysterious about the polar form of $\mathbf{b}^n$; it is just an ordinary map. As such, the effects of the n parameters on its value can be explored. Consider $\mathbf{b}[\mathbf{e}_1, \mathbf{u}^{n-1}]$: one de Casteljau step is carried out with respect to $\mathbf{e}_1$ and $n-1$ steps of the standard algorithm are performed with respect to $\mathbf{u}$. But when $|\mathbf{i}| = n-1$, $\mathbf{b}^1_{\mathbf{i}}(\mathbf{e}_1) = \mathbf{b}_{i+1,j,k}$, so taking a de Casteljau step with $\mathbf{e}_1$ will produce a mesh of degree $n-1$ which is identical to the original mesh, only that the left diagonal has been removed. Similarly, taking a step with $\mathbf{e}_2$ peels away the bottom row, while $\mathbf{e}_3$ peels away the right diagonal. By successively eliminating specific diagonals from the control net, the set of control points can be reduced to any one point. It is easily seen, then, that the control point $\mathbf{b}_{\mathbf{i}} = \mathbf{b}_{ijk}$ can be found by removing the i leftmost diagonals, the j bottom diagonals, and the k rightmost ones. In other words, each control point is given by [Ram88]

$$\mathbf{b}_{\mathbf{i}} = \mathbf{b}[\mathbf{e}_1^i, \mathbf{e}_2^j, \mathbf{e}_3^k]. \tag{3.12}$$

Of course, because the blossom is symmetric it is also true that

$$\mathbf{b}_{\mathbf{i}} = \mathbf{b}[\mathbf{e}_2^j, \mathbf{e}_1^i, \mathbf{e}_3^k].$$

Using this notation, the intermediary points of the de Casteljau algorithm may also be expressed by:

$$\mathbf{b}^r_{\mathbf{i}}(\mathbf{u}) = \mathbf{b}[\mathbf{u}^r, \mathbf{e}_1^i, \mathbf{e}_2^j, \mathbf{e}_3^k], \qquad \text{with} \quad i+j+k+r = n.$$

3.6 Derivatives

Because the Bézier surfaces are defined in terms of barycentric coordinates which use three parameters to describe what is essentially a two-dimensional space, partial derivatives have no geometric interpretation. In their stead the notion of directional derivatives is used.

3.6.1 General Derivation

Every affine space has an associated linear vector space of the same dimension, whose elements are called vectors and are characterized by the property that their components sum to zero. The difference of two barycentric points $\mathbf{d} = (d, e, f) = \mathbf{u}_1 - \mathbf{u}_2$ is a vector because $|\mathbf{d}| = |\mathbf{u}_1| - |\mathbf{u}_2| = 1 - 1 = 0$. Following [Far86] the *directional derivative* of a surface is defined as $\mathbf{x}$ in direction $\mathbf{d}$ by

$$D_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = \sum_{i=1}^{3} \frac{\partial \mathbf{x}}{\partial x_i}(\mathbf{u}) d_i = d\mathbf{x}_u(\mathbf{u}) + e\mathbf{x}_v(\mathbf{u}) + f\mathbf{x}_w(\mathbf{u}).$$

Partials of the surface $\mathbf{b}^n$ are easily computed:

$$\begin{aligned}
\mathbf{b}_u^n(\mathbf{u}) &= \frac{\partial}{\partial u} \sum_{|\mathbf{i}|=n} \frac{n!}{i!j!k!} u^i v^j w^k \mathbf{b}_{\mathbf{i}} \\
&= n \sum_{|\mathbf{i}|=n} \frac{(n-1)!}{(i-1)!j!k!} u^{i-1} v^j w^k \mathbf{b}_{\mathbf{i}} \\
&= n \sum_{|\mathbf{i}|=n-1} \frac{(n-1)!}{i!j!k!} u^i v^j w^k \mathbf{b}_{i+1,j,k} \\
&= n \sum_{|\mathbf{i}|=n-1} \mathbf{b}_{\mathbf{i}+\mathbf{e1}} B_{\mathbf{i}}^{n-1}(\mathbf{u}).
\end{aligned}$$

A similar computation for $\mathbf{b}_v^n$ and $\mathbf{b}_w^n$ shows that

$$\begin{aligned}
D_{\mathbf{d}}\mathbf{b}(\mathbf{u}) &= n \sum_{|\mathbf{i}|=n-1} \left[d\mathbf{b}_{\mathbf{i}+\mathbf{e}_1} + e\mathbf{b}_{\mathbf{i}+\mathbf{e}_2} + f\mathbf{b}_{\mathbf{i}+\mathbf{e}_3} \right] B_{\mathbf{i}}^{n-1}(\mathbf{u}) \\
&= n \sum_{|\mathbf{i}|=n-1} \mathbf{b}_{\mathbf{i}}^{\mathbf{1}}(\mathbf{d}) B_{\mathbf{i}}^{n-1}(\mathbf{u}).
\end{aligned}$$

Thus the directional derivative is obtained by taking one de Casteljau step with respect to $\mathbf{d}$ and the remaining $n-1$ steps with respect to $\mathbf{u}$. This calls for blossoming notation:

$$D_{\mathbf{d}}\mathbf{b}(\mathbf{u}) = n\mathbf{b}[\mathbf{d}, \mathbf{u}^{n-1}], \tag{3.13}$$

and because the polar form is symmetric it follows that

$$D_{\mathbf{d}}\mathbf{b}(\mathbf{u}) = n\mathbf{b}[\mathbf{u}^{n-1}, \mathbf{d}]. \tag{3.14}$$

Continuing with this process, it is possible to generalize (as is done in [GS93, Far93]) and obtain the general form of the r-th directional derivative

$$D^r_{\mathbf{d}}\mathbf{b}(\mathbf{u}) = \frac{n!}{(n-r)!}\mathbf{b}[\mathbf{d}^r, \mathbf{u}^{n-r}].$$

3.6.2 Computing Surface Normals

Equation (3.14) shows that the directional derivative of the surface $\mathbf{b}^n$ at the point $\mathbf{u}$ is obtained by iterating through the de Casteljau algorithm $n-1$ times using the parameter $\mathbf{u}$, and then once with the vector $\mathbf{d}$. That is, the directional derivative is produced as a weighted average of the three Bézier points of level $n-1$:

$$D_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = dn\mathbf{b}^{n-1}_{\mathbf{e}_1}(\mathbf{u}) + en\mathbf{b}^{n-1}_{\mathbf{e}_2}(\mathbf{u}) + fn\mathbf{b}^{n-1}_{\mathbf{e}_3}(\mathbf{u}).$$

This equation holds for every direction $\mathbf{d}$, which means that these three points span the plane tangent to the surface at the point $\mathbf{b}^n(\mathbf{u})$. Thus the surface normal $\mathbf{n}$ – the unit vector perpendicular to the tangent plane – at the point $\mathbf{u}$ is given by:

$$\mathbf{n}(\mathbf{u}) = \frac{D_{\epsilon_1}\mathbf{b}^n(\mathbf{u}) \wedge D_{\epsilon_2}\mathbf{b}^n(\mathbf{u})}{\|D_{\epsilon_1}\mathbf{b}^n(\mathbf{u}) \wedge D_{\epsilon_2}\mathbf{b}^n(\mathbf{u})\|} \tag{3.15}$$

where

$$\epsilon_1 = \mathbf{e}_1 - \mathbf{e}_2 = (1, -1, 0), \qquad \epsilon_2 = \mathbf{e}_3 - \mathbf{e}_2 = (0, -1, 1),$$

and the wedge '$\wedge$' denotes the vector (cross) product operator.

For Equation (3.15) to be meaningful, the generating points must all be distinct and non-collinear. This is generally the case in practice, but robustness problems may ensue if it is not. These may be overcome by, for example, linearly interpolating the normals on nearby vertices. When these points are consistently evaluated for every $\mathbf{u}$, the orientation of all patch normals will be consistent.

3.7 Degree Elevation

The ultimate reason for modeling using surface patches is to give the user easy control over the shape being modeled. This is attained by use of the control points: The modeler can interactively modify the surface shape by altering the spatial position of various control points. At times, however, the control points may not offer sufficient control over the surface, making it necessary to introduce additional ones while maintaining the surface shape. This is known as *degree elevation.* The process of degree elevation requires altering the position of all Bézier points but the ones in the corners, which are interpolated into the patch.

The goal at hand is to obtain a patch of a higher degree which is identical to the old one. That is, if the set $\{\mathbf{b_i}\}_{|\mathbf{i}|=n}$ comprises the control net of the degree-n patch $\mathbf{b}^n$, a new set of control points $\{\hat{\mathbf{b}}_\mathbf{i}\}_{|\mathbf{i}|=n+1}$ is to be acquired. This new set must produce the patch $\hat{\mathbf{b}}^{n+1}$ of degree $n+1$ which satisfies $\hat{\mathbf{b}}^{n+1}(\mathbf{u}) = \mathbf{b}^n(\mathbf{u})$ for all $\mathbf{u} \in \Delta$. Multiplying the RHS of the last identity by $|\mathbf{u}| = u + v + w = 1$ and using (3.11) gives rise to

$$\begin{aligned}
\sum_{|\mathbf{i}|=n+1} \hat{\mathbf{b}}_\mathbf{i} B_\mathbf{i}^{n+1}(\mathbf{u}) &= |\mathbf{u}| \sum_{|\mathbf{i}|=n} \mathbf{b_i} B_\mathbf{i}^n(\mathbf{u}) \\
&= \sum_{|\mathbf{i}|=n} \frac{n!}{i!j!k!} \left[u^{i+1}v^j w^k + u^i v^{j+1} w^k + u^i v^j w^{k+1}\right] \mathbf{b_i} \\
&= \frac{1}{n+1} \sum_{|\mathbf{i}|=n} \left\{ \frac{(n+1)!}{(i+1)!j!k!}(i+1)u^{i+1}v^j w^k + \right. \\
&\qquad \frac{(n+1)!}{i!(j+1)!k!}(j+1)u^i v^{j+1} w^k + \\
&\qquad \left. \frac{(n+1)!}{i!j!(k+1)!}(k+1)u^i v^j w^{k+1} \right\} \mathbf{b_i} \\
&= \frac{1}{n+1} \sum_{|\mathbf{i}|=n+1} \binom{n}{ijk} \left\{ i\mathbf{b}_{i-1,j,k} + j\mathbf{b}_{i,j-1,k} + k\mathbf{b}_{i,j,k-1} \right\} u^i v^j w^k.
\end{aligned}$$

By equating coefficients of like powers on both sides and setting $\mathbf{b_i} = 0$ whenever $|\mathbf{i}| > n$ or one of the terms in multi-index is negative, the desired Bézier points are determined:

$$\hat{\mathbf{b}}_{\mathbf{i}} = \frac{1}{n+1}(i\mathbf{b}_{\mathbf{i}-\mathbf{e}_1} + j\mathbf{b}_{\mathbf{i}-\mathbf{e}_2} + k\mathbf{b}_{\mathbf{i}-\mathbf{e}_3}), \qquad |\mathbf{i}| = n+1. \tag{3.16}$$

This result has a geometric interpretation. Because each the new control point is produced by an affine combination of old ones, the new control net is contained in the convex hull of the old one.

The process of degree elevation may be repeated indefinitely. Farin [Far86] proves that the nets obtained by repeated degree elevations converge uniformly to the patch surface. In other words, if one denotes the original control net by $\mathcal{B}$, and the process of degree elevation by the operator $\mathcal{E}$, Farin shows that

$$\lim_{r\to\infty} \mathcal{E}^r \mathcal{B} = \mathbf{b}^n. \tag{3.17}$$

He notes, however, that just as in the case of curves this convergence is too slow to be of any practical use.

3.8 Extending the Domain Simplex

As polynomials, triangular patches are defined over the entire plane and are not necessarily restricted to their domain simplex. For reasons that will become evident shortly, we consider the effects of transforming the domain simplex of the patch $\mathbf{b}^n$. In particular, if Δ is the domain triangle of $\mathbf{b}^n$ which is derived from three points $(\mathbf{q}, \mathbf{r}, \mathbf{s})$, we want to determine a new set of control points that define the same surface when the domain triangle is switched to $\hat{\Delta}$, which is derived from $(\hat{\mathbf{q}}, \mathbf{r}, \mathbf{s})$ (Figure 3.6).

Some notation is required first. A superscripted multi-index will be used to denote a multi-index whose first component is fixed at the superscripted value, *e.g.*, $\mathbf{i}^r = (r, j, k)$ where $j + k = n - r$. A similar scheme will be used for barycentric tuples, thus $\mathbf{u}^s = (s, v, w)$ where $v + w = 1 - s$.

The line $\overline{\mathbf{rs}}$ is common to both domain triangles and corresponds to the parameters

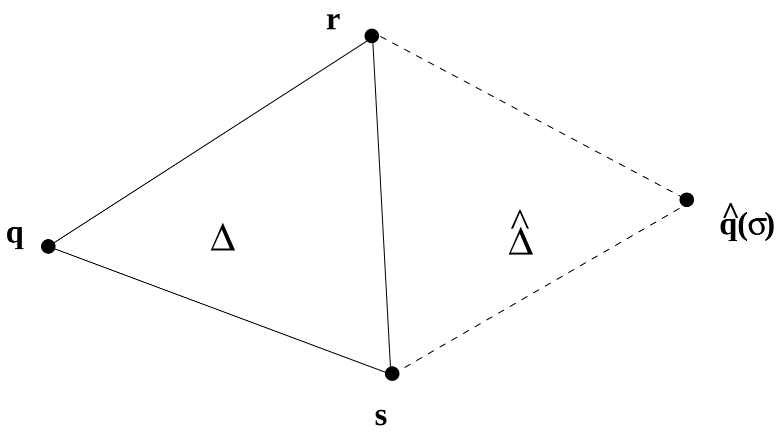

Figure 3.6: Extending the domain simplex

$\mathbf{u}^0 = (0, v, w)$ and $\hat{\mathbf{u}}^0 = (0, \hat{v}, \hat{w})$ in Δ and $\hat{\Delta}$ respectively. The domain triangles are coplanar, so $\hat{\mathbf{q}}$ can be given as a barycentric combination of $(\mathbf{q}, \mathbf{r}, \mathbf{s})$. Say it has coordinates σ with respect to Δ. Notice that this combination will not be *convex* since the point $\hat{\mathbf{q}}$ is external to the triangle $(\mathbf{q}, \mathbf{r}, \mathbf{s})$. Finally, let $\hat{\mathbf{b}}^n$ having control net $\{\hat{\mathbf{b}}_{\mathbf{i}}\}$ be a triangular Bézier patch defined over $\hat{\Delta}$. Farin [Far86] has proved that, with the above notation, the patches $\mathbf{b}^n$ and $\hat{\mathbf{b}}^n$ are equivalent if and only if

$$\hat{\mathbf{b}}_{\mathbf{i}^r} = \mathbf{b}^r_{\mathbf{i}^0}(\sigma), \qquad 0 \le r \le n, \tag{3.18}$$

or, in blossoming notation,

$$\hat{\mathbf{b}}_{\mathbf{i}^r} = \mathbf{b}[\sigma^r, \mathbf{e}_2^j, \mathbf{e}_3^k] \tag{3.19}$$

The patch $\hat{\mathbf{b}}^n$ is easily constructed from $\mathbf{b}^n$ by executing the de Casteljau algorithm with the parameter σ. Each iteration produces the new control points on the $\mathbf{i}^0$ diagonal of the mesh. These control points can be collected during the process to construct the complete Bézier net of $\hat{\mathbf{b}}^n$.

The next two sections will discuss two important applications of condition (3.18).

3.9 Cross Boundary Continuity Conditions

It is generally the case that a surface is modeled using a collection of Bézier patches that are stitched together at the seams. To ensure an overall smoothness of the surface, strict continuity constraints must be imposed along the shared patch boundaries. The continuity conditions necessary for patches $\mathbf{b}^n$ and $\mathbf{c}^n$ to be joined along the diagonal corresponding to $\mathbf{i}^0 = (0, j, k)$ will now be addressed.

3.9.1 C^0 Continuity

C^0 continuity requires only that both patches be adjacent and have no cracks along the common boundary. It has been noted earlier that these boundary diagonals are just plain Bézier curves, and as such are determined completely by the control points along the corresponding edges of the mesh. The desired continuity conditions are obtained by constraining these control points to be identical. Thus patches $\mathbf{b}^n$ and $\mathbf{c}^n$ can be made C^0 continuous along $\mathbf{i}^0$ be requiring that

$$\mathbf{b}_{\mathbf{i}^0} = \mathbf{c}_{\mathbf{i}^0}. \tag{3.20}$$

3.9.2 C^1 Continuity

Since points along the common boundary belong to both patches, C^1 continuity requires the tangent vectors associated with each of these points on both patches to coincide. As mentioned in Section 3.8, the boundary points are all parameterized by some $\mathbf{u}^0 = (0, j, k) = (0, j, 1-j) \in \Delta$. For any direction $\mathbf{d}$ not parallel to the diagonal, Equation (3.13) asserts that

$$D_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}^0) = n \sum_{|\mathbf{i}_0|=n-1} \mathbf{b}^1_{\mathbf{i}_0}(\mathbf{d}) B^{n-1}_{\mathbf{i}_0}(\mathbf{u}^0)$$

where

$$\mathbf{b}^1_{\mathbf{i}_0}(\mathbf{d}) = d\mathbf{b}_{1,j,k} + e\mathbf{d}_{0,j+1,k} + f\mathbf{b}_{0,j,k+1}.$$

So the derivative depends only on the two diagonals of control points indexed by $\mathbf{i}^0$ and $\mathbf{i}^1$.

