COMPUTER SYSTEMS LABORATORY

DEPARTMENTSOFELECTRICALENGINEERINGANDCOMPUTERSCIENCE
STANFORDUNIVERSITY · STANFORD, CA94305

Parametric Curves, Surfaces and Volumes in Computer Graphics and Computer-Aided Geometric Design

James H. Clark

Technical Report No. 221

November 1981

This research was partially supported by NASA Ames Research Center.

Parametric Curves, Surfaces and Volumes in Computer Graphics and Computer-Aided Geometric Design

James H. Clark

Technical Report No. 221

November 1981

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

Abstract

This document has four purposes. it is a tutorial in parametric curve and surface representations, it describes a number of algorithms for generating both shaded and line-drawn pictures of bivariate surfaces and trivariate volumes, it explicitly gives transformations between all of the widely used curve and surface representations, and it proposes a solution to the problem of displaying the results of three-dimensional flow-field calculations.

Key Words and Phrases: Parametric curves, surfaces, volumes, splines, geometric modeling, hidden surface algortihms.

Table of Contents

Introduction

The tutorial portion of this report is Chapters 1 through 4. As a tutorial it describes in detail the techniques used in generating a geometric curve or surface representation with desired properties. It also develops and explains the four widely used cubic representations, Hermite(Coons), Bezier, B-splines and Cardinal splines, and gives transformations that transform from one to the other. In Section 2.2 of Chapter 3, the Bezier form is compared extensively with the Coons form, and it is shown that the Bezier form is superior to the Coons form for manipulating surfaces. This chapter also shows how the user might construct his own geometric representation. In addition, the Bezier and B-spline bases are shown to have properties that make them suitable for recursive subdivision problems.

Chapter 5 then presents several algorithms for generating pictures of parametric surfaces. The first algorithm described makes high quality shaded pictures of parametric surfaces. The second one is an extension of the surface algorithm to make shaded pictures of contours of constant pressure for solutions of 3-D fluid-flow problems. This is a new algorithm, and with the proper equipment, it should prove *very* valuable in providing feedback in this area of fluid-dynamics research. The third algorithm presented is a solution to the problem of finding the boundaries of intersections of arbitrary degree surface patches. Unless the reader is very familiar with the properties of the Bezier representation and recursive algorithms, it is unlikely that Chapter 5 will be understandable without first reading the other chapters.

Chapter 1
Parametric Polynomial Curves

In this chapter we develop mathematical formulations for curves that are polynomial functions of a single parameter variable. The parametric straight line that we have already used extensively is an example of a degree one polynomial curve. As with the straight line, each component of a curve will be expressed as a single-valued scalar function of the parameter variable. For example, in three dimensions, a curve C expressed as a function of parameter t is of the form

$$C(t) = [\ x(t)\ \ y(t)\ \ z(t)\ ].$$

We are interested in using higher degree polynomial curves because they provide higher level geometric modelling primitives than that provided by the straight line. Complex curves that require many straight line segments joined end-to-end in order to accurately represent them can be modelled to the same accuracy with fewer parabolic segments, thereby allowing a reduction in the data required to represent the curve. The same curve can be represented by even fewer segments of higher degree polynomials. In addition, one has the flexiblity with higher degree segments to represent a curve with not only positional continuity, which is provided by straight line segments, but also tangent, curvature and higher derivative continuity. Another reason for developing these curves is in order to provide a basis for the surface and volume formulations of the next chapter. To provide informative and accurate displays of three-dimensional geometric data, it is often convenient to model the data with functions of two or three parameter variables, representing respectively

surfaces or volumes. Many of the algorithms for producing high quality pictures by computer are based upon these formulations. The material presented in this chapter is essential for understanding the next chapter.

In the first section, using the familiar equation of the straight line, the emphasis is on an approach to deriving polynomial representations in terms of geometrically meaningful information, such as the end points of the line. In deriving the equation of the straight line, an approach is used that makes the derivation of higher degree formulations very straightforward. After a brief presentation of parametric quadratic polynomials in Section 2, Section 3 is devoted to developing and analyzing the properties of cubic representations. Particular emphasis is given to the equivalence of the various widely used representations for cubics and to the special properties of the Bezier representation. Cubic Cardinal and B-splines are also discussed in this section. Section 4 then presents the Bezier formulation for arbitrary degree curves and suggests ways in which this formulation is suitable for a wide class of numerical problems.

1. The Straight Line.

We have seen that the straight line segment between points p_1 and P2 is given by the parametric equation

$$L(t) = (P2 - p_1) t + p_1 , \qquad 0 \le t \le 1. \quad (1)$$

Since this equation is linear in t, it represents a straight line segment between, the two points. Note that the vectors p_1 and p_2 may have any number of components. This equation represents a

straight line in a space with any number of dimensions. Since the curves we develop will also be valid for an arbitrary number of dimensions, we will keep the vector notation throughout this chapter, referring to components only when necessary to clarify how the curve might be displayed.

A simple way to derive equation 1 is to consider the degree one polynomial in t in its algebraic form,

$$V(t) = at + b \quad . \qquad (2)$$

This vector function represents n scalar functions, where n is the number of dimensions being considered. For convenience of notation, let us express equation 2 as a matrix product:

$$V(t) = \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = T_1 \quad M_1 \ , \qquad (3)$$

where M_1 is the column vector whose elements are the coefficient vectors in equation 2. M_1 is a 2 by n matrix, since the vectors a and b each have n components. We will refer to the vector $T_1 = [\ t\ 1\ ]$ as the primitive, degree-one polynomial basis.

Equation 3 is the canonical equation for a straight line. By chosing values for a and b, that is, by specifying M_1 , we obtain a particular straight line. Because M_1 in the form of equation 3 does not necessarily express anything geometrically intuitive, it is useful to express it in terms of the end points of a straight line. In order to uniquely determine M_1 we must specify two vector constraints, where each of the two vectors has n components. Let us require that

$$V(0) = P1$$

and $V(1) = p_2$.

That is, the value of M_1 must be such that the line (curve) is at p_1 when t=0 and at p_2 when t=1. These choices of the values for t, t=0 and t=1, are purely for computational convenience. Substituting these constraints into equation 3 yields the two equations

$$p_1 = [0 \quad 1] \; M_1$$
$$\text{and } P2 = [1 \quad 1] \; M_1 \;,$$

which can be combined in the single matrix equation

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} M_1 \;.$$

In this form we see that M_1 can be expressed in terms of the two points p_1 and p_2 by evaluating the inverse of the matrix multiplying M_1 in this equation:

$$M_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \;.$$

Substituting this result back into equation 3 we obtain

$$V(t) = [\, t \quad 1 \,] \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \;,$$

$$\text{or} \quad V(t) = T_1 \; M_1 \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \qquad (5)$$

Equation 5 represents the linear function that passes through p_1 when t=0 and p_2 when t=1. It is identical to equation 1, as can be verified by carrying out the matrix multiplications. In order to establish nomenclature for later reference, we write equation 5 in the following two ways:

$$V(t) = T \begin{bmatrix} dp \\ p_1 \end{bmatrix} , \qquad (5.1)$$

where $dp = p_2 - p_1$, and

$$V(t) = \begin{bmatrix} f_1(t) & f_2(t) \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} , \qquad (5.2)$$

where $f_1(t) = 1-t$
and $f_2(t) = t$.

Equation 5.1 is the algebraic form of 5, since it clearly expresses the algebraic form of equation 1, with the substitutions a=dp and $b=p_1$. Equation 5.2 is called the geometric form, and we refer to the two functions $f_1(t)$ and $f_2(t)$ as the geometric basis functions, since the curve V(t) is expressed as a linear combination of them with the points p_1 and p_2 . Equation 5.2 is called the geometric form because it is expressed in terms of the geometric points p_1 and p_2. We will refer to the matrix

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

as the geometric basis matrix. By pre-multiplication, it transforms the geometrically intuitive point information into the algebraic matrix M_1 of equation 1. By post- multiplication, it transforms the primitive basis $[\, t \;\; 1 \,]$ into the geometric basis functions $[\, f_1(t) \;\; f_2(t) \,]$. This nomenclature will be used freely for the remainder of this chapter and the next.

Exercises.

1-1. Evaluate the derivative of equation 5 with respect to

t and verify that the result gives the correct tangent for the line.

l-2. Derive equation 5 by specifying V(0) and V'(0) rather than V(0) and V(l), where the prime denotes derivative with respect to t.

2. Quadratic Curves.

This section briefly describes several quadratic geometric representations. For many puroses, the cubics developed in the next section are of more practical value. This section is included primarily to further illustrate the approach of the previous section.

By analogy with equation 1, a degree 2 polynomial curve has the algebraic form

$$\begin{aligned} C(t) &= at^2 + bt + c \\ &= [\, t^2 \quad t \quad 1 \,]\; M_2 \,, \qquad (6) \\ &= T_2 \quad M_2 \quad , \end{aligned}$$

where M_2 is the 3 by n algebraic coefficient matrix analogous to M_1 for degree one curves. M_2 has 3 unspecified algebraic, vector components, a, b, and c. These components can be determined uniquely by specifying 3 vector conditions on C(t), analogous to the conditions specified for determining M_1. Any set of three distinct specifications will suffice, e.g. C(t) for three different values of t, C(t) for two values of t and C'(t) for another value of t, etc. The only restriction is that each of the unknowns must appear at least once in the set of equations gen-

erated by the specifications. Each set of specifications will yield another geometric formulation.

As an example, let us specify

$$C(0) = P1 \quad I$$
$$C(.5) = P2 \ ,$$
$$\text{and } C(1) = p_3 \ .$$

substituting these into equation 6 and arranging them in matrix form, we get

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ .25 & .5 & 1 \\ 1 & 1 & 1 \end{bmatrix} M_2$$

from which, by inverting the matrix multiplying M_2, and then substituting the result for M_2 back into equation 6, we obtain the geometric form for this set of specifications:

$$C(t) = T_2 \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} . \qquad (7)$$

Now in order to determine the quadratic curve passing through the point pl when t=0, the point p_2 when t=.5 and the point p_3 when t=1, we simply substitute our three geometric points into this geometric form for the curve.

Figure 1-1 illustrates a two dimensional quadratic curve generated using this geometric form. The derivations of several other useful geometric representations are left as exercises.

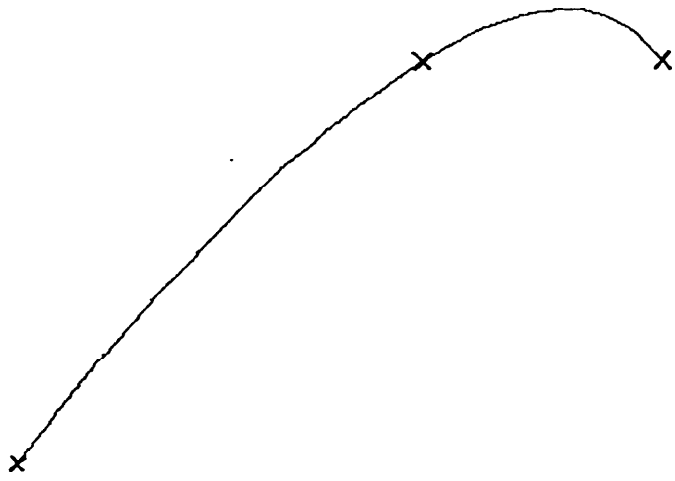


Figure 1-1 Quadratic interpolating curve. The X's indicate the points through which the curve passes when the parameter t has the values 0, 1/2 and 1.

Exercises.

2-1. show by substitution that the curve of equation 7 satisfies the constraints specified for it.

2-2. Find the basis matrix for C(O), C'(0) and C''(0).

2-3. Find the basis matrix for C(O), C'(0) and C(1).

2-4. Given the equation of a parabola in two dimensions, $y=ax^2+bx+c$, express it in the parametric form of equation 7.

3. Parametric Cubics

Parametric cubics are the lowest degree parametric polynomials with sufficient flexibility to allow both positions and tangents to be specified at two distinct points on a curve. For this reason, it is possible to readily join two cubic segments together with continuity of tangent. Consequently, cubics are more useful than quadratics in a wide variety of graphical and geometric design applications. Much of the surface patch development in the next chapter depends upon a good understanding of the material in this section. Therefore, we will discuss the widely used geometric formulations in detail.

All parametric cubics have the same algebraic form, which is that specified in the equation

$$C(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_3 , \qquad (8)$$

$$= a t^3 + b t^2 + c t + d$$

$$= T_3 \quad M_3 \ ,$$

where M_3 is a column vector with elements a, b, c and d and T_3 is the primitive cubic basis. Starting with this algebraic form, we can develop numerous different geometric formulations in a manner similar to that used in the previous two sections. That is, we may determine M_3 by specifying a set of four vector constraints on equation 8. Each set of constraints yields another geometric formulation.

3.1 Cubic Hermite Interpolation.

This section is concerned with a formulation known as Hermite interpolation. It is the type of interpolation used in the bicubic Coons Patch[1,2], which has been widely used in geometric surface design applications. The Hermite form for cubics is obtained by specifying the curve in terms of its end-points and end-tangents. As in the previous two sections, we may specify these conditions for any two values of the parameter t, but for computational convenience, we will chose the values t=0 and t=1. Also, we will consider only the segment of the cubic that lies between these two values. Since we are specifying the tangent to the curve, we must evaluate the derivative of equation 8:

$$C'(t) = [\ 3t^2 \quad 2t \quad 1 \quad 0\ ]\ M_3 \ .$$

Now setting

$$C(0) = p_1 \ ,$$
$$C(1) = p_2 \ ,$$
$$C'(0) = s_1 \ ,$$
$$\text{and } C'(1) = s_2 \ ,$$

$$\begin{bmatrix} p_1 \\ p_2 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} C(0) \\ C(1) \\ C'(0) \\ C'(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_3 \quad .$$

Evaluating the inverse of the matrix multiplying M_3 , we find

$$M_3 = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ s_1 \\ s_2 \end{bmatrix}$$

$$= M_h \quad G_h \quad ,$$

where, for notational convenience, we have labeled the 4 by 4 basis matrix in this equation as Mh and the geometric information Vector as G_h. Substituting this result into equation 8, we finally get

$$C(t) = [\ t^3 \ \ t^2 \ \ t \ \ 1\ ] \quad M_h \quad G_h. \qquad (9)$$

This is the Hermite geometric form for cubics. If we transform the primitive polynomial basis by Mh to obtain the Hermite basis functions, we put equation 9 in a form that expresses an intuitive interpretation of the Hermite representation:

$$C(t) = [\ h_1(t) \ h_2(t) \ g_1(t) \ g_2(t)\ ] \ G_h \qquad (10)$$

$$= h_1(t)\ p_1 + h_2(t)\ p_2 + g_1(t)\ s_1 + g_2(t)\ s_2 \ ,$$

where

$$h_1(t) = 2t^3 - 3t^2 + 1,$$

$$h_2(t) = -2t^3 + 3t^2,$$
$$g_1(t) = t^3 - 2t^2 + t,$$
and $$g_2(t) = t^3 - t^2.$$

The basis functions $h_1(t)$ and $h_2(t)$ "blend" together the two end-points of the curve, p_1 and p_2, and the functions $g_1(t)$ and $g_2(t)$ "blend" together the two end-tangents of the curve, sl and s_2. They are consequently sometimes called blending functions. We could have arrived at these functions by requiring that the curve have the form of equation 10 and determining the functions that exhibit the desired blending behavior. Coons arrived at them in this way[1]. These basis functions are shown in Figure 1-2.

Figure l-3 illustrates several curve segments drawn from this set of conditions. Each successive curve was drawn by increasing the magnitude of the tangent vector at the endpoint t=1. Note that curve 3 has passed a critical point beyond which in order to satisfy the end conditions it must form a spatial loop. In many graphics applications, such as hidden surface removal, and geometric design applications it is important to know when a curve or surface exhibits this behavior. In the next section, we discuss how to predict it.