Because the C^1 constraint is stronger than C^0, it has already been required that border diagonals coincide, so what remains is to ascertain the requirements on the other control points. Equation (3.18) offers a necessary and sufficient condition for $\mathbf{b}^n$ and $\mathbf{c}^n$ to coincide, and hence for all their derivatives up to degree n to agree. By requiring this to hold only for the first degree, the C^1 continuity condition is obtained:

$$\mathbf{c}_{\mathbf{i}^1} = \mathbf{b}^1_{\mathbf{i}^0}(\sigma) = \sigma_1 \mathbf{b}_{1,j,k} + \sigma_2 \mathbf{b}_{0,j+1,k} + \sigma_3 \mathbf{b}_{0,j,k+1}, \tag{3.21}$$

where $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ is the barycentric coordinates of the one vertex of the domain triangle of $\mathbf{c}^n$ not shared with that of $\mathbf{b}^n$.

The same σ is used to constrain all the control points $\mathbf{c}_{\mathbf{i}^1}$. Thus not only will the sub-triangle pairs formed by each set of four control points be coplanar, the parameter σ will make each an affine map of the domain triangles (see Figure 3.7).

As an example, consider the case when both domain triangles are equilateral. The triangles $(\mathbf{p}, \mathbf{q}, \mathbf{r})$ and $(\hat{\mathbf{p}}, \mathbf{q}, \mathbf{r})$ are equal in area and opposite in orientation. It follows from Equation (3.3) then that $\sigma_1 = -1$ and, because both are equilateral, that $\sigma_2 = \sigma_3 = 1$. Thus $\sigma = (-1, 1, 1)$ and the continuity condition:

$$\mathbf{c}_{\mathbf{i}^1} = -\mathbf{b}_{\mathbf{i}^0+\mathbf{e}_1} + \mathbf{b}_{\mathbf{i}^0+\mathbf{e}_2} + \mathbf{b}_{\mathbf{i}^0+\mathbf{e}_3}, \qquad |\mathbf{i}^0| = n - 1, \tag{3.22}$$

is obtained. This combines with the C^0 condition (3.20) to produce what is known as the *rhombus condition* [Far79] (as cited in [HL93])

$$\mathbf{c}_{1,j,k} - \mathbf{c}_{0,j,k+1} = \mathbf{b}_{0,j+1,k} - \mathbf{b}_{1,j,k}. \tag{3.23}$$

That is, the cross-border edges of the Bézier net are pairwise parallel and of equal length (see Figure 3.7).

3.9.3 G^1 continuity

As is evident from Figure 3.7, the insistence on C^1 continuity imposes some rather severe constraints on the shape of the patch. For most purposes geometric continuity provides sufficient smoothness along the common boundaries.

G^1 continuity is established by relaxing the C^1 requirement that the pairs of

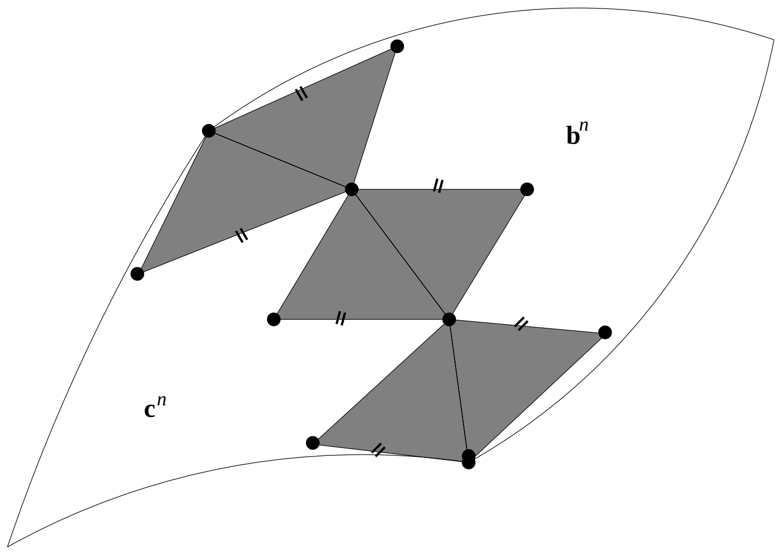

Figure 3.7: The shaded triangles must be coplanar and an affine map of the domain triangles

tangent *vectors* coincide for points on the boundary, and necessitating only the coincidence of the tangent *planes*. The tangent planes are defined by the four points $\mathbf{b}_{0,j+1,k} = \mathbf{c}_{0,j+1,k}$, $\mathbf{b}_{0,j,k+1} = \mathbf{c}_{0,j,k+1}$, $\mathbf{b}_{1,j,k}$, and $\mathbf{c}_{1,j,k}$. The coplanarity condition can be attained by requiring that $\mathbf{c}_{1,j,k}$ be a barycentric combination of the other three; that is,

$$\mathbf{c}_{1,j,k} = \nu_1 \mathbf{b}_{0,j+1,k} + \nu_2 \mathbf{b}_{0,j,k+1} + \nu_3 \mathbf{b}_{1,j,k}, \tag{3.24}$$

where $|\nu| = |(\nu_1, \nu_2, \nu_3)| = 1$.

3.10 Patch Subdivision

It is sometimes the case that a given patch cannot accommodate the constraints of the model. This may occur, for example, when imposing various continuity conditions among three patches: The continuity conditions constrain the Bézier points on the diagonal "parallel" to a given boundary. Because any two boundary diagonals of a single patch share a point in common, that point may have been multiply constrained in an inconsistent manner (this issue will be discussed at length later in Chapter 5). For the moment we note that such patches are called *over-constrained*. One solution

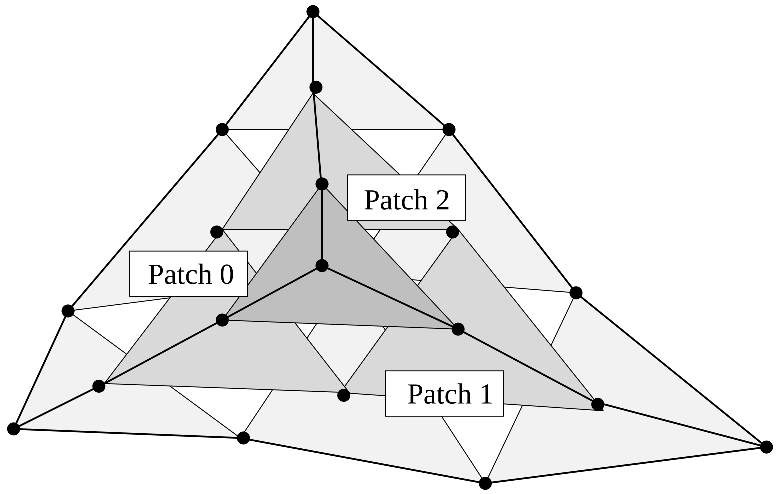

Figure 3.8: Subdividing a patch

to this problem is to split the patch into two or three smaller ones. Each subpatch would inherit some (but not all) of the constraints from the original patch, and the system would be resolved.

Section 3.8 already described how any given patch may undergo a domain simplex transformation. If σ is taken inside the domain Δ of $\mathbf{b}^n$, then Equation (3.18) provides a *subdivision algorithm*. The patches obtained through this process are labeled according to the diagonals they share with the original patch. Unraveling Equation (3.18), one finds that the control meshes of the three subpatches are given by

$$\begin{aligned}
&\text{Patch 0:} && \{\mathbf{b}^r_{0jk} | j + k + r = n\}, \\
&\text{Patch 1:} && \{\mathbf{b}^r_{i0k} | i + k + r = n\}, \quad \text{and} \\
&\text{Patch 2:} && \{\mathbf{b}^r_{ij0} | i + j + r = n\}.
\end{aligned}$$

These are shown in Figure 3.8.

3.11 Improving Efficiency

Theoretical considerations notwithstanding, the rendering of a Bézier patch demands that a polygonalized representation of the surface be shipped down the graphics pipeline. This is ultimately achieved through the process of sampling, by which the surface is evaluated at each point in a set of parameter values. This leads to the consideration of the computational complexity involved in this process.

There are two basic approaches to sampling: patch subdivision, and direct polynomial evaluation. The former is the more popular and is based on the fact that the control net approximates the surface and converges to it (Section 3.10). Patches are recursively subdivided until the convex hull of the control net of each subpatch is deemed sufficiently flat, at which time all but the corner control points of each net are discarded [WW92]. Various subdivision based evaluation schemes are compared with the de Casteljau algorithm in [Car95] before it is concluded that the de Casteljau method is the most cost-effective technique.

A faster method is now shown. In what follows, an estimate will be obtained for the cost of evaluating one surface point on a patch of degree-n using two evaluation techniques: the de Casteljau algorithm and direct evaluation of the polynomial using Horner's rule.

As the speed of basic operations varies between machines, the cost will be considered in terms of the number of addition and multiplication operations involved in each technique. Assuming that points in space are represented in non-homogeneous coordinates, each is composed of three numbers. The addition of two points is performed componentwise, essentially requiring the performance of three separate operations. The same holds true for multiplication of a point by a scalar. In order to estimate the operational cost of each algorithm, two functionals $\mathrm{cost}^{\theta}_{dC}$ and $\mathrm{cost}^{\theta}_{H}$ are defined, where θ is $+$ or $\times$, counting the number of scalar additions and multiplications involved in the evaluation using the de Casteljau algorithm or direct evaluation using Horner's rule respectively.

When considering the de Casteljau algorithm, the term *level* will be used to denote the number of iterations remaining before the algorithm is terminated. An analysis of the algorithm reveals that n iterations are required in order to evaluate a single point on the patch; level-n marks the beginning of the algorithm, and level-0 marks the end. The Bézier net of level-k is composed of $\binom{k+2}{2}$ control points, which make $\binom{k+1}{2} = k(k+1)/2$ point-triples that will be interpolated. Every such interpolation demands three multiplications of points and two additions of points, and so requires

9 multiplication and 6 addition operations. Thus the iteration at level-k requires $9k(k+1)/2$ multiplications, and $6k(k+1)/2 = k(k+1)$ additions. The entire algorithm therefore requires $9\sum_{k=1}^{n} k(k+1)/2$ multiplications, and $3\sum_{k=1}^{n} k(k+1)$ additions. Now

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3},$$

so the computational requirements of the de Casteljau algorithm are:

$$\mathrm{cost}_{dC}^{+} = n(n+1)(n+2), \qquad \text{and} \tag{3.25}$$

$$\mathrm{cost}_{dC}^{\times} = 3n(n+1)(n+2)/2. \tag{3.26}$$

The polynomial $y(x) = \sum_{i=0}^{n} a_i x^i$ can be unraveled using Horner's rule to the form $y(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n))\cdots)$, which is evaluated at the cost of n additions and n multiplications. Evaluating the same polynomial using the straightforward, naive approach would require an equal number of additions, but the number of multiplications would increase to $\sum_{i=0}^{n} = n(n+1)/2 \in O(n^2)$, an order of magnitude higher. The same philosophy is applied to the Bernstein form of the Bézier triangles. Equation (3.11) can be repackaged for Horner's rule as

$$\mathbf{b}^n(\mathbf{u}) = \sum_{i=0}^{n}\sum_{j=0}^{n-i}\binom{n}{ijk}\mathbf{b}_{ijk}u^i v^j w^k = \sum_{i=0}^{n} S_i u^i, \tag{3.27}$$

where $S_i = \sum_{j=0}^{n-i} T_{ij} v^j \mathbf{b}_{ijk}$, and $T_{ij} = \binom{n}{i,j,k} w^k$. The cost of evaluating (3.27) is analyzed as follows:

T_{ij}: The multinomial coefficients do not depend on the parameter u and thus may be computed in advance for the entire patch and stored in a lookup table (Appendix A.1 discusses the efficient computation of these). Each T_{ij} can thus be computed at the cost of zero additions and $k = n - i - j$ multiplications: $k - 1$ of which are required for evaluating w^k, and one for the multinomial coefficient[3].

[3] A method proposed by Stewart [Ste96] uses the binary representation of k to evaluate w^k at the cost of $O(\log k)$ multiplications. In practice, however, the method is not cost-effective as n is generally too small.

S_i: Each S_i is a polynomial of degree $n-i$ in v whose coefficients are points, and thus requires $(n-i)$ additions of points, $n-i$ multiplications of two scalars ($T_{ij}v$), and $n-i$ multiplications of points by scalars. All the necessary coefficients T_{ij} must also be computed. Thus,

$$\begin{aligned}
\mathrm{cost}_H^{+}(S_i) &= 3(n-i) + \sum_{j=0}^{n-i} \mathrm{cost}_H^{+}(T_{ij}) = 3(n-i), \qquad \text{and} \\
\mathrm{cost}_H^{\times}(S_i) &= 4(n-i) + \sum_{j=0}^{n-i} \mathrm{cost}_H^{\times}(T_{ij}) \\
&= 4(n-i) + \sum_{j=0}^{n-i}(n-i-j) \\
&= 4(n-i) + (n-i)(n-i+1)/2 \\
&= (n-i)(n-i+9)/2.
\end{aligned}$$

Equation (3.27) is a polynomial of degree n in u with point coefficients, and thus involves $3n$ multiplications and an equal number of additions. As before, all the coefficients S_i must also be computed. Thus

$$\begin{aligned}
\mathrm{cost}_H^{+}(\mathbf{b}^n) &= 3n + \sum_{i=0}^{n} \mathrm{cost}_H^{+}(S_i) \\
&= 3n + \sum_{i=0}^{n} 3(n-i) \\
&= 3n + 3n(n+1)/2 \\
&= 3n(n+3)/2, \qquad \text{and} \qquad (3.28) \\
\mathrm{cost}_H^{\times}(\mathbf{b}^n) &= 3n + \sum_{i=0}^{n} \mathrm{cost}_H^{\times}(S_i) \\
&= 3n + \sum_{i=0}^{n}(n-i)(n-i+9)/2 \\
&= 3n + [n(n+1)(2n+1)/12 + 9n(n+1)/4] \\
&= n[(n+1)(n+14)/6 + 3]. \qquad (3.29)
\end{aligned}$$

This analysis shows that $\mathrm{cost}_H^{+}(\mathbf{b}^n) \ll \mathrm{cost}_{dC}^{+}(\mathbf{b}^n)$ and that $\mathrm{cost}_H^{\times}(\mathbf{b}^n) < \mathrm{cost}_{dC}^{\times}(\mathbf{b}^n)$, hence the direct evaluation using Horner's rule is cheaper than iterating through the de Casteljau algorithm. For comparison, the computed costs for $n = 2, \ldots, 9$ are summarized in the table below.

n	cost_H^+	cost_{dC}^+	$\mathrm{cost}_H^\times$	$\mathrm{cost}_{dC}^\times$
2	15	24	22	36
3	27	60	43	90
4	42	120	72	180
5	60	210	110	315
6	81	336	158	504
7	105	504	217	756
8	131	720	288	1080
9	162	990	372	1485

Table 3.1: Summary of computed costs of evaluation using Horner's rule and the de Casteljau algorithm

Two caveats are in order here. First, the most common surface patch used is a cubic. As such the difference in cost between de Casteljau evaluation and direct evaluation is not very significant. This distinction is further blurred when surface normals are computed. In this case Horner's method must be employed three times for each point in order to find the tangent plane (cf. Section 3.6.2), where each tangent plane point is of one degree less than the patch.

The second caveat is mentioned in [Far93]. The direct evaluation computes repeated products of coefficients that are represented as floating point numbers whose values lie between 0 and 1. Given the limited precision of numbers within this range it is not unlikely that errors will be incurred, leading to the numerical instability of the result for large surfaces.

3.12 Remarks

The Bézier triangle formulation presented in this chapter has been generalized by Seidel [Sei91] to a triangular element called a *B-patch*, and later by Greiner and Seidel [GS93] to a triangular bivariate *B-spline* patch. These have been used to construct piecewise polynomial surfaces over arbitrary triangulations.

Chapter 4

Texture mapping

Texture mapping is one of the most successful techniques used in computer graphics. It was first introduced by Catmull ([Cat74]) in 1974 as a means of enhancing the realism of rendered images and boosting their richness without affecting the complexity of their underlying geometry. Texture mapping has since become so prevalent that untextured images seem plastic and boring. In the recent Walt Disney/Pixar production *Toy Story*, nearly every character, and certainly every scene, has been enlivened with some texture.

The general idea behind texture mapping is based on a simple principle. Rather than model the microfacet geometry of the texture on a surface in all its minute detail, one can instead *shrink-wrap* a planar picture of the texture around the object. Though simple in principle, this shrink wrapping process is unfortunately complicated in practice. It invariably introduces spatial distortions to the texture patterns, which may considerably alter their visual appearance on the surface.

Following a somewhat more detailed description of the general problem and a survey of various mapping techniques, this chapter will discuss two methods of overcoming the difficulties. The first attempts to strike a balance between texture distortion and texture discontinuities, and is based on the work of Bennis et al. [BVI91]. It utilizes differential geometric methods to reduce the distortion incurred by preserving geodesic curvature on the surface through the mapping. The second method produces a surface area parameterization of the surface, which is then used as a cookie cutter

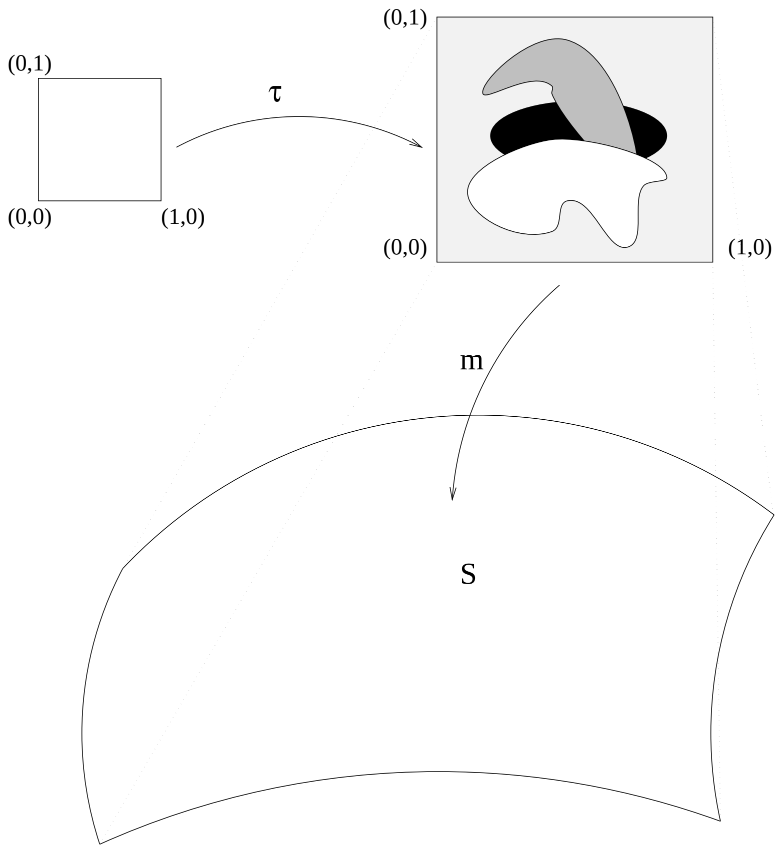

Figure 4.1: The texture mapping process

for the texture.

4.1 General Problem

An abstract definition of textures is provided in [FFC87]. Briefly, a *texture* is a correspondence $\tau : P \subset \mathbf{R}^2 \rightarrow C$ between a region P of the plane and a finite set of colors (or intensities) C, whereby each point in the region is assigned a color value. In practice, a texture is just a discrete image, which amounts to a two-dimensional array of color values. The object of texture mapping is to assign these values to every point on the surface in a convincing manner. Toward that end, a correspondence must be found between the surface and the region on the plane, by virtue of which the region will be wrapped nicely about the surface. See Figure 4.1.

Traditional texture mapping is performed in three stages. In the first, the texture is produced. This is usually done by either optically scanning a photograph, by manufacturing it manually using some drawing program, or by synthesizing the image procedurally and dynamically. In the second stage, the correspondence between the 3D surface and the 2D texture is established, whereby each surface point (x, y, z) is assigned a texture coordinate (u, v). This correspondence is called the *texture map.* Finally, the third stage assigns a color value to every surface point depending on the surrounding neighborhood of the corresponding texture point. For the purpose of simplicity, it will be assumed in the remainder of this paper that the third stage is straightforward and direct, even though, in practice, it is not. The color sampling generally produces undesired and inconsistent artifacts, and must be subjected to various antialiasing filtering techniques.

The texture map is nontrivial, as mapping a planar texture onto a three-dimensional surface without distortion is essentially equivalent to a non-warped flattening of the surface. In most cases, doing so is impossible. In fact, the small class of surfaces which can undergo a non-warped flattening has received a special name: such surfaces are called *developable surfaces* (accounts of which can be found in many places, *e.g.*, [Kre59] and [dC76]). By definition, then, fully spreading out a non-developable surface will necessarily introduce distortions. The ultimate goal of texture mapping is to minimize these.

More formally, this goal may be stated as follows. Given a surface $S \in \mathbf{R}^3$ with an area metric $\mathcal{A}_3 : S \to \mathbf{R}$ and a planar area metric $\mathcal{A}_2 : \mathbf{R}^2 \to \mathbf{R}$, we seek the following goal:

Goal. *Find a mapping $m : S \to \mathbf{R}^2$ s.t. for any small region $R \subset S$, the area quotient $\mathcal{A}_3(R)/\mathcal{A}_2(m(R)) \approx k$ for some constant k.*

In other words, the goal is to find a map that, for any small region on the surface with constant surface area A, its image A' on the plane is approximately constant. Note that in principle this definition may admit local anomalities in aspect ratio, so that a stronger constraint may be needed.

4.2 Survey of Various Mapping Techniques

The idea of using a texture map as a cheap means of obtaining extra image detail was first introduced by Catmull [Cat74], and made use of the existing surface parameterization. Because the spatial dimensions of the texture and of the parameter spaces are identical, Catmull proposed that the texture coordinates assigned to each surface point simply be the parameter coordinates. This method essentially takes a rectangular 2D texture and warps it through the parameterized surface formulation until it fits onto a 3D surface. Unfortunately, this process does not respect angles and distances, and as a result may distort a texture beyond recognition. Additional problems may arise along the boundaries of patches under this method [Blo85].

Blinn and Newell [BN76] propose the use of a two phase mapping process. In the first, the planar texture is embedded in an intermediate 3D object such as a box or a cylinder, heuristically chosen so as to minimize the incurred distortion. Phase two then projects this object onto the target surface. This method produces fairly good results when the objects are chosen correctly, but that is seldom an easy task.

In 1987, Fiume et al. [FFC87] proposed the use of *polygonal conformal mappings* to map one polygon onto an arbitrary convex polygon while preserving the angles between intersecting line segments. In particular, they proposed to map square pieces of texture onto planar polygonal elements (*e.g.*, triangles, quadrilaterals, and hexagons) on the surface being modeled. Distances, however, are not preserved by this technique, thus leaving the problem of distortions unresolved. Moreover, the line segments whose intersection angles have been preserved are typically mapped to arcs. Nevertheless, this method produces good results for some applications.

Other authors, such as Hanrahan and Haeberli [HH90] and Maillot et al. [MYV93], emphasize the importance of interactivity whilst mapping textures on 3D shapes. The user is provided with a set of mapping functions by which the texture plane can be altered.

A different class of textures which has recently received attention is that of procedural textures. These textures are synthesized with a particular object in mind, and can thus take into account specific aspects of its geometry throughout the generation process. By virtue of their algorithmic nature, procedural textures generally have some recurring pattern, and are thus useful for modeling various biological phenomena such as animal skins. One subclass of these procedural textures simulates a chemical processes of reaction and diffusion. The textures thus synthesized are known as *Reaction-Diffusion textures*; techniques for their generation have been suggested in [Tur91] and [WK91]. The methods range from pre-warping the texture during the generation process in order to compensate for the impending distortion ([WK91]), to generating a canonical surface mesh in lieu of any pre-existing one upon which the texture is grown ([Tur91]).

Although procedural textures are useful in many applications, their service is rather limited. The majority of textures are acquired digitally, precluding the use of creative synthesis techniques. Hence, a more general map is required to overcome texture distortion. Evidently no *grab-all* method exists. Various maps outperform others in different situations.

4.3 Surface Flattening Using Geodesic Curvature Preservation

Because of the impossibility of flattening a general surface perfectly, one is faced with two options when applying the texture. One option is to disregard the surface curvature, and to therefore accept the incurred distortions. The other is to break the surface up into regions of more-or-less constant curvature, and texture them individually. Surfaces produced in this way will suffer from many discontinuities in their texture.

Because neither option is particularly attractive, an alternative approach is to strike a balance between the two extremes. This latter approach is taken by the method described below. A surface is flattened along some parametric lines while keeping the incurred distortion beneath some tolerance level. The flattening method

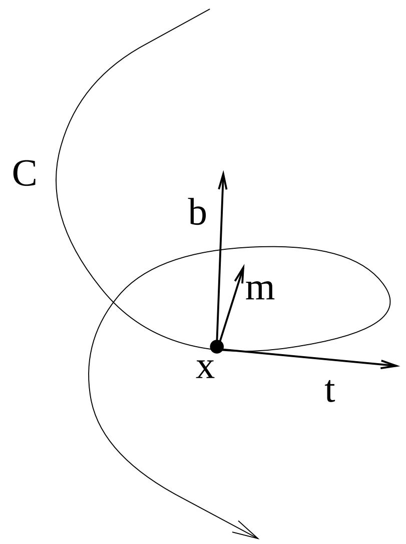

Figure 4.2: The Frenet frame

used is aimed at the preservation of the differential-geometric notion of *geodesic curvature*, and heavily relies on a preservation operator $\mathcal{P}$ which will presently be defined.

4.3.1 Local properties of curves and surfaces

A brief review of some necessary concepts of differential geometry is now presented, details of which can be found in [dC76] and [Wil59]. Let $C \subset \mathbf{R}^3$ be a space curve given by the arclength parameterization

$$C := \{\mathbf{x}(s) \in \mathbf{R}^3 \quad | \quad s \in [a, b] \subset \mathbf{R}\},$$

where the Cartesian coordinates (x, y, z) of $\mathbf{x}$ are each differentiable functions of the arclength s.

The *Frenet frame* (Figure 4.2) is a tool used in differential geometry to describe the local behavior of a curve in terms of a local coordinate system at each point. The Frenet frame $(\mathbf{x}, \mathbf{t}, \mathbf{m}, \mathbf{b})$ is an orthonormal frame centered at the point $\mathbf{x} \in C$, whose axes $(\mathbf{t}, \mathbf{m}, \mathbf{b})$ are given by the formulas

$$\mathbf{t} = \mathbf{x}', \qquad \mathbf{m} = \frac{\mathbf{x}''}{\|\mathbf{x}''\|}, \qquad \mathbf{b} = \mathbf{t} \wedge \mathbf{m},$$

where primes denote arclength derivatives, and the wedge '$\wedge$' denotes the vector (cross) product. The axes have received intuitive names: $\mathbf{t}(s)$ is called the *tangent vector*, $\mathbf{m}(s)$ is called the *main normal vector*, and $\mathbf{b}(s)$ is called the *binormal vector* vector of the curve at s. It is not difficult to show that the variation of the Frenet frame is given by the *Frenet-Serret* formulas:

$$\left\{\begin{array}{lcl} \mathbf{t}' & = & \kappa\mathbf{m} \\ \mathbf{m}' & = & -\kappa\mathbf{t} - \tau\mathbf{b} \\ \mathbf{b}' & = & -\tau\mathbf{m}. \end{array}\right.$$

The value $\kappa(s) = \|\mathbf{t}'(s)\|$ (recall that $\mathbf{m}$ has unit length) is non-negative, and is called the *main curvature* of $\mathbf{x}$ at s. It measures the rate at which the curve pulls away from its tangent at s. The value $|\tau(s)| = \|\mathbf{b}'(s)\|$, which could be positive *or* negative, is called the *torsion* of $\mathbf{x}$ at s. It measures how rapidly the curve pulls away from the plane determined by the tangent and main normal vectors – called the *osculating plane* – at s. The sign of the torsion determines the direction in which the curve points at the time at which it crosses the osculating plane.

Now consider the parametric surface $S \subset \mathbf{R}^3$ given by the bivariate parameterization

$$S := \{\mathbf{x}(u,v) \in \mathbf{R}^3 \quad | \quad (u,v) \in [a,b]^2 \subset \mathbf{R}^2\},$$

where the Cartesian coordinates of the surface points are differentiable functions of both parameters. A surface is called *orientable* if it makes sense to define an inside and an outside direction. An example of an orientable surface is a sphere; the classic example or a non-orientable surface is the so-called Möbius strip. The forthcoming development will be restricted to smooth orientable surfaces, in which $\mathbf{n}(u,v)$ is the unit outward-pointing *normal vector* to S at $\mathbf{x}(u,v)$. The *tangent plane*, $\mathbf{T}_{\mathbf{x}}(S)$, of S at $\mathbf{x}$ is spanned by the set of points $\mathbf{y} \in \mathbf{R}^3$ satisfying $(\mathbf{x} - \mathbf{y}) \cdot \mathbf{n} = 0$, where the notation $\mathbf{u} \cdot \mathbf{v}$ denotes the inner (dot) product of $\mathbf{u}$ and $\mathbf{v}$.

Let $C_S \subset \mathbf{R}^3$ be a curve on the surface S, given by arclength parameterization $\mathbf{x}(s)$. Its second arclength derivative $\mathbf{x}'' = \mathbf{t}'$, the acceleration of a particle moving along the curve, can be separated into two components. One, $\mathbf{t}'_n$, is normal to the

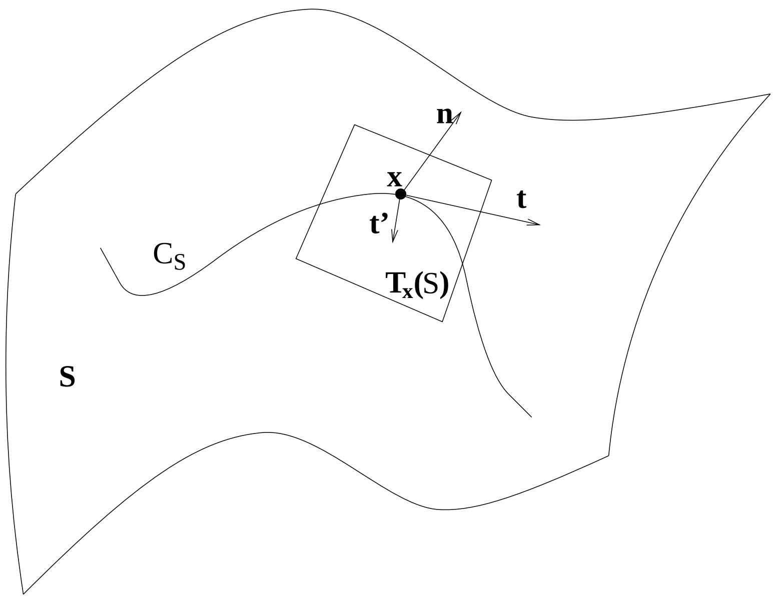

Figure 4.3: The curvature of a surface curve

surface; the other, $\mathbf{t}'_g$, is tangent to it:

$$\begin{aligned}\mathbf{t}'_n &= (\mathbf{t}' \cdot \mathbf{n})\mathbf{n} \\ \mathbf{t}'_g &= \mathbf{t}' - \mathbf{t}'_n.\end{aligned}$$

Looking from a viewpoint on the surface, only the tangential component can be seen (Figure 4.3). Based on this fact, we define a special kind of curvature for surface curves which is analogous to the notion of (plain) curvature of planar curves.

Definition. *The* geodesic curvature κ_g *of a curve* C_S *belonging to a surface* S *at the point* $\mathbf{x}$ *is the norm of the tangential acceleration of the curve at that point:*

$$|\kappa_g| = \|\mathbf{t}'_g\|.$$

This definition is not entirely precise since the geodesic curvature is a signed value, however for our purposes it suffices. Note that the *geodesic* curvature κ_g is not the same as the *main* curvature κ which contains the normal component as well.

Using geodesic curvature, the concept of a straight line can be extended from planes to surfaces. The curve C_S is said to be *geodesic at a point* $\mathbf{x}$ if κ_g vanishes there, and is called a *geodesic* if it is geodesic everywhere. That is, a surface curve is a geodesic if its acceleration is restricted to the "up" direction at every point. For an observer on the surface, a particle moving along a geodesic curve will appear to be moving in a straight line. Just as straight lines provide the shortest path between two points on a plane, geodesics provide the shortest path between two points on a surface. Moreover, if C_S is orthogonally projected onto the surface tangent plane $\mathbf{T}_{\mathbf{x}}(S)$ at a point $\mathbf{x}$, the projection curve $C_{T_x(S)}$ (on $\mathbf{T}_{\mathbf{x}}(S)$) will be locally straight in a neighborhood of $\mathbf{x}$ if and only if C_S is geodesic at $\mathbf{x}$. This is stated formally in the following lemma ([Wil59, dC76]):

Lemma. *The geodesic curvature κ_g of the curve C_S at the point $\mathbf{x} \in C_S$ is equal to the curvature at $\mathbf{x}$ of the planar curve $C_{T_x(S)}$ obtained by projecting C_S along the surface normal onto the tangent plane at $\mathbf{x}$.*

4.3.2 Faithful Surface Curve Projection

In practice, surfaces are sampled and surface curves are given by chord-line approximations whose vertices lie at the sampled points. Arclength is approximated by chordlength, given as the Euclidean distance $d(\cdot, \cdot)$ between two points. In what follows, let $S \subset \mathbf{R}^3$ be the sampled surface which is to be flattened, and let $F \subset \mathbf{R}^2$ be the flattening plane. The piecewise linear, planar curve $C_F \subset \mathbf{R}^2$ will be called a *faithful representation* of the discretized surface curve $C_S \subset \mathbf{R}^3$ if the curvature at each vertex on the chord-line C_F is identical to the geodesic curvature at the corresponding point on chord-line approximated C_S, and if the chordlengths of the corresponding segments on both curves agree.

The previous lemma offers a method for mapping a chord-line on a surface to a chord-line on a plane. The geodesic curvature of a curve can be preserved through the projection process, by using a projection map that respects the curvature of the local projection of the curve on the tangent plane at each point. The lemma provides the basis for just such an operator which, in addition to respecting curvature, preserves distances between neighboring points on isoparametric curves.