3.2 Bezier Cubics.

In dealing with individual cubic segments, the particular geometric form one uses depends upon what is most convenient for the context in which they are being used. In graphics and geometric design, the Bezier form has been found to be extremely useful. One of the reasons for this is that in the Bezier form one can explicitly predict the behavior of an entire curve segment on the basis of the behavior of four geometric control points. In contrast, the Hermite representation gives one an

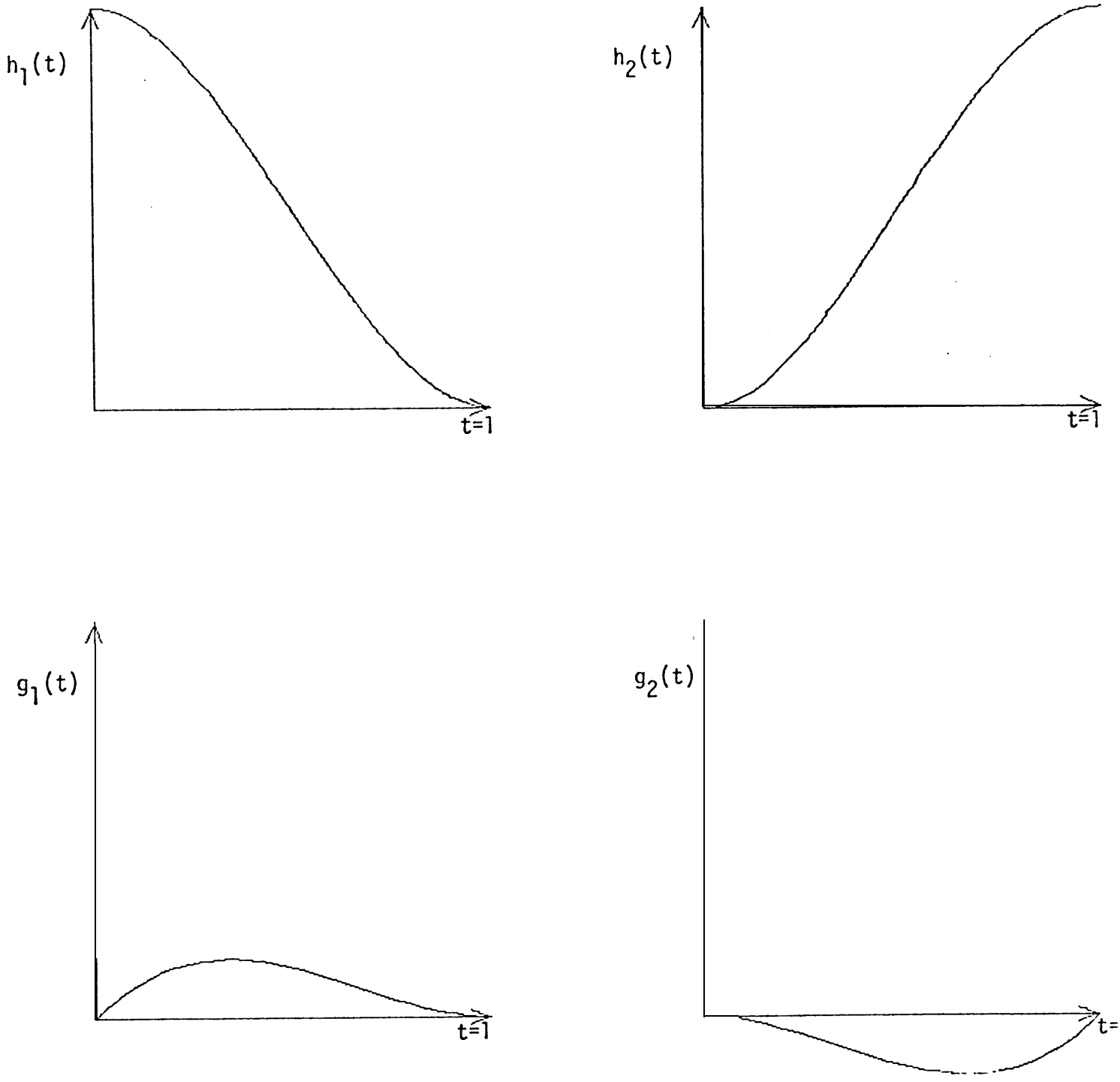


Figure 1-2 The hermite basis functions for cubics. The $h_i(t)$ blend together the end positions and the $g_i(t)$ blend together the end tangents of a cubic curve.

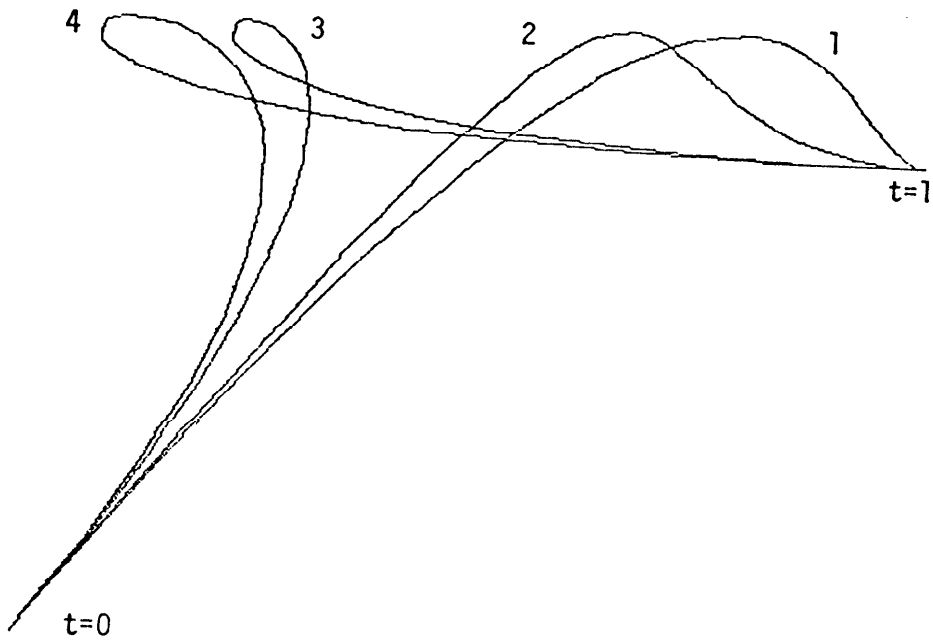


Figure 1-3 Four different hermite curves. Each successive curve was generated by increasing the magnitude of the tangent at t=1.

explicit understanding of the behavior of a curve segment at its endpoints only, while its global behavior is only implicitly known. This section presents the principle properties of the Bezier cubic representation that are of interest in the graphics/geometric-design context. Since the prope'rties of Bezier curves hold for any degree curve, we will defer presenting the mathematics from which the properties are derived until Chapter 2, which deals with curves of arbitrary degree.

The Bezier cubic representation expresses the tangents sl and s_2 of the Hermite form in terms of difference vectors between two geometric points. Thus all of the information about the curve is contained in geometric points, which are referred to as control points. These four geometric control points, ql, q_2, q_3 and $_{44'}$ are related to the Hermite geometry information by the equations

$$\begin{aligned} Pl &= q_1 , \\ p_2 &= q_4 , \\ s_1 &= 3 (q_2 - q_1) , \\ \text{and} \quad s_2 &= 3 (q_4 - q_3) . \end{aligned} \tag{11}$$

Figure 1-4 shows a set of 4 points, q_i , connected by an open polygon and the generated curve. The only difference between these two representations is in the way in which the end tangents to the curve are specified. The transformation from one representation to the other is very simple. However, the Bezier representation provides much more intuitive information, especially in the surface formulation.