The operator is parameterized by two corresponding pairs of points, one on the surface, the other on the plane. Suppose $M_0, M_1, M_2 \in S$ are three neighboring surface points and that $P_0, P_1 \in F$ are the points on the flattening plane that correspond to the first two surface points. Ideally, the operator $\mathcal{P}_{P_0,P_1}^{M_0,M_1}$ will take the point $M_2 \in S$ and produce the point $P_2 \in F$ that simultaneously satisfies the two conditions:

$$\begin{array}{lrcll} (i) & \angle P_0 P_1 P_2 & = & \angle M_0 M_1 M_2, & \text{and} \\ (ii) & d(P_1, P_2) & = & d(M_1, M_2), & \end{array}$$

where $\angle PQR$ is the angle between the chords $\overline{PQ}$ and $\overline{QR}$.

Denoting by $\mathbf{T}_{\mathbf{P}_i}$ and $\mathbf{n}_i$ the tangent plane and surface normal at the point M_i, we define the operator $\mathcal{P}_{P_0,P_1}^{M_0,M_1} : S \rightarrow F$ which sends $M_2 \mapsto P_2$ by the following algorithm (see Figure 4.4):

Algorithm (Isometric and Conformal Operator):

1. *Project M_0 and M_2 onto $\mathbf{T}_{\mathbf{P}1}$, providing the points $\widetilde{M}_0$ and $\widetilde{M}_2$ given by:*

$$\widetilde{M}_i = M_i + ((M_1 - M_i) \cdot \mathbf{n}_1)\mathbf{n}_1, \qquad i = 0, 2.$$

2. *Dilate $\widetilde{M}_2$ in $\mathbf{T}_{\mathbf{P}1}$ giving $\widehat{M}_2$ such that $d(M_1, M_2) = d(M_1, \widehat{M}_2)$:*

$$\widehat{M}_2 = M_1 + \frac{\|M_2 - M_1\|}{\|\widetilde{M}_2 - M_1\|}(\widetilde{M}_2 - M_1).$$

3. *Define a right handed local, orthogonal coordinate system $(M_1, \hat{e}_1, \hat{e}_2)$ on $\mathbf{T}_{\mathbf{P}1}$ whose $\hat{e}_1$ axis is in the direction $M_1 - \widetilde{M}_0$, and let (ξ, η) be the coordinates of $\widehat{M}_2$ in that system. Then P_2 is the point with coordinates (ξ, η) in the local coordinate system (P_1, e_1, e_2) on F whose e_1 axis is in the direction $P_1 - P_0$:*

$$P_2 = P_1 + (\hat{x}_1 e_1 + \hat{x}_2 e_2),$$

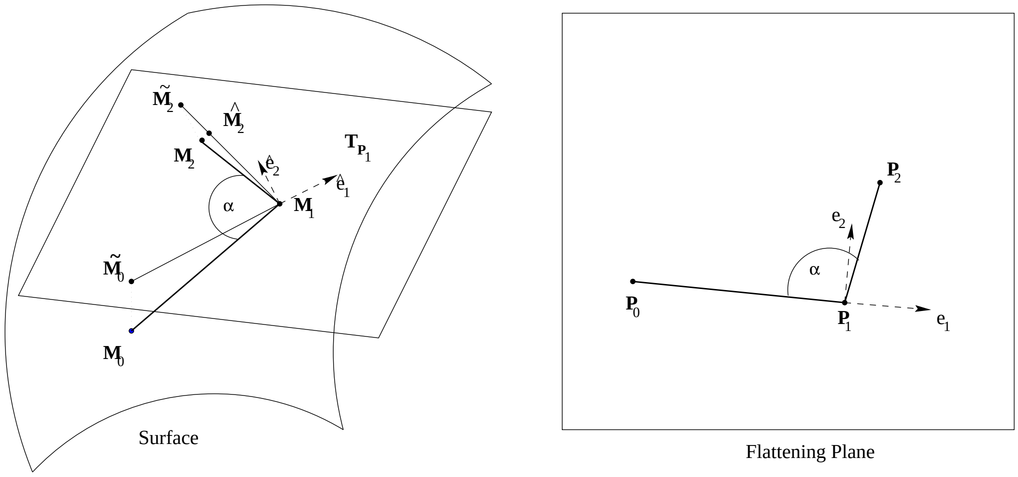

Figure 4.4: The preservation operator $\mathcal{P}$

where the axes $\hat{e}_1, \hat{e}_2, e_1, e_2$ and the coordinates (ξ, η) are given by

$$\begin{aligned}
\hat{e}_1 &= \frac{M_1 - \widetilde{M}_0}{||M_1 - \widetilde{M}_0||} \\
\hat{e}_2 &= \mathbf{n}_1 \times \hat{e}_1 \\
e_1 &= \frac{P_1 - P_0}{||P_1 - P_0||} = a\tilde{\imath} + b\tilde{\jmath} \\
e_2 &= -b\tilde{\imath} + a\tilde{\jmath} \\
\xi &= (\widehat{M}_2 - M_1) \cdot \hat{e}_1 \\
\eta &= (\widehat{M}_2 - M_1) \cdot \hat{e}_2
\end{aligned}$$

Given a chord line approximation of a surface curve, the operator $\mathcal{P}$ may be used to produce a planar projection that preserves chord-length and geodesic curvature. Suppose $M_0, \ldots, M_n \in S$ are $n+1$ surface points defining a curve which is to be projected onto the plane F, producing the planar curve $P_0, \ldots, P_n \in F$. Projecting the segment M_0M_1 requires nothing more than selecting a starting point P_0 and a direction, and placing P_1 at a distance of $d(M_0, M_1)$ from P_0 in the desired direction. The remaining remaining flattened points are given by

$$P_i = \mathcal{P}^{M_{i-2}, M_{i-1}}_{P_{i-2}, P_{i-1}}(M_i), \qquad i = 3, \ldots n.$$

Since $d(P_0, P_1) = d(M_0, M_1)$, and by step (2) of the algorithm $d(P_{i-2}, P_{i-1}) =$

$d(M_{i-2}, M_{i-1})$ for $i = 3, \ldots, n$, all arclengths are preserved. For a sufficiently refined sampling, the tangents to the curve are approximated by the chord segments, so the angles between successive chord segments on the tangent-plane projection $\mathbf{T}_{\mathbf{P}_i}$ approximate the curvature of the curve. The geodesic curvature at points $M_1, \ldots, M_{n-1}$ is thus preserved according to the lemma and, hence, the flattened curve is a faithful projection of the surface curve.

4.3.3 Surface Flattening

Bennis et al. [BVI91] propose several methods for piecewise surface flattening using the flattening operator $\mathcal{P}$. The general strategy, common to all methods, is to define a distortion metric and then flatten a region until the total distortion incurred in it reaches some threshold. When the threshold is surpassed the region is "closed" and flattening is resumed on a new region. In this way distortion is minimized inside the regions at the expense of introducing texture discontinuities between adjacent ones.

Regions are flattened by successively flattening sets of surface curves that cover them. Each individual region is *grown* by selecting an internal isoparametric curve and flattening it faithfully. Transversal isoparametric curves are developed in the same manner. Because flattening non-developable surfaces necessarily introduces distortion, the curves whose parameters run parallel to those of the original isoparametric one become distorted, since neither their geodesic curvature nor their arclength can be controlled. The various methods presented in [BVI91] differ in the techniques they use to alleviate this warping, and in the distortion metrics used in conjunction with them.

The most interesting method of development entails the flattening of the curves parallel[1] to the original one. Unlike the original curve, the flattening of these is guaranteed to preserve neither curvature nor arclength. It proceeds as follows. Suppose that $C'_S \subset S$ is a surface curve parallel to the original one, and that $M \in C'_S$ is a point on that curve. Assuming that S is sampled over a regular rectangular grid, M has

[1] In parameter space.

four neighbors N_0, N_1, N_2, and N_3. Each of these is surrounded by three additional neighboring points N_i^0, N_i^1, N_i^2; the fourth one N_i^3 being M (see Figure 4.5). For the time being, suppose that all these neighboring points have been previously projected onto the flattening plane F, so that $Q_i \in F$ corresponds to $N_i \in S$ and that $Q_i^j \in F$ corresponds to $N_i^j \in S$. Each of the neighbors N_i provides three different methods of applying the preservation operator $\mathcal{P}$ to M, producing the three (perhaps identical) points $\bar{P}^{ij} \in F$:

$$\bar{P}^{ij} = \mathcal{P}_{Q_i^j, Q_i}^{N_i^j, N_i}(M), \qquad j = 0, 1, 2.$$

These are called *candidate points*. Applying $\mathcal{P}$ in this way to all four neighbors of M produces 12 candidate points $\bar{P}^{ij}$, $(j \in \{0,1,2\}, i \in \{0,1,2,3\})$. Because the preservation operator requires many points Q_i and Q_i^j on the projection plane F that may not have been evaluated, the desired point $P = \mathcal{P}(M)$ is given as the centroid of all those candidates $\bar{P}^{ij}$ that were indeed available at the time of preservation.

For a developable surface S all the candidate points coincide, preserving the geodesic curvature and arclengths of both isoparametric curves of S intersecting at M. In general, however, this will not occur, and distortion will be introduced. Moreover, this distortion will be compounded as the distance between the parallel curves and the original one increases.

Two metrics for evaluating this distortion are suggested. Under the first metric, the distortive effect of the flattening is measured by comparing the arclengths of the surface curve segments with that of their projected counterparts, and summing up the discrepancies. With $\{M_i\} \subset C'_S$ being the set of consecutive points on the curve C'_S, and $\{P_i\} \subset F$ being their projection, the arclength distortion metric $\mathcal{D}_{\mathcal{A}}$ is given by

$$\mathcal{D}_{\mathcal{A}}(C'_S) = \sum_i \frac{|d(M_i, M_{i+1}) - d(P_i, P_{i+1})|}{d(M_i, M_{i+1})}.$$

Clearly a low value of $\mathcal{D}_{\mathcal{A}}$ indicates a low discrepancy between the corresponding arclengths, and a less distorted projection.

The second metric $\mathcal{D}_{\mathcal{D}}$ measures the distortive effect of the flattening by determining how widely dispersed the candidate points are. For a cluster of n coplanar

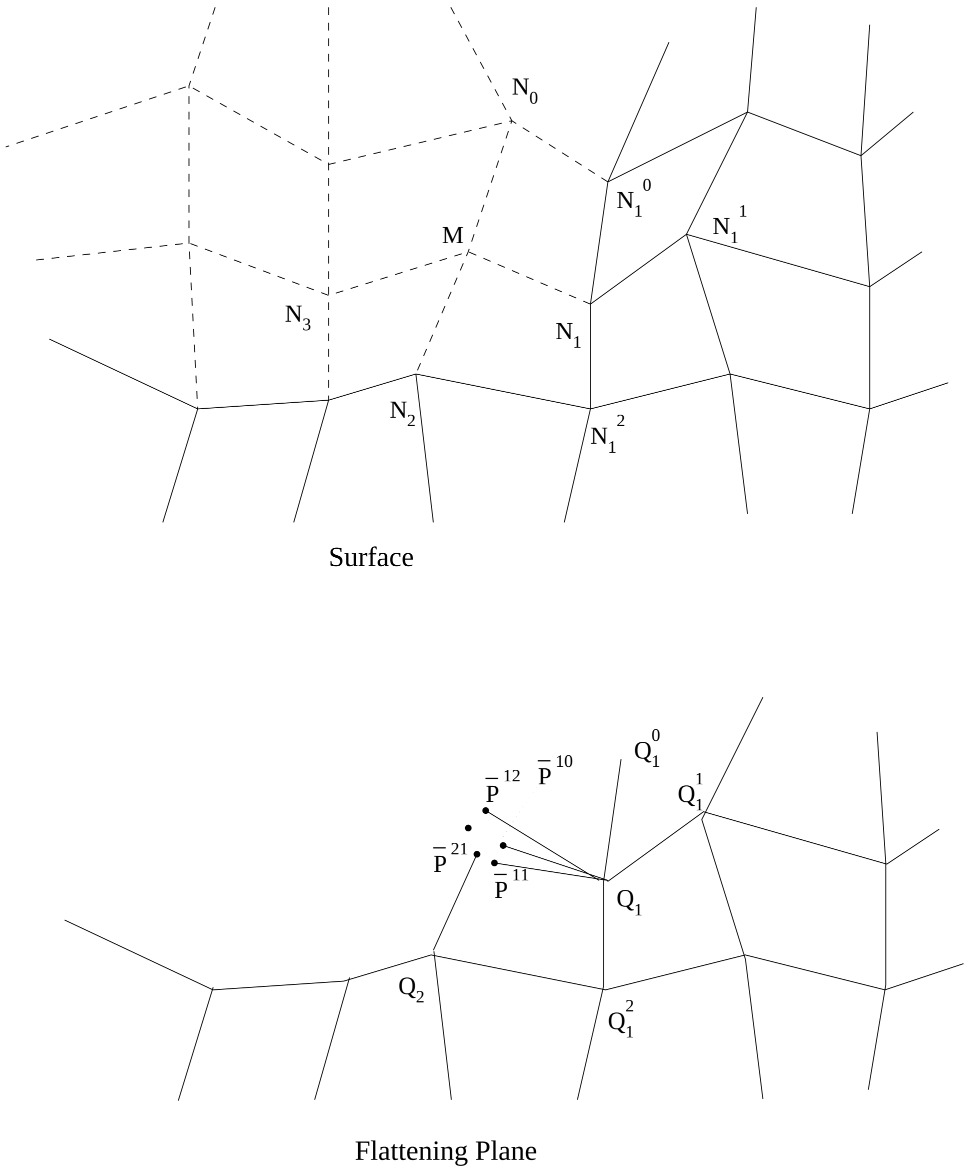

Figure 4.5: Flattening surfaces using the preservation operator $\mathcal{P}$

points $\{P_k\}$ sharing the plane with a point P, one says that the *dispersion of the cluster around* P is given by

$$\mathrm{dsp}(P) = \frac{1}{n}\sum_{k=1}^{n} \|P - P_k\|^2.$$

The value of $\mathrm{dsp}(P)$ is minimized when P is at the centroid of the cluster, in which case $\mathrm{dsp}(P)$ is called the *dispersion of the cluster*. The dispersion of a cluster around their centroid corresponds to the statistical notion of *variance*, a measure that determines how widely a set of samples deviates from its mean value. With $\{M_i\} \subset C'_S$ and $\{P_i\} \subset F$ as above, the metric $\mathcal{D}_\mathcal{D}$ is given as the cumulative dispersion of the candidate points $\{\bar{P}^{ij}\}$ defining each point in the projection:

$$\mathcal{D}_\mathcal{D}(C'_S) = \sum_i \mathrm{dsp}(P_i).$$

The denser the candidate clusters are, the lesser the value of their dispersion, and the better the flattening achieved of S.

4.4 Surface Area Parameterization

A different approach to texture mapping is more closely associated with the goal stated in Section 4.1. The general idea behind this method is that given a surface area parameterization of the target object, the triangulation of the sampled points will yield surface elements (*i.e.*, triangles) of constant size. Under an ideal parameterization, all triangles will have the same area and aspect ratio. A *cookie cutter* approach can thus be applied to the texture, slicing it up uniformly into triangles of the desired size which can then be placed directly on the triangulated object. The texture coordinates arising from this uniform triangulation of the texture are assigned to the surface-area parameterized samples of the surface.

The analytic formulation of a surface area parameterization is quite messy, which suggests that obtaining one in practice is a grueling task. A wide variety of techniques for mesh generation have been proposed in the field of finite-element analysis. One such method is proposed in [Tur91], suggests using the mesh induced by the *Voronoi regions*[2] of a set of uniformly distributed surface points. Because the scattered points are evenly distributed, the cells will all be of approximately equal size. Unfortunately, they are not all formed necessarily in the same polygonal shape, and thus complicate the task of cutting the texture. Another drawback of this method is its computational cost: The uniform distribution of the point set is obtained through a relaxation process which must be completed prior to the computation of the Voronoi regions. A possible method of overcoming the first difficulty is by using the dual graph of the Voronoi diagram, called the *Delaunay triangulation*, in which all regions are triangular in shape. The computational cost, however, remains a problem.

The method proposed below is based on a relaxation method that perturbs the parameterization using a gradient descent optimization process. An initial estimate is produced by a uniform knot distribution, which is then coerced into a surface area parameterization. To motivate this approach, its one-dimensional analogue – a method for obtaining an arc-length parameterization of space curves – is presented first.

4.4.1 Achieving Arc-Length Parameterization

The parameterization of a curve can be given by an ordered set of parameter values called a *knot vector*. The terms in this vector correspond to the points on the curve; each point is produced by its corresponding parameter (*knot*) value. The number of sample points on the curve – its *density* – depends of course on the number of knots. When the knot vector represents an arclength parameterization, the distance between each subsequent pair of these curve points is constant.

The desired knot vector can be found by a process of iterated improvement of an

[2]The *Voronoi regions* generated by a given set of points $\{\mathbf{x}_i\}$ is a set of regions $\{R_i\}$, where each region R_i is composed of all the points that are closer to the generator $\mathbf{x}_i$ than to any other generating point. In set theoretic notation,

$$R_i := \{\mathbf{x} \quad | \quad d(\mathbf{x}, \mathbf{x}_i) \le d(\mathbf{x}, \mathbf{x}_j), \forall j \neq i\}.$$

initial guess, which ought to be easily attainable. A reasonable starting point could be a uniform knot distribution, *i.e.*, a vector whose parameters are evenly spaced. The basic technique is that with each iteration, the endpoints of the "worst" segment are perturbed slightly so that its length more closely matches that of the mean. The endpoints of the actual curve – the first and last knots – should not be touched so that the curve segment remains within its bounds. Obviously, perturbing the endpoints of one arc disrupts the length of its neighbors, which is why an iterative process is employed. Iteration is halted when a sufficiently good approximation is reached. The reasoning behind this approach is that the naive starting point and the small steps will ensure the avoidance of strange topological and geometric anomalies.

Let $\mathbf{c}$ be the space curve, and $\mathbf{x} = (x_0, \ldots, x_n)$ be a knot vector. An approximation to the curve is given by the set of $n+1$ sample points $\mathbf{c}[\mathbf{x}] = (\mathbf{c}_0, \ldots, \mathbf{c}_n)$ where $\mathbf{c}_i = \mathbf{c}(x_i)$. Given the initial guess $\mathbf{x}$, a new knot vector $\mathbf{x}' = (x'_0, \ldots, x'_n)$ is sought, that satisfies

$$\delta\mathbf{c}'_i \equiv \|\mathbf{c}'_i - \mathbf{c}'_{i-1}\| \approx \chi, \qquad i = 1, \ldots, n,$$

where $\chi = \frac{1}{n}\sum \|\mathbf{c}_i - \mathbf{c}_{i-1}\|$ is the average chord length, $\mathbf{c}'_i = \mathbf{c}(x'_i)$, $x'_0 = x_0$, and $x'_n = x_n$. The *squared discrepancy* $D(\mathbf{c}_i)$ of the chord $\overline{\mathbf{c}_{i-1}\mathbf{c}_i}$ is defined to be its squared "distance" from the mean chord length: $D(\mathbf{c}_i) = (\delta\mathbf{c}_i - \chi)^2$. Using it, the goal can be stated as solving the following optimization problem: Beginning with the naive estimate $\mathbf{x}$, find an improved parameterization $\mathbf{x}'$ that minimizes the sample variance $\sigma^2_{\mathbf{c}}(\mathbf{x}') = \sum_i D(\mathbf{c}'_i)$.

Algorithm (Arc-Length Relaxation):

1. *Let γ and ε be two positive constants.*

2. *Do*

   Choose $\mathbf{c}'_j$ such that $D(\mathbf{c}'_j) = \max_i\{D(\mathbf{c}'_i)\}$

   Perturb x'_{j-1} and x'_j by γ within (x'_{j-2}, x'_{j+1}), without moving x'_0 and x'_n

   until $\sigma^2_{\mathbf{c}}(\mathbf{x}') \leq \varepsilon$.

The perturbation is applied by adding or subtracting γ from the parameter, depending on the length of $\delta \mathbf{c}'_j$ relative to χ.

The constants γ and ε must be judiciously chosen in order to ensure the algorithm halts. The permissible variance ε relies heavily on the perturbation factor γ, as convergence of the parameterization to within ε of the goal may be impossible if γ is not sufficiently small.

The above algorithm was provided as a means to motivate the algorithm for surfaces. In itself, however, it is extremely inefficient as the desired knot spacing can be explicitly computed by estimating the arclength first and then successively solving for $\mathbf{c}_1, \ldots, \mathbf{c}_{n-1}$.

4.4.2 Achieving-Surface Area Parameterization

A sampled surface area consists of a collection of sample points which are triangulated during the rendering process. This triangulation produces a list of arcs connecting the sample points, the configuration of which approximates the surface. A direct extension of the method used for curves makes use of these arcs to obtain a surface-area parameterization. The list of arcs may be viewed as a list of triangles formed by them. In contrast to method for curves, however, where the arc lengths provided the parameter constraints, we now use the *area* and *aspect ratio* of these triangles to impose restrictions.

Once again, the parameterization is specified by a list of parameter values which now have two dimensions rather than one. It is therefore convenient to hold these values in an array, which we appropriately call the *knot array.* Following the same methodology, the process is begun by guessing an approximation which is subsequently improved by an iterative process of relaxation. A reasonable and cheap initial guess for the knot array is a uniform distribution of knots along the isoparametric contours.

Let $\mathbf{S}(\mathbf{u})$ be the parameterized surface at hand, and let $\mathbf{X} = (x_{ij})_{\substack{0\ldots m \\ 0\ldots n}}$, $x_{ij} \in [0,1] \times [0,1]$, be the knot array. An approximation of the surface is given by the set of $(m+1)\times(n+1)$ points $\mathbf{S}[\mathbf{X}] = (s_{ij})_{\substack{0\ldots m \\ 0\ldots n}}$ where $s_{ij} = \mathbf{S}(x_{ij})$. The set $\mathbf{S}[\mathbf{X}]$ also induces a triangulation $\{T_{ij}\}$ of the surface, wherein the triangle T_{ij} is formed by the three points $s_{i,j}, s_{i+1,j}$, and $s_{i+1,j+1}$. All the sample points but s_{0n}, which is insignificant as will be seen below, are involved in this triangulation.

The relaxation is carried out as an optimization process of an error function using a gradient descent search technique. To facilitate this, the metric m is defined on the triangles T_{ij} given by $m(T_{ij}) = \mathcal{A}(T_{ij})^2 + \bar{\mathcal{A}}(T_{ij})^2$, where $\mathcal{A}(T_{ij})$ is the signed area of T_{ij} and $\bar{\mathcal{A}}(T_{ij})$ is the triangle's aspect ratio. Finally, let $\chi = \frac{1}{nm}\sum_{i,j} m(T_{ij})$ be the average metric on $\mathbf{S}[\mathbf{X}]$.

A new knot array $\mathbf{X}' = (x'_{ij})$ that satisfies

$$m(T'_{ij}) \approx \chi \qquad \begin{array}{l} i = 0, \ldots, n-1 \\ j = 0, \ldots, m-1 \end{array} \tag{4.1}$$

is to be computed, where T'_{ij} is the triangle formed by $s'_{i,j}, s'_{i+1,j}$, and $s'_{i+1,j+1}$. The only requirement is that under the new parameterization, the patch will stay within the confines produced by the old parameterization. In other words, the boundary curves must not be changed as a result of the new knot array. Since the point produced by x_{0n} is on such a boundary curve its new value is prescribed, and therefore its neglection by the optimization process is immaterial. Once again, the *squared discrepancy* $D(T'_{ij})$ of a triangle T'_{ij} is defined according to the distance of its metric from the mean: $D(T'_{ij}) = (m(T'_{ij}) - \chi)^2$. The sought-after knot array $\mathbf{X}'$ must minimize the variance $\sigma_S^2(\mathbf{X}') = \sum_{i,j} D(T'_{ij})$.

With the surface notation in place, the optimization algorithm is virtually identical to the one described for curves.

Algorithm (Surface-Area Relaxation):

1. *Let γ and ε be two positive constants.*

2. *Do*

   Choose T'_{ij} such that $D(T'_{ij}) = \max_{ij}\{D(T'_{ij})\}$

Perturb the parameters $x'_{i,j}, x'_{i+1,j}, x'_{i+1,j+1}$ *that do not lie on the boundary by* γ *in some parametric direction*

until $\sigma^2(\mathbf{X}') \leq \varepsilon$.

The perturbation is applied by adding or subtracting γ in a parametric direction that depends on the discrepancy of T_{ij}. Of course the parameters γ and ε must be well chosen in order for the algorithm to terminate.

Measuring the quality of the triangulation on the basis of area and aspect ratio is equivalent to measuring it on the basis of area and side length. As a result, if the aspect ratio is properly chosen then the arcs will all be of equal length. Note that the relaxation process takes into account only the triangles T'_{ij} formed by the points $s'_{i,j}, s'_{i+1,j}, s'_{i+1,j+1}$, but not the triangles $\hat{T}'_{ij}$ formed by $s_{i,j+1}, s_{i+1,j}, s_{i+1,j+1}$. However when the side lengths are equal, T'_{ij} and $\hat{T}'_{ij}$ have identical shape, and hence the uniformly cut texture pieces will fit directly.

As an alternative to gradient descent, other optimization techniques such as random walks or simulated annealing may be used. This is of particular interest in the case of surfaces for which the search space is sufficiently large to make the stochastic techniques cost effective.

4.5 Remarks

Before concluding this chapter, it must be restated that the problem of texture mapping involves more than just "fitting a square peg in a round hole". To date, the general approach has been focused for the most part on finding a global solution, in which the object to be textured is considered as a whole. Because of the mathematical impossibility of texturing non-developable surfaces without distortion, the inclusion of distortion has been accepted as a fact of life. Objects are individually textured in such ways that the distortion is tucked out of sight. This works for surfaces of relatively small curvature, but when the curvature becomes too large the distortion may be impossible to hide. Several authors (*e.g.*, [Hos84]) have proposed methods to detect these areas and remove them from the surface.

A different approach is to discard the global solution in favor of a collection of local ones ([MYV93]). The surface is split into several independent regions, each of which is textured on its own. Every region then assumes a unique texture and a unique map called a *chart* as derived from the differential geometric notion. Each chart is continuous in its interior, but not necessarily along the boundaries. The collection of charts – the *atlas* – provides a cover of the surface.

Chapter 5

Piecewise Surfaces

When the surfaces modeled are large or complex, the task of controlling their shape becomes rather daunting. To afford the designer adequate control over the fine details, the modeler must provide a large number of control points. Were the geometric model to be constructed from one piece, the multitude of control points would result in a surface of very high degree. In addition to the numerical instability involved in the computation of high degree polynomials on a finite precision machine, the time complexity would be unduly large. Since the purpose of having multiple control points is to afford local shape control, it may safe to assume that the effects of changes will have limited extent. In this case, the recomputation of the entire surface every time a small change is made is rather redundant. For these reasons, surfaces are generally constructed piecewise from collections of patches of a relatively low degree.

The development of Bézier triangles in Chapter 3, and the subsequent development of texture mapping in Chapter 4, were focused primarily on formulations for individual patches. As such, some of the more subtle issues that arise when large surfaces are constructed from multiple patches stitched together were not addressed in those chapter. These will be discussed now. Some of the inherent problems encountered when creating an overall smooth surface from one that has been paved with triangular patches are presented first, and a solution is provided. Next, the extension of the texturing techniques to piecewise surfaces is described.

5.1 Patch-Patch Surface Continuity Constraints

As was mentioned in Section 3.9, strict continuity conditions must be met along all the boundaries between patches to ensure the smooth appearance of the entire surface. To obtain a cohesive surface that is devoid of holes and cracks, one must at the very least insist on C^0 continuity. To ensure the patches meet smoothly, one might wish to demand continuity of the first derivative. Indeed, a sufficient condition for C^1 continuity was derived in Section 3.9.2, by which the control points parallel to the boundary on one patch were positioned appropriately. This, however, is difficult to enforce and often imposes very severe restraints on the shape of the surface. Occasionally, the surface topology itself may cause C^1 continuity to be unattainable [Her87].

The primary concern of the modeler is with visual appearance. As such, the mathematical precision may often be relaxed somewhat and still produce convincing results. An alternative approach to derivative continuity is to require that tangent planes be continuous along the boundaries. Such surfaces are known as G^1 continuous and have been described in Section 3.9.3. Additional methods for obtaining G^1 continuity are described in [Far93, Pip87, Her87].

Before elaborating on the continuity issues, an additional bit of differential geometry is introduced. Recall that infinitely many curves on a surface S may pass through a point $\mathbf{p} \in S$. The curvature at $\mathbf{p}$ of each of these curves is different, but the curvature of the actual surface at that point depends only on S. A quadratic form on the surface is thus defined to characterize the curvature properties at every point. The *second fundamental form* of S at $\mathbf{p}$ is denoted by $\mathrm{II}_{\mathbf{p}}$ and is defined by [dC76]

$$\mathrm{II}_{\mathbf{p}}(\mathbf{v}) = < d\mathbf{n}_{\mathbf{p}}(\mathbf{v}), \mathbf{v} >,$$

where $\mathbf{v}$ is any vector in the tangent plane $\mathbf{T}_{\mathbf{p}}(S)$ of S at $\mathbf{p}$, $< \cdot, \cdot >$ denotes the natural inner (dot) product on $\mathbf{T}_{\mathbf{p}}(S)$, and $d\mathbf{n}_{\mathbf{p}}(\mathbf{v})$ measures the rate of change of the surface normal at $\mathbf{p}$ in the direction $\mathbf{v}$.

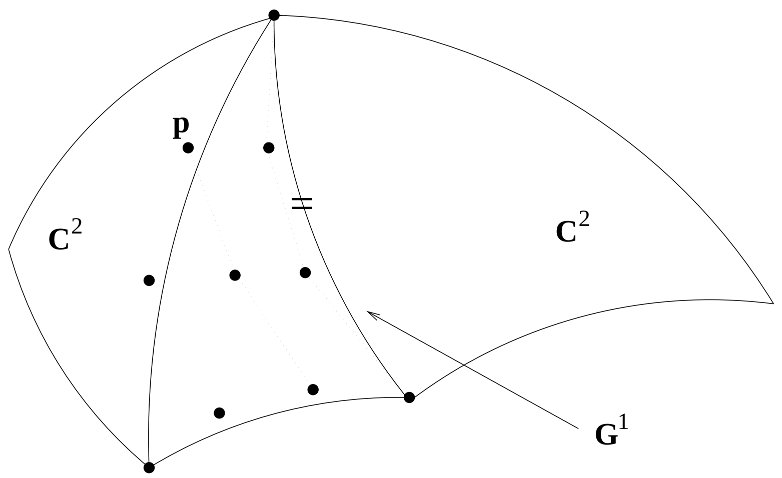

Figure 5.1: The Vertex Consistency Problem: By enforcing G^1 continuity along the boundary marked by '=', the control point **p** becomes over-constrained

5.1.1 Vertex Consistency

Unfortunately, G^1 continuity is not always sufficient to provide the overall smoothness of a piecewise surface. Specifically, it offers no respite from what is known as the *vertex consistency problem*. Consider, for example, three abutting surface patches which share a common vertex. The composite surface is made smooth by first insisting on the fairness of the individual patches, which can be achieved by demanding that each be C^2 continuous internally. Smoothness of the seams is gained by requiring tangent plane continuity along the patch boundaries. As previously described, making the middle patch G^1 continuous with one of its neighbors prescribes the position of all the control points along the two diagonals parallel to the border (see Figure 5.1). One of these points, **p**, will always lie on the boundary with the other patch, and hence cannot be moved. It seems, therefore, that even C^0 continuity with the second neighbor is unobtainable, let alone G^1 continuity.

More generally, the problem of continuity may be formulated in the following way. Given two patches that share a common vertex, find a third triangular patch that abuts the two, and that, together with them, forms an overall G^1 surface. Clearly, the new patch must share the common vertex. Its two remaining corners are given by those on the corresponding neighbors. The required continuity condition prescribes the normals at the three vertices. Therefore the problem can be stated even more generally as follows: Find a triangular Bézier patch that interpolates three vertices whose position and normal vectors are given. This is known as finding the *Hermite interpolant* [Far86].

This problem is not trivial, as a direct solution does not always exist. For example, van Wijk [vW86] has shown that when an even number (greater than four) of tensor-product patches meet at one vertex, tangent plane continuity may not be achievable. The result has later been strengthened by Peters [Pet91] who provided a sufficient condition for the constraints to be satisfied for arbitrary patches. The condition requires that all boundary curves which meet at a common vertex be compatible with some second fundamental form at the vertex.

5.1.2 The Clough-Tocher Interpolant

A number of solutions to the vertex consistency problem are given in [Far86, Her87]; a comparison of various methods is provided by [MLL+92]. The interpolation scheme presented by Piper in [Pip87], a variant of the Clough-Tocher interpolant used in [Far86], will be described below. The basic idea behind this method is to first generate a patch that interpolates the data, and then split it into three polynomial subpatches in order to satisfy the strict continuity constraints with neighbors. Schemes that follow this general approach are called *split-domain schemes*. It has been shown in [Pet91] that split domain schemes sufficiently decouple the conflicting constraints, and produce viable solutions.

Piper's method proceeds by first constructing a cubic patch whose vertices $\mathbf{b}_{003}$, $\mathbf{b}_{030}$, and $\mathbf{b}_{300}$ are the given positions. The point $\mathbf{b}_{210}$ is produced by projecting the point $\mathbf{b}_{030}$ onto the tangent plane at $\mathbf{b}_{300}$, and adding a scaled down version of the vector joining the latter vertex and the projected point to $\mathbf{b}_{300}$ (see Figure 5.2). The remaining boundary points, $\mathbf{b}_{120}$, $\mathbf{b}_{021}$, $\mathbf{b}_{012}$, $\mathbf{b}_{102}$, and $\mathbf{b}_{201}$ are produced in an analogous fashion. For each boundary point, the scaling factor is chosen such that the surface normals and positions of the vertices are interpolated properly.