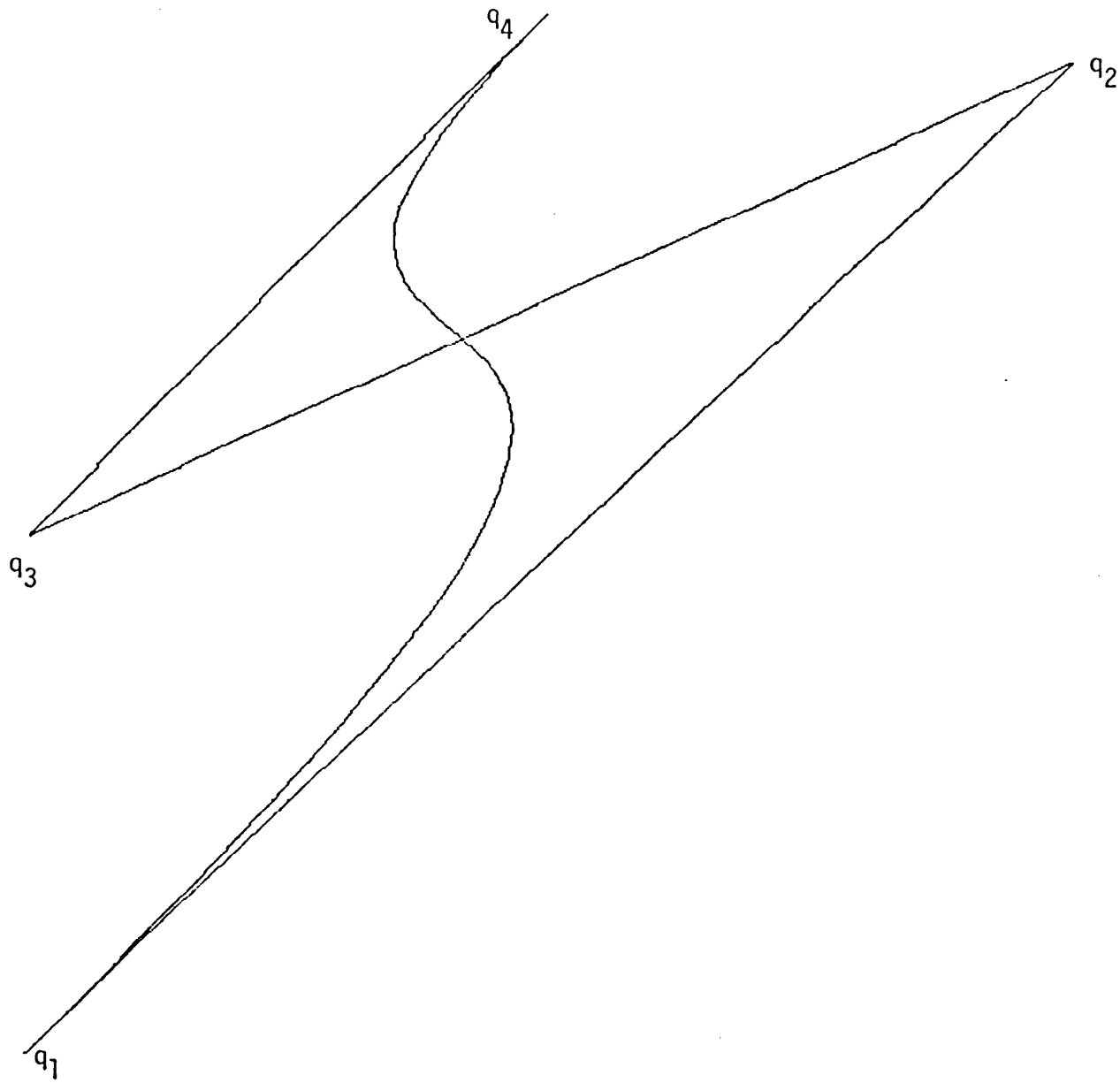


Figure 1-4 A cubic Bezier curve. The four points q_i are the points approximated by the curve; each is multiplied by the appropriate binomial basis function.

Expressed in matrix form, equations 11 become

$$\begin{matrix} p_1 \\ p_2 \\ s_1 \\ s_2 \end{matrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}, \qquad (12)$$

or $G_h = M_{hb} G_b$,

where we have labeled the column vector of Bezier geometry information on the right by G_b, and the matrix that transforms it into Hermite form by M_{hb}. Now if this equation is substituted into equation 9, we get the Bezier geometric form for a cubic

$$C(t) = [\, t^3 \; t^2 \; t \; 1 \,] \; M_b \; G_b \;, \qquad (13)$$

where

$$M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = M_h \; M_{hb} \qquad (14)$$

is the Bezier basis matrix.

One of the properties of the Bezier geometric form becomes apparent when the primitive basis is transformed by Mb. The four Bezier basis functions obtained can be expressed after rearranging as

$$b_1(t) = (1-t)^3,$$
$$b_2(t) = 3t\,(1-t)',$$
$$b_3(t) = 3t^2\,(1-t),$$

and $b_4(t) = t^3$, (15)

so that equation 14 becomes

$$C(t) = b_1(t)\, q_1 + b_2(t)\, q_2 + b_3(t)\, q_3 + b_4(t)\, q_4 . \quad (16)$$

Referring to Figure l-5, which illustrates the Bezier basis functions of equation 16, we see that all of them range between 0 and 1. Also, we see from equation 15 that the four of them sum to 1 for all values of t. Equation 16 shows that the curve is just the weighted average of the control points, with the Bezier basis functions as weights. Since the basis functions have the properties indicated here, the curve cannot go outside of the convex hull of the control points. (The convex hull is the least enclosing hull that contains all of the points.) This convex hull property is very important. Since all cubic curves have a Bezier representation, one can easily determine the limiting behavior of any cubic by transforming it into Bezier form, since it cannot go outside of its Bezier convex hull.

Another property of the binomial basis is referred to as the variation diminishing property[3]. A Bezier curve crosses a line (a plane in 3-D) no more times than the open polygon formed by connecting its control points crosses it. The curve mimics the behavior of the control polygon. In fact, it is useful to think of the polygon as a zeroeth order representation of the curve. A first order representation is obtained by recursively subdividing the curve, as described in the next chapter. Similarly for a second order representation, etc. This property allows us to predict the behavior of A Bezier curve from the behavior of its control polugon. This is especially useful in determining the