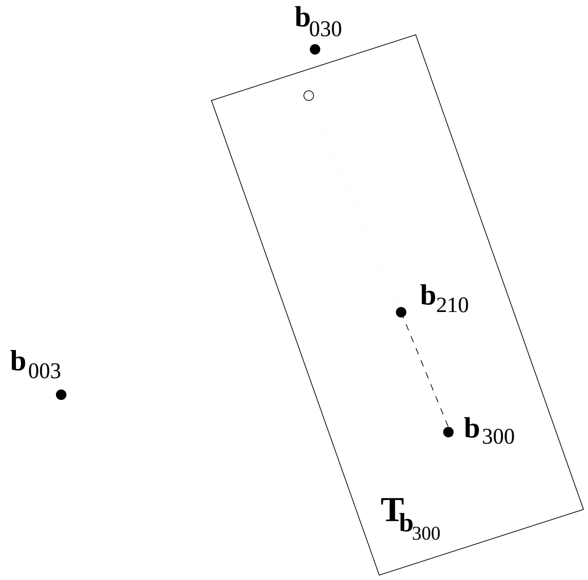

Figure 5.2: Construction of the Clough-Tocher interpolant

The final control point, $\mathbf{b}_{111}$, is given by the rather bizarre formula [Far86]

$$\mathbf{b}_{111} = \frac{\sum_{i=0}^{3} \mathbf{p}_i/m_i}{\sum_{i=0}^{3} 1/m_i}, \tag{5.1}$$

where

$$\begin{aligned} \mathbf{p}_1 &= (\mathbf{b}_{120} + \mathbf{b}_{102})/2, \\ \mathbf{p}_2 &= (\mathbf{b}_{012} + \mathbf{b}_{210})/2, \qquad \text{and} \\ \mathbf{p}_3 &= (\mathbf{b}_{021} + \mathbf{b}_{201})/2. \end{aligned}$$

The quantity m_i measures the deviation of the boundary curve i from a straight line, where $m_i = 0$ indicates a straight line. The ostensibly odd choice of $\mathbf{b}_{111}$ is used to curb the undulating appearance of the patch. It forces a ruled surface to be produced when one of the boundary curves is a straight line.

The patch thus produced interpolates the given data and is internally smooth, but does not form a clean fit with its neighbors. This problem is overcome by splitting the patch along its centroid, and smoothing the resulting terrain. The boundaries between the three subpatches are referred to as the *internal* boundaries; the others

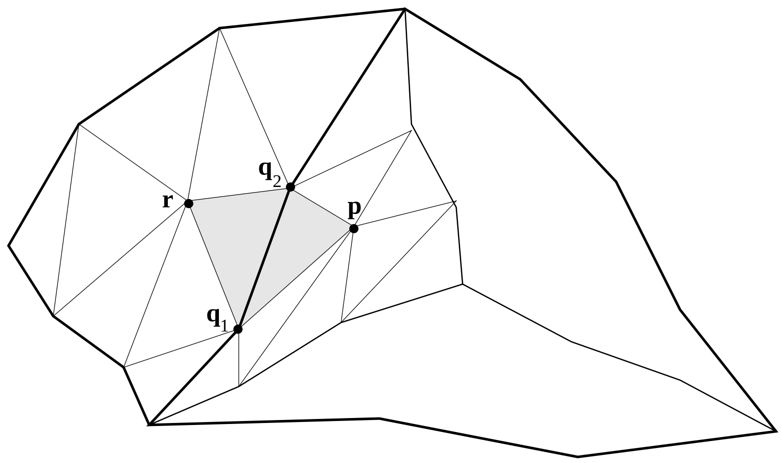

Figure 5.3: Splitting the Clough-Tocher interpolant: The control point **p** is projected onto the plane spanned by the control points $\mathbf{q}_1$, $\mathbf{q}_2$, **r** of the neighboring patch.

– formerly the boundaries of the unified patch – are referred to as the *external* ones. The collection of patches is made smooth by enforcing continuity constraints between the subpatches and their exterior neighbors. This is achieved as follows. First, the central control point of each sub-patch is adjusted, if necessary, until it becomes coplanar with the triangle of control points on the patch opposite the boundary (the shaded regions in Figure 5.3). Next, the control points along the interior boundaries are modified to maintain smoothness between the sub-patches. The trio is now degree elevated, and the control points of the quartic sub-patches adjusted so as to satisfy the continuity conditions with the external neighbors. Finally, the control points along the interior boundaries are adjusted so as to join with C^1 continuity.

There is much freedom in the choice of the initial central control point $\mathbf{b}_{111}$. When the method fails to produce visually satisfying results, an alternate and slightly weaker form of continuity may be used. The excess freedom is reduced by requiring additional information at the patch vertices, namely the second fundamental form. The class of *approximate G^1 continuity* was introduced by DeRose and Mann in [DM92] and offers an enhanced form of C^0 continuity which is nearly G^1. Roughly speaking, the disparity of the continuity of the surface normals from G^1 is bounded by a parameter ε, in the sense that the angle between two surface normals is at most ε. Such surfaces are referred to as being of class ε-G^1.

A final word on the use of the Clough-Tocher interpolant: Using the method just described, a smooth bridge can be formed between two patches that share a common vertex. In order to create this bridge, all the designer must do is specify the vertices forming the corners. The normal information is then conferred by the two existing patches. The method can also be used to create a *new* patch that abuts another that already exists. Once the designer has selected the common boundary, all that is required are the location and the normal of the third vertex. Once selected, the new patch can be immediately fabricated. This method thus provides a simple means of setting up a patch from scratch. Rather than specify the complete control net, the designer need only identify the location and normal information at the three corners. The control net is then created from this data and the design progresses as before. One detail is missing from this creation process: The task of specifying the locations and the normal directions of new vertices. This issue is a detail of the User-Interface design, and will be addressed in the next chapter.

5.2 Mapping Textures on Composite Surfaces

The construction of surfaces using piecewise collections of surface patches requires special care on boundary curves to ensure smooth visual interpolation. Texture cohesion must also be maintained along these curves in order to keep the boundaries from showing through the composite surface.

5.2.1 Concentric Ring Projection

Under the surface flattening method of texture mapping (Section 4.3.3), cohesion may be attained by using the boundary of each patch as the initial flattening curve. The boundary will be projected with no distortion, ensuring a visually smooth transition of textures from one patch to the next. As the flattening scheme relies heavily on surface normals, it is necessary that the tangent planes of adjacent patches coincide along the patch boundaries. At the very least, G^1 continuity between adjacent patches is required. This is a moot point, however, because when adjacent patches are not

G^1 continuous, the perforation is clearly visible even when no texturing is applied. Complete texture continuity is therefore not necessary in this case.

Once the patch boundary has been faithfully flattened, the patch interior must be developed in a consistent fashion. In lieu of using the parallel curves described in Section 4.3.3, concentric rings of isoparametric curves are used in this case. Because the direction of development flows inwardly from the patch boundary, each point on an internal ring is given as the centroid of all the candidates formed by the points directly behind it. The number of contributing candidates depends on the parameterization of the patch and on the location of the point. For example, points on a corner[1] of a ring are defined by candidates formed by both isoparametric curves, and hence have more contributors than points defined by candidates formed from one curve only. Both distortion metrics $\mathcal{D}_\mathcal{A}$ and $\mathcal{D}_\mathcal{D}$ may be used to determine when an acceptable distortion level has been surpassed. Once this level has been exceeded, a discontinuity is introduced to the texture by halting the development process and faithfully flattening the next concentric ring. Development is then resumed as before until the center of the patch is reached and the entire patch has been flattened.

Unfortunately, concentric ring flattening suffers from a severe drawback. The method relies on faithfully flattening a closed loop, a task that is generally impossible to perform with precision. The flattening procedure works well for all but the last point on the ring, for there is no guarantee that the loop will be properly closed. When the first point is faithfully projected from the last, it may not coincide with the location assumed for the first point when the loop was begun (Figure 5.4). The arc between the first and the last points is therefore generally incorrect and leads to a local distortion. Once the boundary has been distorted, the distortion is "bled" deeper into the texture by the $\mathcal{P}$ operator as it averages points on the faithfully flattened ring. This leads to a streaking effect. This distortion occurs at every point of discontinuity when a new concentric ring is faithfully flattened. The problem is compounded by the fact that a patch whose boundary is already distorted by this

[1] A *corner* of an isoparametric ring is a point that simultaneously lies on two of the isoparametric curves comprising the ring.

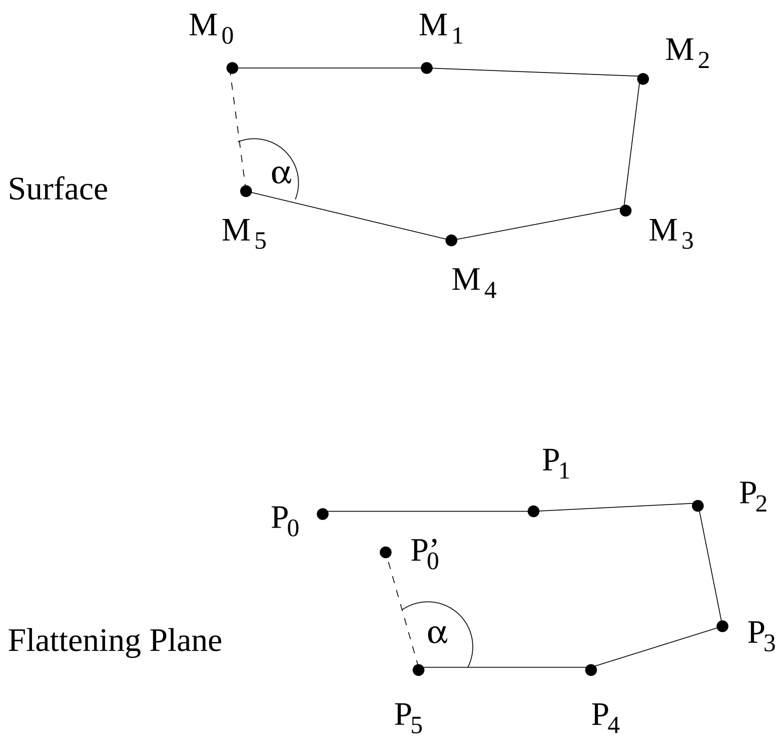

Figure 5.4: Ring flattening: When closing the ring, there is no guarantee that $\mathbf{P}_0$ and that $\mathbf{P}'_0$ will coincide

may inherit additional distortion from that on the boundary of its neighbors when continuity conditions are enforced.

5.2.2 Piecewise Surface-Area Texturing

The texture between adjacent patches may be made continuous by forcing the patches to share the texture coordinates on the common patch corners. This is akin to draping a sheet of cloth over the composite texture, and pegging it down at the corners of every patch. Once the pegs have been set, the material is cut in straight lines between the pegs. Each piece is then assigned to its corresponding patch. This method works well when the boundary curves of the patches are not excessively wavy, and may thus be wrapped with the straight-edged fabric.

When the patches have all been assigned an appropriate swath, barycentric combinations of the corners may be used to assign texture coordinates to sampled points along the boundary. This will ensure a distribution of the texture that befits the sampling rate of the patch. A correspondence between the parametric directions on the patch and those on the texture will no longer be necessary. Having assigned texture coordinates to the patch boundaries, the interior may be cut up using barycentric combinations of the boundary points thus obtaining the required uniform texture meshing. The surface area parameterization of the patch interior may now commence exactly as described in Section 4.4.2.

5.3 Summary So Far

At this point, the process by which individual patches can be created, manipulated, and textured is known. It is also clear how a collection of individual patches can be pieced together in a consistent fashion to form a large and cohesive surface. These theoretical results can now be gathered and put to use in the implementation of a textured-surface modeler. This is the subject of the next chapter.

Chapter 6

Implementation

Having developed the theory that facilitates the creation and the manipulation of general surfaces, several implementation details remain to be resolved. These essentially fall into two categories: the design of a user interface and the resolution of technical issues.

The design of the user interface deserves special consideration as it acts as the bridge between the internal models and the "outside world." As the development of this tool has been focused on geometric intuition, the work would be all for naught if the front-end was overly complicated. All too often in the past, the developers of software have failed to heed this point. The key to the widespread use of computer graphics lies not only in the technical advances of the field, as has been pointed out by [HH90], but is significantly dependent on the interactive techniques that make these concepts readily accessible.

The user interface must at once be simple to intuit and powerful in functionality. The concept of *intuition* itself is somewhat murky, as what is obvious to one person may be obscure to another. Although somewhat rough, the user interface described below tries to find some common ground.

This chapter will discuss the user interface to the model and the technical issues involved in its implementation. Some images produced by it will be exhibited.

6.1 The User Interface

A non-trivial challenge in the design of a modeler for 3D graphics is the development of an intuitive input device that can specify points in three-space. Various three-dimensional mechanical apparatuses have been devised to overcome this problem, but none have gained mainstream popularity. The majority of pointing devices are still two dimensional, and typically fall into one of two standard categories: the mouse/trackball, and the tablet. Thus one of the issues a usable human interface must address is the problem of mapping the various flat input surfaces into three-dimensional virtual space.

6.1.1 Manipulation of Scenes

One solution to this mapping problem is the use of a *virtual sphere* [CMS88]. Using a virtual sphere, the trackball becomes a hypothetical crystal ball containing the scene, which is rotated by an angle equivalent to the physical rotation of trackball. A similar approach has been adopted by our implementation, with the trackball being replaced by a mouse. Dragging the mouse with its rightmost button (Mouse-3) depressed produces a virtual-sphere rotation. Similarly, dragging the mouse with the middle button (Mouse-2) depressed affords a *zooming* effect.

6.1.2 Selection

There are two basic modes of selection, for which the leftmost mouse button (Mouse-1) is used. Using the editing mode, existing objects are selected and manipulated. At all times the modeler maintains a patch of highest priority, called the *hot patch*, which is the one the user is allowed to modify. Each patch has a specific hot *point* which is the control point currently under modification. By dragging Mouse-1, the hot *point* on the hot *patch* is moved a corresponding amount on the virtual plane parallel to the screen on which the point lies. Selection of the hot point is performed by double clicking the mouse, whence the object under the pointer which is "closest" to the front of the screen is returned. When another control point on the hot-patch lies in the vicinity of the mouse pointer, it is selected as the next hot-point. If none is found, then the *patch* under the mouse pointer becomes the next hot-patch.

The second selection model, the drawing mode, is used for laying down the control points when specifying a new patch. Using the results of Section 5.1.2, a patch may be specified by its three corner control points and their corresponding normals. Thus, a new patch may be described by the location of its corners, with estimated normal vectors. Under this mode, depressing Mouse-1 selects a corner point for a new patch. Selection works in a similar way as before, with the exception that, when no existing control point is in the vicinity of the mouse click, the screen point is backward projected through the rendering pipeline to produce a point in space.

6.1.3 User Control

Control of the modeler is provided through a control panel (Figure 6.1). Through it, the user may select the illumination model, the texturing method, and the texture to be applied. It also provides a direct method of specifying the coordinates of the current hot point. The selection and viewing modes are selected through this panel. A slider offers direct control over the permissible distortion level in the surface flattening method of texture mapping.

An additional widget called the *compass* also appears on the bottom left corner of the panel. Its task is similar to that of a physical compass, and assists the user in determining the orientation of the scene. The compass is composed of three arrows. Each arrow represents a coordinate axis and points in the (respective) positive direction. Each arrow is uniquely colored and thus provides a visual cue to the orientation of the axis.

In addition to the control panel, the following keyboard operations are available and are applied to the active patch.

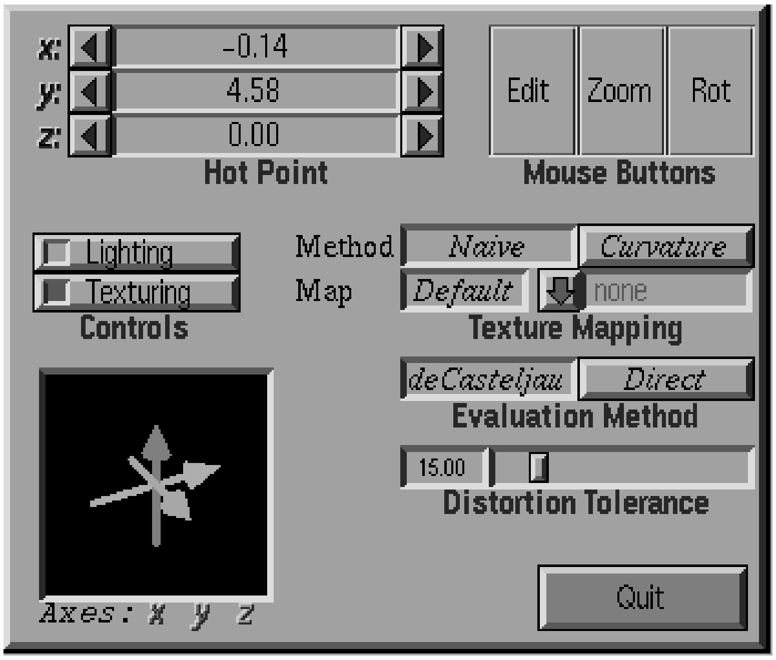

Figure 6.1: The control panel

Key	Description
0	Impose C^0 continuity between the patch and the one previously active
1	Impose C^1 continuity between the patch and the one previously active
d	Decrease the density of the surface meshing
i	Increase the density of the surface meshing
r	Raise the degree of the patch
s	Subdivide the patch at its centroid

6.2 Technical Issues

The implementation code for the modeler has been written in C++ using the OpenGL graphics libraries, and has been designed to run on an Silicon Graphics workstation, running under IRIX 5.3. As described above, the technical issues can be separated into three parts: those involved in the surface construction, those involved in the texturing process, and those involved in the user interface. Details of these are discussed below.

6.2.1 Model Construction

The basic component of the piecewise surface is the triangular patch. Correspondingly, the data structure used in the construction of such a surface is an object representing a Bézier triangle. Each individual patch is completely described by an instance of the class, which contains all the information necessary to resurrect itself. In addition to the control net, each object contains the degree of the patch, its sampling rate, and a list of neighboring patches. The structure also contains the mesh of sampled surface points, thus eliminating the need to recompute the surface every time the patch is rendered. The surface is computed only when the control mesh or the surface representation have been changed. The former may occur as a result of direct control-point manipulation by the user, or as a result of continuity enforcement. The latter can occur only when the sampling resolution has been altered.

The Control Mesh

The triangular control mesh is implemented using a one-dimensional array of control-points, of length $(n+1)(n+2)/2$, where n is the degree of the patch. Points are represented as lists of their three coordinate values. The multi-index (i, j, k) which gives the row and column position of the point in the mesh (as described in Section 3.3) is replaced by a single index q. Since, in most cases, we are not concerned with the specific indices of the triple, a simple sequential enumeration of the control points suffices. The conversion between the single index q and the multi-index (i, j, k) is tedious and time consuming, so when explicit coordinates are required the rows and columns of the mesh are independently enumerated (using nested `for` loops).

Surface Sampling

As mentioned above, the patch surface is computed only when the control mesh has been modified. The unit triangle is used as a common domain, so that all patches are defined over the identical parameter space. The sampling of the surface is initially performed over a uniform knot distribution. The resolution of sample points – the *sampling rate* – is controlled by a patch-specific parameter which determines the number of "rows" of points that will be sampled. Because of the initial uniformity of samples, a resolution factor of m specifies that $\binom{m+2}{2}$ points be sampled. By default $m = 12$, and hence patches start off with $\binom{14}{2} = 91$ sample points. The resolution parameter may be interactively altered by the user to either *thin-out* or *thicken* the surface meshing.

The method of sampling is also controlled by the user. Patches may be generated either by using the de Casteljau algorithm (Section 3.3) or by direct evaluation using Horner's rule (Section 3.11). When the former is selected, the barycentric coordinates of each point are blossomed using the de Casteljau algorithm to produce the corresponding surface point. Because the de Casteljau recursion is always one level deep, the control points of one level are sufficient to produce those of the next. To generate the control mesh for level r we use, and then over-write, the mesh of level $r-1$. Since the Bézier points must be preserved throughout the blossoming process, a temporary work-space array is dynamically created at the time of generation, into which the level-0 control mesh (*i.e.*, the Bézier net) is copied each time blossoming is invoked.

When the direct method is applied, no work space is necessary since each sampled point is computed directly using convex combinations of the control points. Because the set of trinomial coefficients is constant per patch, it is computed in advance, once for each patch, and stored in the object. The reason separate copies are kept is that the coefficients depend upon the degree of the patch, which is not constant. The sampled points are then computed using Horner's rule as described in Chapter 3.

The collection of surface points thus obtained is stored in a structure identical to the one used for the control net, albeit of a higher degree.

Normal Computation

As described in Section 3.6.2, the normal at every point can be computed from the three level-$(n-1)$ points generating the surface point in the de Casteljau algorithm. Thus, when computing the surface by means of blossoming, the evaluation of every

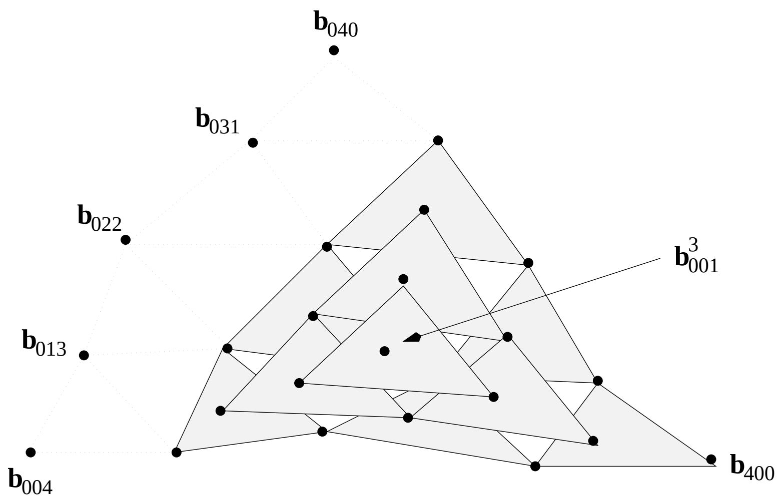

Figure 6.2: Modified direct evaluation of the de Casteljau points

point produces the normal to the surface as well, at the added cost of computing a vector (cross) product.

Direct evaluation does not enjoy the same benefit. To overcome this shortfall, the method of direct evaluation is modified slightly. Instead of evaluating the *surface point*, Horner's rule is used thrice to compute the three level-$(n-1)$ *de Casteljau points*. These, then, are used to produce both the surface point and the normal at the point. For a patch of degree n, the de Casteljau point $\mathbf{b}_{001}^{n-1}$ is computed by evaluating the degree $n-1$ patch whose control net is identical to that of the original with the diagonal $\mathbf{i}^0$ removed (see Figure 6.2). The remaining de Casteljau points, $\mathbf{b}_{010}^{n-1}$ and $\mathbf{b}_{100}^{n-1}$, are produced in an analogous fashion.

Having computed a normal to every sampled point, the collection is stored together with the patch in a structure identical to that housing the surface points.

Imposing Continuity Constraints

Continuity constraints between two patches are imposed by warping one of the patches to fit its neighbor. The boundary between the two is smoothed by adjusting the necessary control points on one of the patches, while holding the others fixed. This precipitates a stretching effect. The boundaries in question are determined by the location of the hot point on the corresponding patches: The diagonal on which the

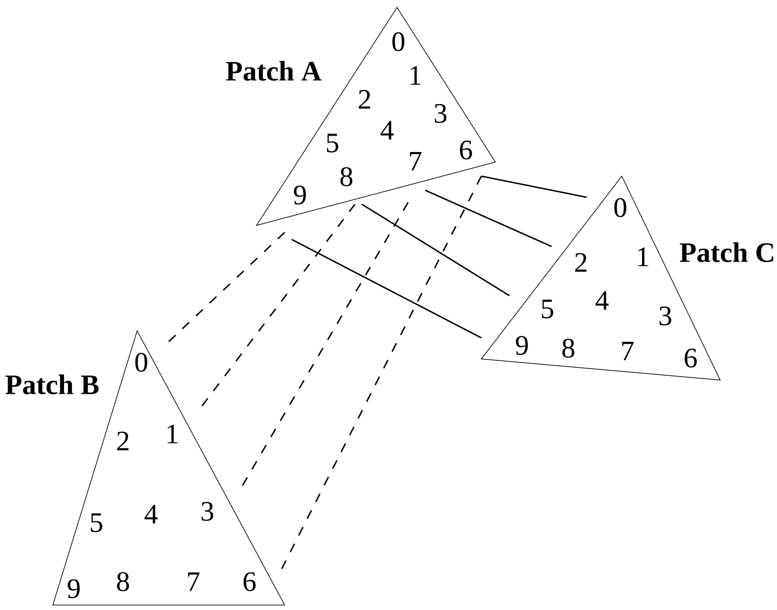

Figure 6.3: Imposing continuity between patches

point lies is designated as the one on which the continuity conditions will be imposed. Interior points are considered to lie on the "bottom" diagonal. Specifying corners is problematic as they simultaneously belong to two boundaries. In this case, an arbitrary choice is made depending on the orientation of the patch, and hence the selection cannot always be visually determined.

Another problem which arises when imposing continuity between two different diagonals comes from the sequential enumeration of control points. As is illustrated in Figure 6.3, the control points are, in most cases, matched in increasing order. For example, points {6, 7, 8, 9} of patch A in the figure are mapped respectively to points {0, 2, 5, 9} of patch C. In some cases, however, the order of the points must be flipped: points {9, 8, 7, 6} (decreasing order) of patch A are mapped respectively to points {0, 1, 3, 6}(increasing order) of patch B. This order flipping occurs only when the "bottom" diagonal of one patch is mapped to the "right hand" diagonal (by convention, the one containing the control point '1') of the other.

6.2.2 Rendering

Surface design is typically carried out using wireframe models, which are dressed with texture only at an advanced stage of the modeling process. One reason for this is that, by observing the wireframe model, the designer can simultaneously apprehend the shape of the object from the front and from the back, having nothing to obstruct the view. Physical models are traditionally constructed as ribbed frames for similar motives.

Another reason why textures show up relatively late in the design process is that ornamented objects are considerably more complex to render than wireframe ones. As such, they are computationally intensive to render and tend to hinder the response time of user interaction. Because of this, when the scene is put in motion (while rotating it, for example) and must be redrawn many times in quick succession, all objects are rendered in wireframe. When the motion stops, the scene is redrawn one final time with the appropriate use of texture.

For these reasons, the modeler must provide the rendition capacity both in wireframe and in full texture.

Wireframe Rendering

A distinction must be made between the *control mesh* which is the logical triangular structure of the control points, and the *surface mesh* which is the logical triangular structure of the sampled surface points. The wireframe model is produced by connecting each point on the surface mesh to the two closest points on the row immediately below it. The triangulation is performed by drawing a three-sided (hollow) polygon whose vertices lie at the three points. All points but those on the bottom row of the surface mesh are triangulated in this way.

It is often convenient to triangulate the control mesh as well in order to highlight the active (hot) patch. This is performed in an analogous manner to the triangulation of the surface mesh, using polygons of a different color.

General Texture Mapping

Of the two texturing techniques discussed in Chapter 4, only surface flattening has been fully implemented. The second method presented has been implemented partially, only to the extent that its advantage over surface flattening scheme becomes clear. Both implementation have adapted the results of the previous chapters to triangular domains.

The advantage of defining texture maps over piecewise surfaces is that local changes in surface geometry generally require only local changes to the texture map. When a texture mapping process is unaware of the underlying surface representation, it cannot disregard the regions unaffected by the change and must re-map the texture everywhere. Context sensitive maps, however, can determine when no changes are necessary along a patch boundary. In this case, the interior of the patch may be absolved from being remapped with no further ado.

Although nominally the texture is parameterized over the unit square, the texture *space* is regarded as a torus giving the texture an infinite extent. Texture coordinates outside of the unit square are reduced to their congruence class modulo 1, giving interior coordinates. The texture is then drawn onto the surface by interpolating between the mapped points using OpenGL's 2-dimensional texture mapping facilities `glTexImage2D*()`, `glTexEnv*()`, and `glTexCoord2*()`.

Piecewise Surface Flattening of Triangular Bézier Patches

Flattening of each patch is performed after it has been completely sampled but before the patch has been rendered. The preservation operator $\mathcal{P}$ requires both the location and the normal information at every sampled point; both of which have been computed previously during the sampling process.

Using the piecewise flattening algorithm described in Section 5.2, every point on an internal ring has two neighbors on each of the isoparametric curves comprising the ring behind it. Standard points therefore have two neighbors, corner points have four, and center points sometimes have a total of six. Each of these neighbors gives rise

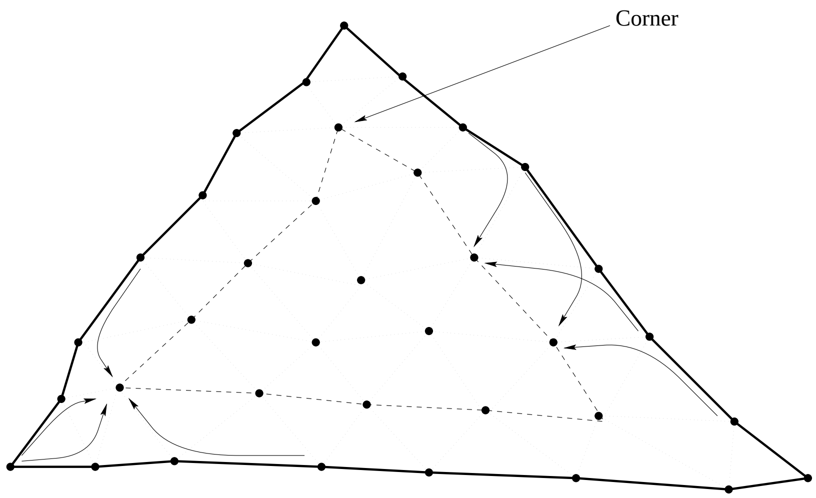

Figure 6.4: Flattening Bézier triangles: The thick line indicates the patch boundary that has been faithfully preserved; the dashed line indicates the concentric ring currently being flattened. The curved arrowheads show the formation of candidates for the interior points.

to one candidate point (Section 4.3.3), the creation of which requires two previously flattened points. One such point is supplied by the neighbor. The other chosen is the one on the external ring which immediately follows the neighbor (see Figure 6.4). The accumulated distortion is calculated using the dispersion metric $\mathcal{D}_{\mathcal{D}}$.

Surface-Area Parameterized Textures

The surface-area parameterized textures currently make use only of a uniform knot distribution along the isoparametric contours of the domain simplex. The relaxation method described in Section 4.4.2 has not yet been implemented, and thus the initial guessed parameterization – the input to the optimization process – is being used. Nevertheless, the results of this method show that, even without an optimized knot structure, the rendered objects generally have a superior appearance to those textured using the flattening technique.

6.2.3 User Interface

The selection of objects is afforded by mouse click, whereby the object beneath the pointer that is "closest" to the front of the screen is returned. This is done by assigning a unique identifier to every object drawn. In addition, every screen pixel maintains a stack of all the objects sharing it in screen-space, which are those mapped to it through the rendering pipeline. When the mouse is clicked, the list of all objects falling under the pointer is then reviewed, and the one with the smallest z-value returned (in a similar fashion to the z-buffer algorithm). Since the screen space is potentially very large, the stacks are created "on the fly" at the time of mouse-click, and only in an immediate neighborhood of the pointer. This is performed by redrawing the clipped scene into a non-visible selection buffer. The stack results are returned by OpenGL's `gluPickMatrix()` procedure.

When selection is used to produce points from scratch, the screen point must be backward projected through the rendering pipeline to produce a point in space. This requires a guess as to the appropriate z-value of the screen point, which may be taken to be on the $z = 0$ plane[1]. The back projection is performed by applying the inverse of the rendering projection matrix to the selected point. A straightforward method of inverting this matrix is presented in Appendix B.

Some Comments Regarding the Controller

The user interface to the modeler makes use of OpenGL's GLUT toolkit [Kil95] for the modeling window, and the Xforms library [ZO95] for the controller. Both libraries are designed to execute as stand-alone routines. Each has its own main event loop to which call-back routines are registered. To get the two working synchronously in parallel, GLUT was designated as the master event loop and Xforms as the slave. To do this, the *null* event of GLUT (*i.e.*, the default event when nothing else is happening) was set to poll the event loop of Xforms which was instructed to bypass is own null events. Thus, when the Xforms event queue is empty, control is returned to GLUT.

[1] A better, albeit more complicated, estimate would be the mean z value of all objects in the scene.

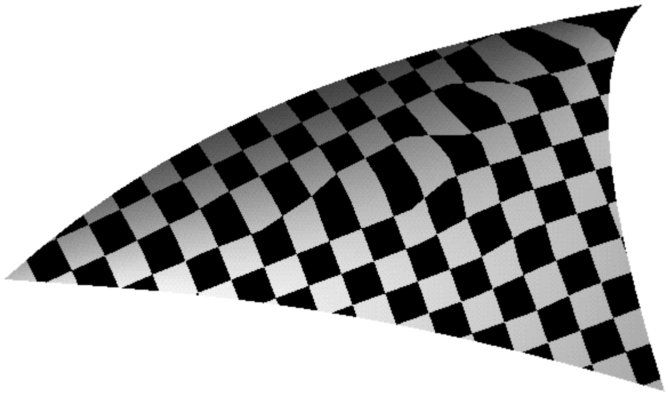

Figure 6.5: Single patch, textured using surface flattening

6.3 Results

Single Patch

For the purpose of demonstration, Figure 6.5 shows a Bézier triangle with a checkerboard texture that was mapped onto it using the surface flattening algorithm. It is immediately apparent that this technique performs well near the boundaries, which are faithfully flattened, but produces undesired artifacts in the center. Notice that much of the distortion is incurred around the top apex. This is because the flattening of the boundary ring commenced with that point. For reasons specified in Section 5.2.1, the flattened ring could not be properly closed and hence the isometry and conformity were lost at the final point. The distortion then bled inward toward the center of the patch, through the (non-faithful) flattening of concentric rings.

Figure 6.5 ought to be contrasted with Figure 6.6, which was textured using the initial uniform knot distribution of the surface-area method. For the sake of completion, Figure 6.7 shows the wireframe rendition of the same patch.