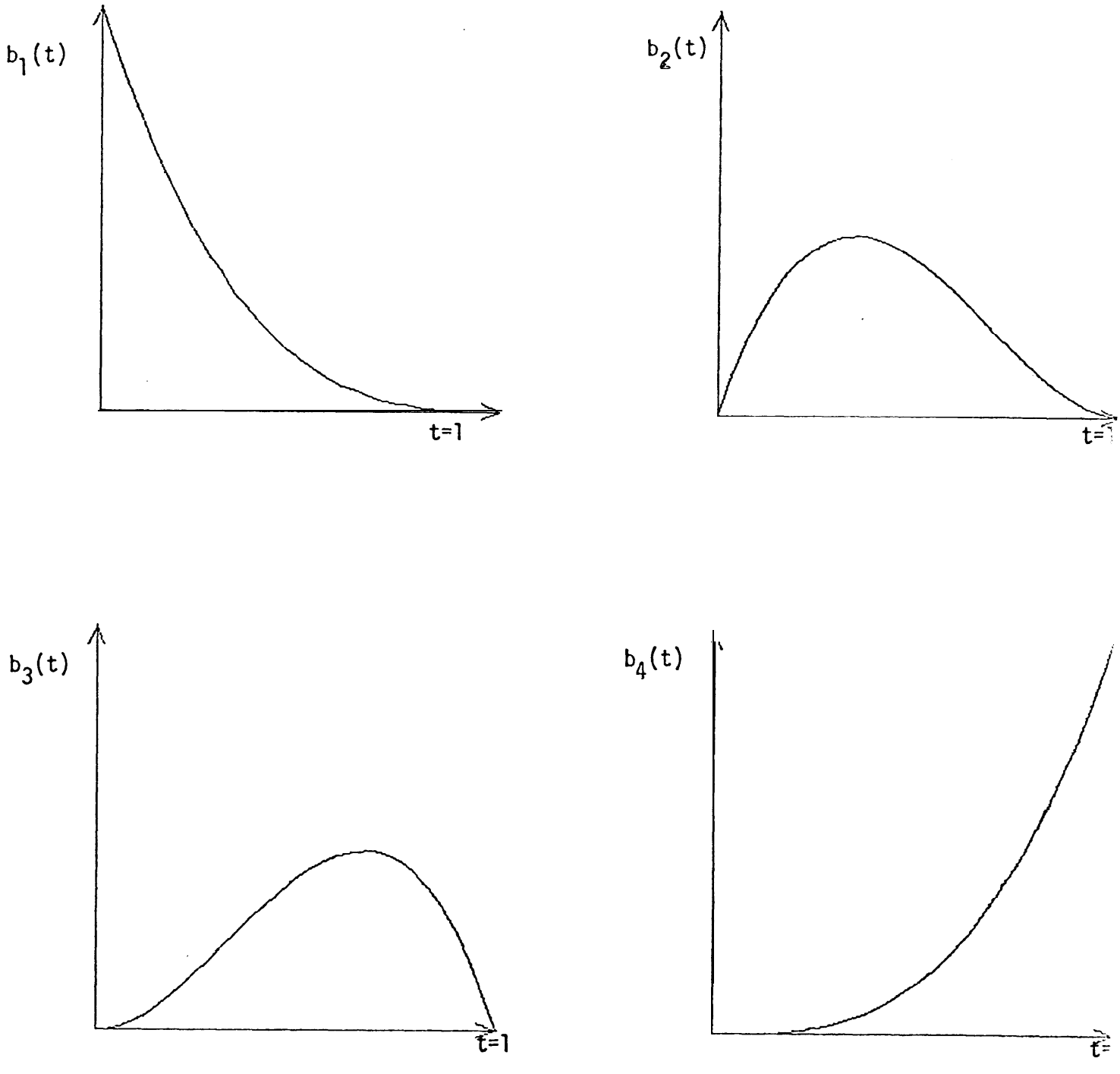


Figure 1-5 The degree 3 binomial distribution functions, used as the basis functions of Bezier cubic curves.

behavior of surface patches.

3.4 Segment Continuity and Cardinal Splines.

One of the main reasons for introducing cubic curve segments is that they have sufficient flexibility to be joined to other segments with continuity of tangent as well as position. This continuity can be accomplished in differents ways. For example, in the Hermite form, two cubic segments may be joined with continuity of tangent by simply adjusting their tangents to be the same. In the Bezier form, the same thing can be accomplished by adjusting the control points as shown in Figure 1-6.

Both of these approaches establish continuity by explicitly adjusting the tangents to the curves at their endpoints. An alternative approach is provided by splines. A spline is a composite curve made up of segments that implicitly join together with continuity of derivative of order n, usually referred to as C^n continuity. Using spline techniques, continuity is "automatic", that is, C^n continuity is guaranteed by virtue of the basis used in representing the individual segments. A spline representation that is simply related to the two cubic representations we have discussed is the cubic Cardinal spline. We give a simple development of this spline here by deriving its basis matrix in terms of the Hermite basis.

Suppose we wish to pass a smooth (continuous in tangent) curve through a sequence of points in space, as shown in Figure 1-7a. Assume for now that we have both a position and a tangent vector at each of these points, i.e. at each point we know p_i and s_i of the Hermite representation. Then in terms of the Hermite geometric form, a composite curve that is C^1 continuous may be represented by a sequence of cubic Hermite segments

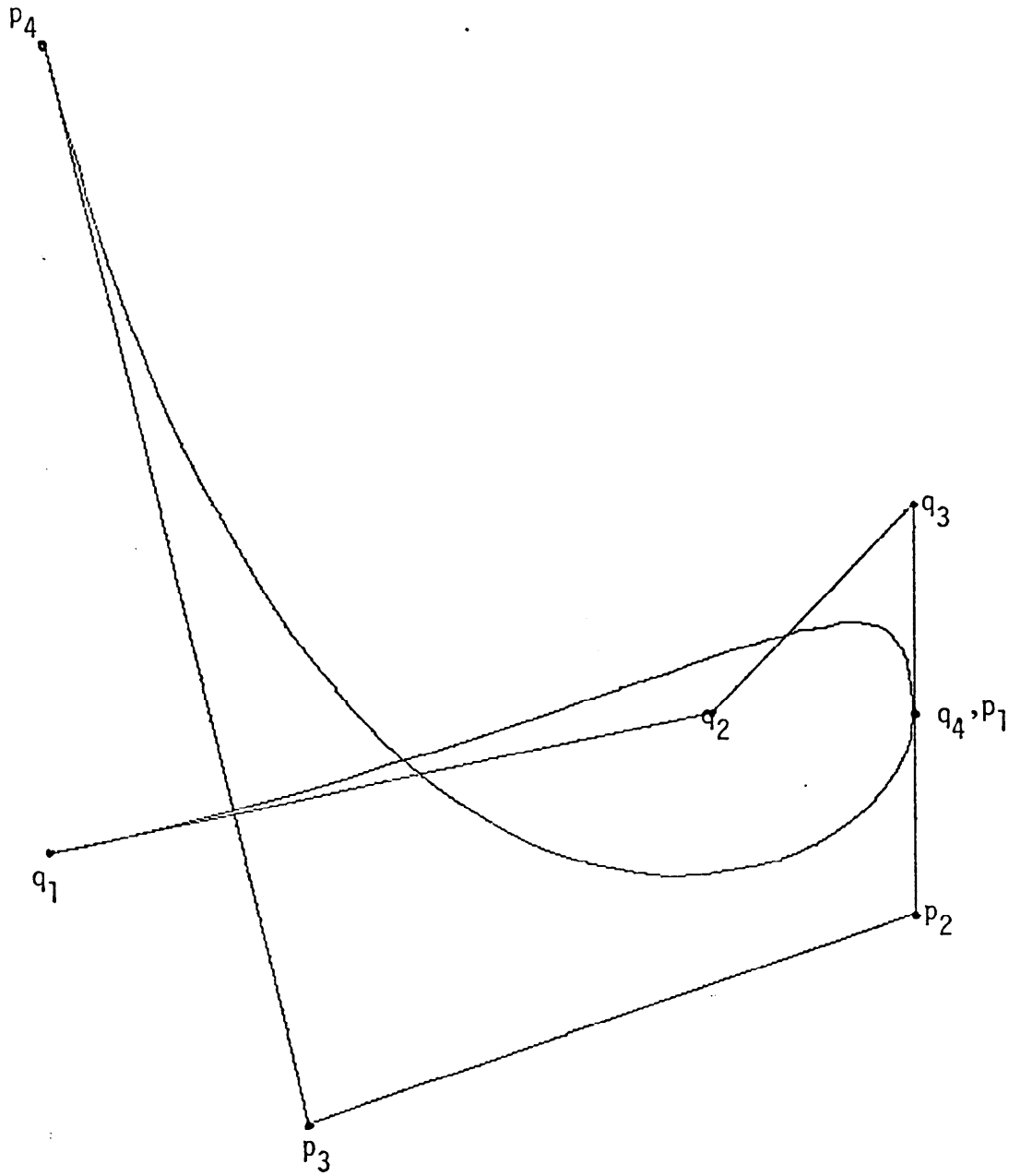


Figure 1-6 Two Bezier curves joined with tangent continuity, i.e. the line q_3, q_4 is colinear with the line p_1, p_2.

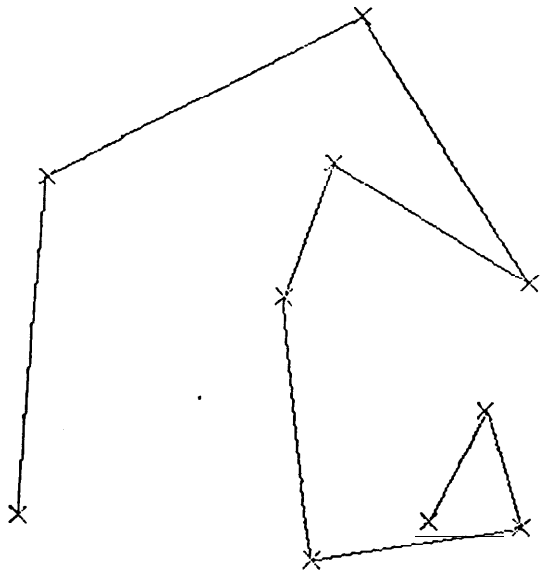


Figure 1-7a

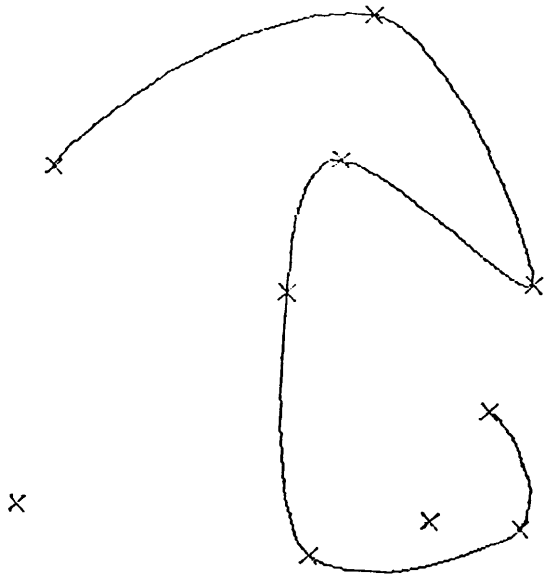


Figure 1-7b

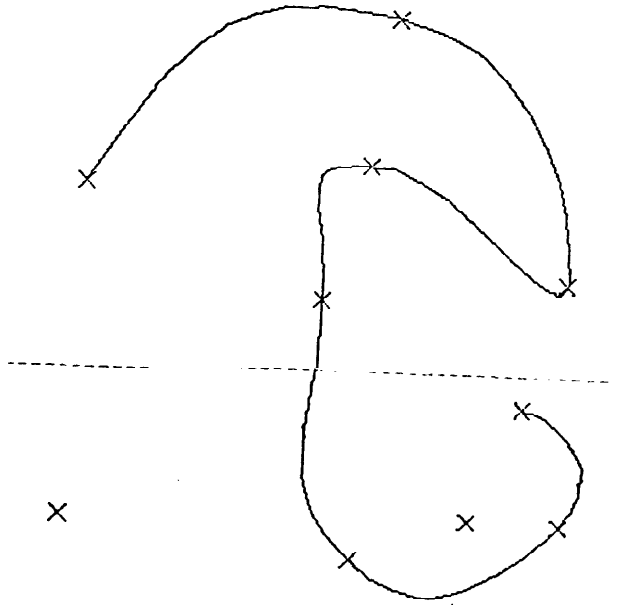


Figure 1-7c

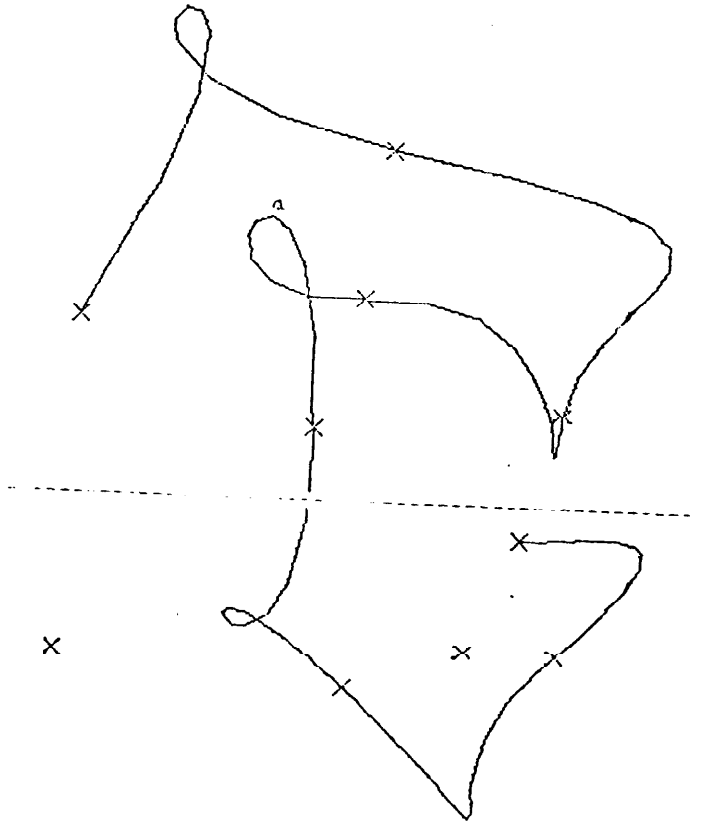


Figure 1-7d

Figure 1-7 These four illustrations demonstrate the dependance of the behaviour of the Cardinal spline on the coefficient a in equation 20. 1-7a is the original polygon to be fitted, 1-7b is with a=.5, 17c is with a=1.5 and 1-7d is with a=3.0 .

$$C_i(t) = [\, t^3 \;\; t^3 \;\; t \;\; 1 \,]\; M_h \begin{bmatrix} p_i \\ p_{i+1} \\ s_i \\ s_{i+1} \end{bmatrix} \qquad (17)$$

In order to obtain the Cardinal basis matrix, we choose the tangent vector at point p_i to be a constant multiple of the difference vector between the two adjacent points p_{i-1} and p_{i+1} :

$$s_i = a\,(\, p_{i+1} - p_{i-1} \,) .$$

where a is a positive scalar coefficient. Using this definition, the Hermite geometric information in equation 17 assumes the matrix form

$$\begin{bmatrix} p_i \\ p_{i+1} \\ s_i \\ s_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & 0 & a & 0 \\ 0 & -a & 0 & a \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix} \qquad (18)$$

$$= M_{hc}\; G_c \quad .$$

The column vector on the right of this equation, G_c, is the Cardinal geometric vector, and the matrix M_{hc} transforms it into the Hermite form of the left hand side of the equation. To obtain the cardinal basis matrix, M_c, substitute (18) into (17) obtaining

$$C_i(t) = [\, t^3 \;\; t^2 \;\; t \;\; 1 \,]\; M_c \;\; G_c \;, \qquad (19)$$

where $M_c = M_h \; M_{hc} =$

$$\begin{bmatrix} -a & 2-a & -2+a & a \\ 2a & -3+a & 3-2a & -a \\ -a & 0 & a & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad (20)$$

Fiqure l-8 shows the basis functions corresponding to this matrix, where a has been chosen equal to 1. Note that they sum to 1 like the Bezier basis functions. However, since several of them are negative, they do not have the convex hull property.

From equations 19 and 20 it is easy to verify that

$$C_i(0) = p_i \ ,$$
$$C_i(1) = p_{i+1} \ ,$$
$$C_i'(0) = a (p_{i+1} - p_{i-1}) \ ,$$
$$\text{and } C_i'(1) = a (Pi+2 - Pi) \ .$$

Since segment i at t=l joins with positional and tangent continuity to segment i+l at t=O, the curve is everywhere C^1 continuous.

Figure l-7 shows several curves generated using this cubic Cardinal spline basis. The dependence of the curve on the coefficient a is illustrated here. Figure l-7b shows the curve generated with a=. 5, l-7b is with a=l.5 and l-7d is with a=3.

Using this Cardinal representation it is therefore possible to generate a composite curve whose pieces are cubic segments. It should be clear that in order to understand the limiting behavior of the composite curve, we can transform each of the segments into their Bezier form and test the limits of the composite bounding box of this collection of control points. The matrix that transforms from Cardinal to Bezier form is

$$M_{bc} = M_{bh} M_{hc} \ ,$$

where M_{bh} is the inverse of M_{hb} in equation 12 and M_{hc} is from equation 18, giving