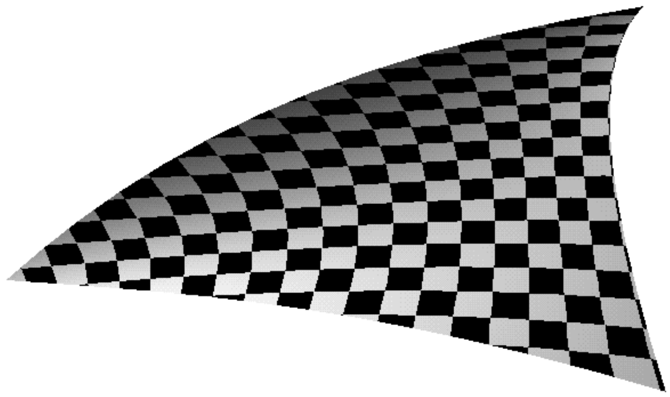

Figure 6.6: Single patch, textured using surface-area parameterization

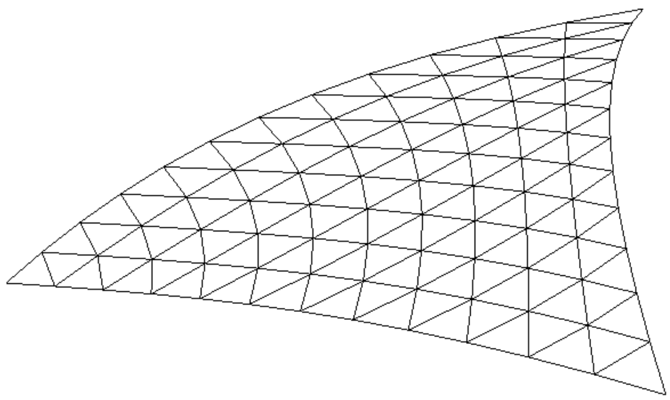

Figure 6.7: Single patch, wireframe rendition

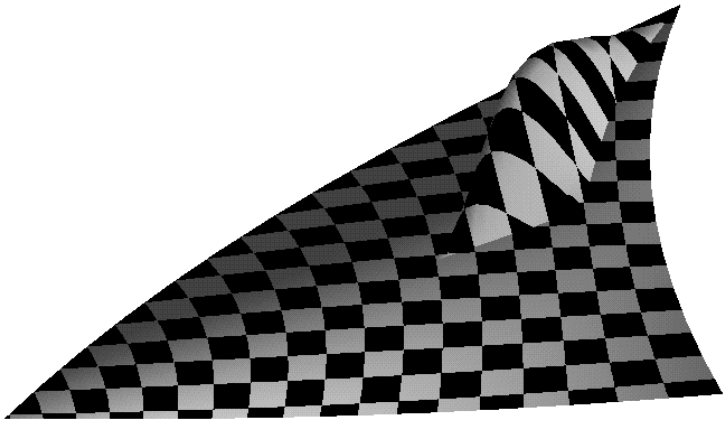

Figure 6.8: Multiple patches, textured using surface-area parameterization

Multiple patches

These results extend well to the case of surfaces constructed from multiple patches. Note that it is not always desirable to have a model that is entirely smooth. Sharp edges may be introduced to the model by forming seam lines that are C^0 continuous but not C^1. An example of such a surface can been seen in Figures 6.8 and 6.9.

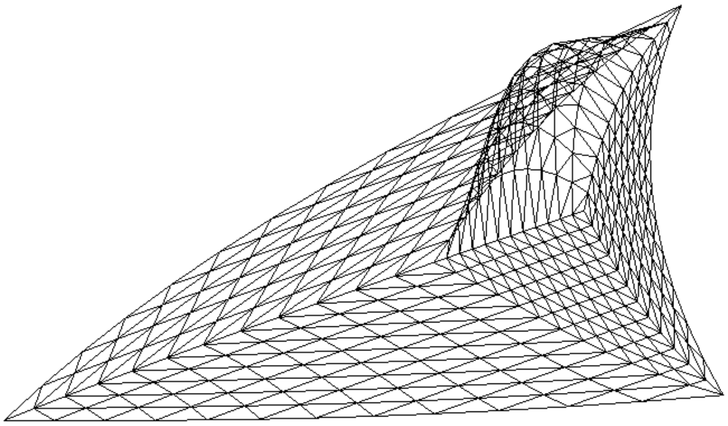

Figure 6.9: Multiple patches, wireframe rendition

Chapter 7

Conclusion

A geometrically intuitive design tool has been presented, which has the capability of constructing textured surfaces. The surfaces are constructed from a piecewise meshing of triangularly shaped surface elements. This reduces the occurrence of degenerate cases and alleviates the complications brought on by the tensor-product surface elements typically used. Methods have been discussed by which many patches may be interconnected in a smooth and apparently seamless fashion. Other methods have been proposed that will enable the application of texture to the composite surface in a cohesive fashion, using a patch-by-patch approach. An implementation of all the above in the form of a general surface modeler with texturing capacity has also been developed.

7.1 Thesis Contributions

The contributions of this thesis are twofold. First, the theory of triangular Bézier patches has been collected from a range of sources and coerced into a common form that is beneficial to the development of a tool for general surface modeling. Previous developments of this theory had neglected to deal with several topics. These include the issue of resolving the conflicting constraints incurred when imposing continuity between multiple patches on large surfaces, and the consideration of surface sampling techniques and the computational expenses involved thereof. Mysteriously absent from other developments has been the geometric intuition that provides the framework for this surface formulation. All have been addressed herein.

The second area of contribution is that of texture mapping. Traditional texture mapping has been applied to complete sampled surfaces which have consisted of one polygonal mesh, heeding no regard to their underlying representation. A correction to this approach has been attempted by developing models wherewith textures are applied directly to the individual patches comprising the mesh. Two methods of doing so were supplied. The first involved the flattening of patches and was an adaptation of a previously suggested scheme. This method was successful only to a limited degree, as the original application of the model was based on the use of many small patches between which texture discontinuities were encouraged. It has been shown that when applied to large surfaces, this method fails to produces satisfactory results.

A new method of texture mapping was proposed, and has shown very promising results. The approach is based on the concept of surface-area parameterization of the surface elements. The results produced by a partial implementation of this method exhibited far better surface consistency than those produced by other methods.

7.2 Future Work

The preliminary results produced by the surface-area texture maps were quite encouraging, even when applied to composite surfaces. This technique offers much promise and must be further developed. Specifically, the measures regarding the surface area elements referred to in Section 4.4.2 must be made precise. Further testing is also required.

At the beginning of this work, the author thought that a relationship could be found between the surface patch and the texture model by which the structure of the former could be exploited for the latter. So far, this goal has been elusive. Additional development is necessary to find an inexpensive method by which a sampled point will be computed directly from another nearby, perhaps using a forward differencing approach. The relaxation stage of the reparameterization would be greatly accelerated by this, thereby improving the entire texture mapping process.

Finally, the implementation of the modeler must (as always) be improved. The current software was written for the purpose of creating a prototype and does not offer the complete functionality of a full-grown textured-surface modeler.

Appendix A

Detailed Derivations

A.1 Trinomial Identity

Using the combinatorial identities $\binom{n}{\mathbf{i}} = \binom{n}{i}\binom{n-i}{j}$ and $\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$ one has that:

$$\begin{aligned}
\binom{n}{\mathbf{i}} &= \binom{n}{i}\binom{n-i}{j} \\
&= \left[\binom{n-1}{i} + \binom{n-1}{i-1}\right]\left[\binom{n-i-1}{j}\binom{n-i-1}{j-1}\right] \\
&= \binom{n-1}{i}\binom{n-i-1}{j} + \binom{n-1}{i-1}\binom{n-i-1}{j} + \binom{n-1}{i}\binom{n-i-1}{j-1} + \\
&\quad \binom{n-1}{i-1}\binom{n-i-1}{j-1} \\
&= \frac{(n-1)!}{i!j!(n-i-j-1)!} + \frac{(n-1)!}{i!(j-1)!(n-i-j)!} + \frac{(n-1)!}{(n-i)(i-1)!j!k!}[j+k] \\
&= \binom{n-1}{i,j,k-1} + \binom{n-1}{i,j-1,k} + \binom{n-1}{i-1,j,k},
\end{aligned}$$

because $j+k=n-i$.

A.2 Efficient Computation of Multinomial Coefficients

The trinomial coefficients may be computed efficiently using using the recursive identities:

$$\binom{n}{i,j,k} = \frac{i+1}{j}\binom{n}{i+1,j-1,k} = \frac{i+1}{k}\binom{n}{i+1,j,k-1}. \tag{A.1}$$

To incrementally generate the lookup table, one simply has to begin with the term $\binom{n}{n,0,0} = 1$ and work backwards from $i = n - j$ to $i = 0$ for each j using (A.1). A similar method was used in [Mil95] to evaluate the binomial coefficients.

Appendix B

Inverting the Projection Matrix

Fundamentally, the graphics pipeline takes the homogeneous coordinates of a point (x, y, z, w) from which it produces a new point (x', y') in the planar coordinate system of the physical device. The depth coordinate, z', is often maintained as well for use by hidden surface removal algorithms. Given a depth value, the point (x', y', z') may be embedded into $\mathbf{R}^4$ by setting $w' = 1$. Thus the graphics pipeline can be seen as an operator $\mathbf{Q}$ mapping $\mathbf{R}^4 \to \mathbf{R}^4$, $\mathbf{Q} : (x, y, z, w) \mapsto (x', y', z', w')$.

The matrix of $\mathbf{Q}$ is essentially composed of two separate parts: a transformation $\mathbf{L}$ followed by a projections $\mathbf{P}$. In other words, we have $\mathbf{Q} = \mathbf{PL}$, and hence $\mathbf{Q}^{-1} = (\mathbf{PL})^{-1} = \mathbf{L}^{-1}\mathbf{P}^{-1}$. The inversion of $\mathbf{Q}$, then, may be obtained by taking the product of the inverted $\mathbf{L}$ and the inverted $\mathbf{P}$. We show a simple way in which this can be done.

Transformations are composed of rotations and translations. The general form of a translation is given in [FvDFH92] by

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where (T_x, T_y, T_z) is the translation vector. Clearly the inverse of this translation is specified by the vector $(-T_x, -T_y, -T_z)$ and is hence given by

$$\mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The general form of a rotation is also given in [FvDFH92] by

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The 3×3 rotation submatrix (r_{ij}) is an orthonormal matrix; its inverse, therefore, is given by its transpose (r_{ji}). Hence $\mathbf{R}$ is also orthonormal. Its inverse is

$$\mathbf{R}^{-1} = \mathbf{R}^t = \begin{pmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Writing $\bar{\mathbf{T}}$ as the translation *vector* and $\bar{\mathbf{R}}$ as the rotation *submatrix*, we represent the above matrices in block form and compute their product:

$$\begin{aligned} RT &= \begin{pmatrix} \bar{\mathbf{R}} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & \bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \bar{\mathbf{R}} & \bar{\mathbf{R}}\bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{B.1}$$

Now $\bar{\mathbf{R}}$ is a 3×3 matrix, and $\bar{\mathbf{T}}$ is 3×1, so that $\bar{\mathbf{R}}\bar{\mathbf{T}}$ is 3×1, and is just the rotated translation vector. Similarly,

$$\begin{aligned} TR &= \begin{pmatrix} 1 & \bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{R}} & 0 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \bar{\mathbf{R}} & \bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{B.2}$$

Thus the product of rotations and translations are always of the form of Equation (B.2). It is easily verified that these matrices form a group under the operation of matrix multiplication, where the inverse of every matrix of this form is given by

$$\begin{pmatrix} \bar{\mathbf{R}} & \bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \bar{\mathbf{R}}^t & -\bar{\mathbf{R}}^t\bar{\mathbf{T}} \\ 0 & 1 \end{pmatrix}. \tag{B.3}$$

It follows, that every transformation matrix $\mathbf{L}$ is a member of this group, and is therefore in the form of Equation (B.2). Its inverse is thus given by Equation (B.3), which is easily computed by transposing the 3×3 rotation submatrix on the top-left, and applying it to the negated translation vector on the top-right.

The perspective projection matrix $\mathbf{P}$ is given in [NDW93] as

$$\mathbf{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \tag{B.4}$$

where l, r, b, n, f are respectively the left, right, bottom, top, near, and far clipping planes making up the frustum which defines the prospective view volume. This matrix is well defined provided $l \neq r, t \neq b$, and $n \neq f$. Its inverse is given by

$$\mathbf{P}^{-1} = \begin{pmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{pmatrix}, \tag{B.5}$$

which can be computed directly from Equation (B.4) by rearranging and inverting its individual terms.

The inverse of the projection matrix $\mathbf{Q}$ is now given by the product of $\mathbf{L}^{-1}$ from Equation (B.3) and $\mathbf{P}^{-1}$ from Equation (B.5).

Bibliography

[BFK84] Wolfgang Böhm, Gerald Farin, and Jürgen Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1:1–60, 1984.

[Blo85] Jules Bloomenthal. Modeling the mighty maple. *SIGGRAPH 85, Proceedings of Computer Graphics*, 19(3):305–311, July 1985.

[BN76] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.

[BVI91] Chakib Bennis, Jean-Marc Vézien, and Gérard Iglésias. Piecewise surface flattening for non-distorted texture mapping. *SIGGRAPH 91, Proceedings of Computer Graphics*, 25(4):237–246, July 1991.

[Car95] Jeromy Carrière. Evaluating tensor product and triangular Bézier patches. Technical Report CS-95-22, University of Waterloo, May 1995.

[Cat74] Edwin A. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of Computer Sciences, University of Utah, December 1974.

[CMS88] Michael Chen, S. Joy Mumford, and Abigail Sellen. A study of interactive 3-D rotation using 2-D control devices. *Computer Graphics*, 22(4):121–129, August 1988.

[dB87] Carl de Boor. B-form basics. In Gerald Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 131–148. SIAM, 1987.

[dC59] Paul de Casteljau. Outillages méthodes calcul. Technical report, André Citroën Automobiles SA, Paris, 1959.

[dC63] Paul de Casteljau. Courbes et surfâces a pôles. Technical report, André Citroën Automobiles SA, Paris, 1963.

[dC76] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice-Hall Inc., Englewood Cliifs, NJ, 1976.

[dC92] Paul de Casteljau. POLynomials, POLar forms, and interPOLation. In Tom Lyche and Larry L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 57–68. Academic Press, New York, 1992.

[DM92] Tony DeRose and Stephen Mann. An approximatelt G^1 cubic surface interpolant. In Tom Lyche and Larry L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 185–196. Academic Press, New York, 1992.

[Far79] Gerald Farin. *Subsplines über Dreiecken.* PhD thesis, TU Braunschweig, Braunschweig, F.R.G., 1979.

[Far86] Gerald Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3:83–127, 1986.

[Far93] Gerald Farin. *Curves and Surfaces for CAGD.* Academic Press, London, 3rd edition, 1993.

[FFC87] Eugene Fiume, Alain Fournier, and V. Canale. Conformal texture mapping. In G. Maréchal, editor, *Eurographics 87*, pages 53–64. Elsevier Science Publishers B. V. (North-Holland), 1987.

[FvDFH92] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1992.

[GS93] Günter Greiner and Hans-Peter Seidel. Splines in computer graphics: Polar forms and triangular B-spline surfaces. In *EUROGRAPHICS 93: State of the Art Reports*. Eurographics, 1993.

[Her87] Gary Herron. Techniques for visual continuity. In Gerald Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 163–174. SIAM, 1987.

[HH90] Pat Hanrahan and Paul Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. *SIGGRAPH 90, Proceedings of Computer Graphics*, 24(4):215–223, August 1990.

[HL93] Josef Hoscheck and Dieter Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, Wellesley MA, 1993. Larry L. Shumaker trans.

[Hos84] Josef Hoschek. Detecting regions of undesirable curvature. *Computer Aided Geometric Design*, 1:183–192, 1984.

[Kil95] Mark J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface: API Version 2*. Silicon Graphics, Inc., August 1995.

[Kre59] Erwin Kreyszig. *Differential Geometry*. University of Toronto Press, Toronto, 1959.

[LD89] Charles T. Loop and Tony D. DeRose. A multisided generalization of Bézier surfaces. *ACM Transactions on Graphics*, 8(3):204–234, July 1989.

[Mil95] Robert D. Miller. Quick and simple Bézier curve drawing. In Alan Wm Paeth, editor, *Graphic Gems V*, pages 206–209. AP Professional, 1995.

[MLL+92] S. Mann, C. Loop, M. Lounsbery, D. Meyers, J Painter, T. DeRose, and K. Sloan. A survey of parametric scattered data fitting using triangular interpolants. In H. Hagen, editor, *Curve and Surface Design*. SIAM, 1992.

[MYV93] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. *SIGGRAPH 93, Computer Graphics Proceedings*, 27(4):27–34, August 1993.

[NDW93] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programmin Guide: The Official Guide to Learning OpenGL, Release 1*. Silicon Graphics, Inc., 1993.

[Pet91] Jörg Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–246, 1991.

[Pip87] Bruce R. Piper. Visually smooth interpolation with triangular Bézier patches. In Gerald Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 221–223. SIAM, 1987.

[Ram88] Lyle Ramshaw. Béziers and B-splines as multiaffine maps. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 757–776. Springer-Verlag, 1988.

[Sei91] Hans-Peter Seidel. Symmetric recursive algorithms for surfaces: B-patches and the de Boor algorithm for polynomials over triangles. *Constructive Approximation*, 7:257–279, 1991.

[Ste96] A. James Stewart. Personal communication, 1996.

[SWS95] Takafumi Saito, Guo-Jin Wang, and Thomas W. Sederberg. Hodographs and normals of rational curves and surfaces. *Computer Aided Geometric Design*, 12:417–430, 1995.

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using Reaction-Diffusion. *SIGGRAPH 91, Proceedings of Computer Graphics*, 25(4):289–298, July 1991.

[vW86] Jarke J. van Wijk. Bicubic patches for approximating non-rectangular control-point meshes. *Computer Aided Geometric Design*, 3:1–13, 1986.

[Wil59] T. J. Willmore. *An Introduction to Differential Geometry.* Oxford University Press, 1959.

[WK91] Andrew Witkin and Michael Kass. Reaction-Diffusion textures. *SIGGRAPH 91, Proceedings of Computer Graphics*, 25(4):299–308, July 1991.

[WW88] M. Watkins and A. Worsey. Degree reduction for Bézier curves. *Computer Aided Design*, 7(20):398–405, 1988.

[WW92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* ACM Press, 1992.

[Zho85] J. Zhou. *The Positivity and Convexity of Bézier Polynomials Over Triangles.* PhD thesis, Beijing University, 1985.

[ZO95] T. C. Zhao and Mark Overmars. *Forms Library: A Graphical User Interface Toolkit for X*, 1995.