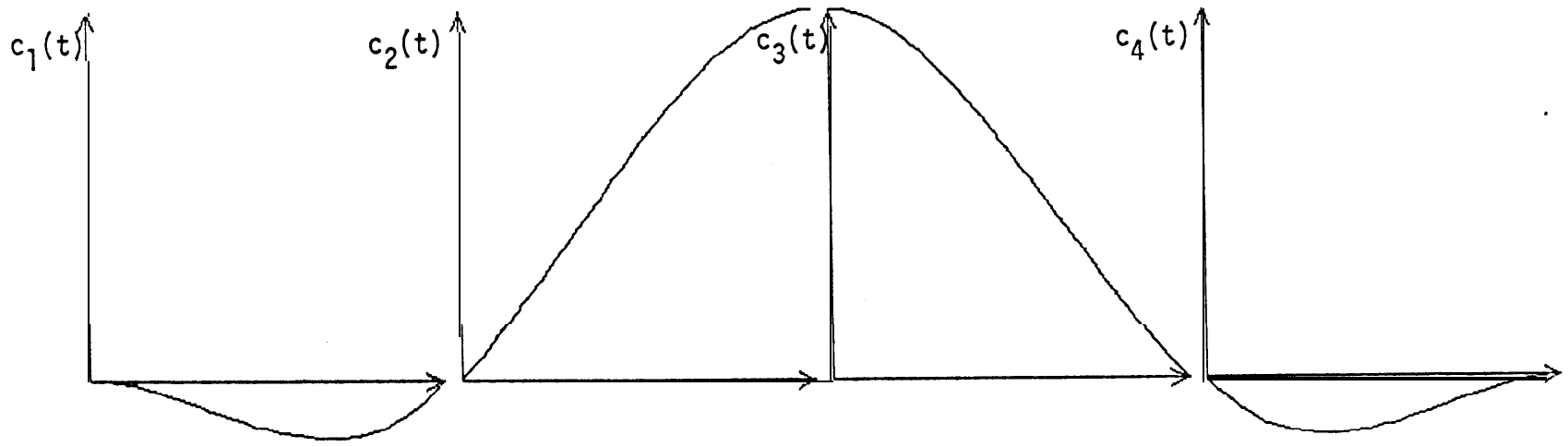


Figure 1-8 The Cardinal spline basis functions. The functions have been illustrated together in this way to demonstrate the continuity they share.

$$M_{bc} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -a/3 & 1 & a/3 & 0 \\ 0 & a/3 & 1 & -a/3 \\ 0 & 0 & 1 & 0 \end{bmatrix} . \qquad (21)$$

3.5 Cubic B-splines.

A spline formulation that has become popular in recent years is the B-spline. Mathematical treatments of these splines can be found in papers by Riesenfeld[3], Schoenberg[4], and de Boor[5]. These splines share with the Bezier formulation the convex hull property, which is a result of the total positivity of the B-spline basis. A more important reason for introducing these splines, however, is that the B-spline surface representation generalizes conveniently to non-rectangular topologies.

The details of the derivation of the cubic B-spline basis are not essential here. It is given by

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \qquad (22)$$

Similar to the Cardinal spline, segment i of a composite B-spline curve that approximates a sequence of points $p_1, p_2, \ldots, p_n$ in a vector space is

$$C_i(t) = T_3 \; M_b \; G_B^1 \; ,$$

where

$$G_B^i = \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix} ,$$

and $T_3 = [\, t^3 \;\; t^2 \;\; t \;\; 1 \,]$ is the primitive cubic basis.

Let us analyze the behavior of this cubic. Setting $t=0$, we see

$$C_i(0) = [0 \;\; 0 \;\; 0 \;\; 1\,]\; MB\; G_B^i \qquad (23)$$

$$= [\, 1 \;\; 4 \;\; 1 \;\; 0 \,]\; G_B^i$$

$$= (p_{i-1} + 'Pi + p_{i+1}) / 6 .$$

Therefore, a B-spline segment does not necessarily interpolate, or pass through, any of the control points at its t=O end. Rather, it starts at a point that is a weighted average of the first three control points defining the segment. The tangent and curvature of a B-spline segment at t=O are likewise given respectively by

$$C_i'(0) = (p_{i+1} - p_{i-1}) / 2, \qquad (24)$$

and

$$C_i''(0) = p_{i+1} - 2p_i + p_{i-1} \cdot \qquad (25)$$

The unique feature of the cubic B-spline is that in addition to maintaining position and tangent continuity, it also has curvature continuity, i.e.

$$C_i''(1) = C_{i+1}''(0).$$

This is possible with the B-spline, even though it is only a

cubic, because the segment is not required to pass through any of the control points. Other special features of the B-spline will be discussed in the chapters on surfaces.

Exercises.

3.5-1 Verify that cubic B-splines, as given by M_B, exhibit C^2 continuity.

3.5-2 Find the matrices M_{Bb}, M_{Bc} and M_{Bh}.

3.5-3 Prove that cubic B-splines satisfy the convex hull property.

4. Arbitrary Degree Curves.

All of the previous sections were concerned with curves of at most degree three. Since many computer graphics applications require no more than continuity of tangent, degree three curves are usually sufficient. However, for completeness, in this section we extend the Bezier formulation and show that the convex hull property holds for arbitrary degree. Although they are not presented here, each of the representations discussed in the previous sections have higher degree representations as well. Since transforming from one representation to another involves nothing more than matrix inversions and multiplications, it is a simple task to transform any representation into the Bezier form.

Referring to equations 15, we see that they can be expressed as

$$b_i(t) = \binom{3}{i} t^i \quad (1-t)^{3-i} \quad . \qquad (26)$$

These are the degree 3 binomial distribution functions. As one might expect, the degree n Bezier curve has the degree n binomial distribution functions as basis functions. That is,

$$C(t) = \sum_{i=0}^{n} Bn\,,i(t) \quad P_{i+1} \quad , \quad 0<t<1, \qquad (27)$$

where

$$B_{n,i}(t) = \binom{n}{i} t^i \quad (1-t)^{n-i} \quad . \qquad (28)$$

We will show in the next chapter that curves expressed using the arbitrary degree binomial basis functions make certain types of cumputations much simpler, e.g., finding the intersections of two curves in space and finding the zeros of a polynomial.

Exercises.

4-1 Show that the mth derivative $C^m(t)$ of a curve expressed in Bezier form at t=O is determined by its first m control points. By symmetry, at t=1, the last m control points determine $C^m(t)$.

4-2 Determine the placement of the first m control points of a curve C(t) in order that it is continuous to curve A(t) to mth derivative, i.e. $A^k(1) = C^k(0)$, k=O through m.

Summary.

Tn this chapter we have shown that parametric polynomial curves can be expressed in many different geometric formulations, all of which reduce to the same algebraic representation. The particular formulation one uses depends upon the application. If M_m is the algebraic coefficient matrix for a curve of degree m, it can be expressed in terms of a set of m+1 distinct vector conditions on the curve, that is, the geometric information matrix, by techniques analogous to those leading to equations 5, 7 and 9. Given any geometric representation, say e with basis matrix M_e, in order to transform this representation into another, say f with basis matrix M_f, the equivalence of the two expressions for the curve,

$$[\ t^m\ t^{m-1}\ .\ .\ .\ t\ 1\ ]\ M_e\ G_e = [\ t^m\ t^{m-1}\ .\ .\ .\ t\ 1\ ]\ M_f\ G_f\ ,$$

requires that

$$M_{fe} = M_f^{-1}\ M_e$$

be the matrix that transforms G_e into G_f, by premultiplication. It also transforms the basis matrix M_f into M_e by post multiplication.

The Bezier form is very convenient in many graphics/geometric-design contexts because of the convex hull property presented in the previous section. B-splines extend this property to spline curves. These properties combined with recursive algorithms make tractable many otherwise difficult problems, which will be discussed in the next chapter.

Chapter 2
Computational Techniques for Parametric Curves

This chapter is composed of two parts, both of which are presentations of techniques useful in making drawings of curves that are described in parametric form. The first part is concerned with subdivision methods. These subdivision methods are the basis for several non-deterministic algorithms for finding properties of curves, such as the points where they cross a line or plane. In Chapter 5, the subdivision algorithms will be shown for rendering shaded pictures of surfaces and finding the curves of intersection of arbitrary degree surfaces. The second part of the chapter presents forward difference techniques for curve generation.

1. Subdivision Methods

A number of useful algorithms for finding properties of curves and surfaces make use of subdivision of the original curve or surface. The basic premise in these algorithms is that one is interested in the behavior of a curve or surface only in the engineering sense, that is to within some tolerance. All of the algorithms that use this subdivision approach are non-deterministic in that they provide an answer to the problem to within a specified tolerance. At their core is an assumption that the curve whose properties are desired can be subdivided into two parts, each of which can be expressed in the same form as the original curve, i.e. with a parameter whose range is from 0 to 1.

Another essential ingredient of the algorithms is that they are recursive. The recursion arises by analysing whether a given curve itself satisfies the tolerance requirements; if it does, then the problem is solved to within the tolerance. If it does not, then it is split into its two halves, each of which is expressed in the exact same form as the original curve, and the algorithm is recursively applied to each of the two halves. Recursion terminates when the tolerance is satisfied. In this section we present the mathematics required to split a curve that is expressed relative to any of the bases given in the previous chapter.

1.1 Subdivision of Cubics.

Although subdivision methods exist for arbitrary degree curves, in this section we restrict ourselves to cubics. Section 1.3 presents subdivision techniques for arbitrary degree Bezier curves. The techniques presented here generalize in a straightforward way to arbitrary degree.

Consider a parametric cubic curve expressed using any of the bases discussed in the previous chapter:

$$C(t) = T(t)\ M\ G, \qquad (1)$$

where $T(t) = [\ t^3\ t^2\ t\ 1\ ]$, M is the basis matrix and G is the geometry matrix. We wish to express this curve in terms of two curves that together make up the original curve; one of them represents the half of the curve corresponding to $0 \le t \le 1/2$ and the other that corresponding to $1/2 \le t \le 1$.

It should be clear that each of the bases discussed in the previous chapter are symmetric in the sense that the same space curve will be generated if we replace t with 1-t. This simply

says that the ordering of the points in any of the expressions can be reversed and the same curve will be generated, with the exception that it will be parametrized in the opposite sense. Because of this, we present here the equations required for splitting to get the "first" half of the curve. We can get the equations for the second half of the curve by applying the equations obtained here to the control points in the opposite order.

Let us define the parameter for the first half of the curve C(t) above as t_1 such that $t_1/2=t$. This parameter, t_1, varies between 0 and 1 as t varies between 0 and 1/2, which is desired if this subcurve of C(t) is to have the same form as the original curve. Substituting this into equation 1, we get for the first half of the curve, which we call $C_1(t_1)$,

$$C_1(t_1) =$$

$$C(t_1/2) = T(t_1/2)\ M\ G$$

$$= [\ t_1^3/8 \quad t_1^2/4 \quad t_1/2 \quad 1\ ]\ M\ G$$

$$= [\ t_1^3 \ t_1^2 \ t_1 \ 1\ ] \begin{bmatrix} 1/8 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M\ G \quad (2)$$

$$= T(t_1)\ S\ M\ G,$$

where S is the simple diagonal matrix shown. Now as t_1 varies between 0 and 1, the first half of the curve is generated by this expression. Since this curve representing the first half must also must be a cubic, it must also have an expression like that in equation (1), that is

$$C_1(t_1) = T(t_1)\ M\ G_1, \qquad (3)$$

where G_1 represents the geometry matrix for this half of the curve. Setting (2) equal to (3), and requiring that the equation hold for arbitrary values of t_1, we get

$$M\ G_1 = S\ M\ G$$

or

$$\begin{aligned} G_1 &= [\ M^{-1}\ S\ M\ ]\ G \\ &= H_1\ G. \end{aligned} \qquad (4)$$

That is, the geometry matrix for the first half of the curve, G_1, can be obtained from the geometry matrix for the original curve, G, by premultiplyng G by H_1, which is computed as the product indicated. We will refer to H_1 as the splitting matrix for the first half of the curve. Each of the bases given in the previous chapter has its own splitting matrix.

Exercises.

2-1. Find the splitting matrices for each of the bases given in chapter 1.

2-2. Derive the splitting matrix for the second half of the curve, that is H_2.

1.2 Subdivision of Bezier Cubics.

In the previous chapter we showed that curves do not go outside of the convex hull of their Bezier control points. Curve representations that have this property are especially well suited for the recursive algorithms presented in Chapters 4 and 5. Since a curve behaves no worse than its Bezier control points, we can restrict ourselves at each level of a subdividing recursive algorithm to determining the behavior of the control points. Therefore, aside from specifying what we mean by the "behavior" of the control points, which depends upon the particular problem at hand, we must at each stage of the recursion determine the new control points for the two subcurves. This arithmetic takes an especially simple form in the Bezier representation.

Applying equation (4) using the Bezier basis, we find the splitting matrix for the first half to be

$$H_1 = \begin{vmatrix} 1 & 0 & 0 & 0' \\ 1/2 & 1/2 & 0 & 0 \\ 1/4 & 2/4 & 1/4 & 0 \\ 1/8 & 3/8 & 3/8 & 1/8 \end{vmatrix} .$$

By the symmetry argument given above, the second half splitting matrix is given by

$$H_2 = \begin{vmatrix} 1/8 & 3/8 & 3/8 & 1/8 \\ 0 & 1/4 & 2/4 & 1/4 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{vmatrix} .$$

If we label the control points of the sub-curve $C_1(t_1)$ by ri and those Of $C_2(t_2)$ by Si, it is easy to show from the splitting matrices that they are given by the relation

$$r_{i+1} = q_0^i \qquad (5)$$

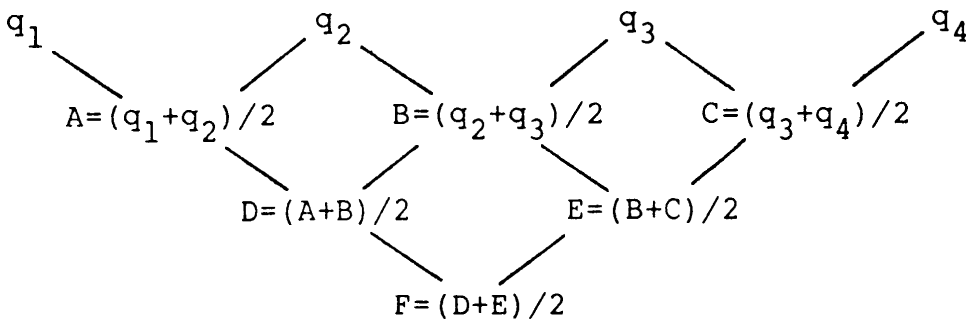
and $s_{i+1} = q_i^{n-1}$,

where q_m^n is given by the recurrence relation

$$q_m^{n+1} = (q_m^n + q_{m+1}^n) / 2 , \qquad (6)$$

and $q_j^0 = q_{j+1}$.

The following diagram illustrates the computation implied by these equations. Assuming that the original curve C(t) is given in terms of the original control points q_i, we obtain the new sets of control points ri and s_i by



with $r_1 = q_1$,
$r_2 = A$,
$r_3 = D$,
$r_4 = F$,
$s_1 = F$,
$s_2 = E$,
$s_3 = C$,
and $s_4 = q_4$.

In other words, the control points of the two subcurves are obtained from the parent curve's control points by six adds and six divides by 2, times the number of components in the curve.

This is a very modest amount of work to split a curve into two subcurves. Figure 2-1 illustrates the two subcurves' control polygons obtained by applying this splitting process to the original control polygon. If this splitting process is carried out repetitively, the control polygons converge to the cubic curve.

It should be clear from this discussion that a curve in Bezier form can behave no worse than its control polygon. For example, a curve has no point of inflection unless its Bezier control polygon does. This property is called the variation diminshing property[3].

The convex hull property mentioned in the previous chapter and this subcurve property provide a powerful computational tool when combined with recursive algorithms. Consider as an example the task of determining, to within a tolerence E, the intersection points of two cubic curves. Since the minimum bounding volumes that surround the control points totally enclose the convex hull, which in turn encloses the curve, all one need consider at any time is the bounding boxes of the two sets of control points. The curves cannot go outside of their respective boxes. Initially, test the bounding boxes of the two sets of control points for overlap. If they do overlap, consider the curve having the larger bounding box. If its size is less than E, emit the coordinates of one of its points as an intersection point, and return. If its size is greater than E, subdivide the control points according to the above rules, forming two sub-control polygons. Recursively apply this algorithm, comparing each of these two sub-polygons in turn to the smaller of the two original curves. If they do not overlap at any stage of the recursion, they cannot intersect.

The following Algol procedure gives the details of the algorithm. Note that the routine makes use of the routines SIZE, SPLIT and OVERLAP, which perform obvious functions.

```
procedure INTERSECT( A, B, E );
    real array A, B[1:4,1:3];
    real E;
begin
    real array X, Y[1:4,1:3];
if OVERLAP( A, B ) then
    begin
    if SIZE(A)>SIZE(B) then
        begin
        if SIZE(A)>E then
            begin
            SPLIT( X, Y, A );
            INTERSECT( X, B, E );
            INTERSECT( Y, B, E );
            end
        else
            OUTPUT( A )
        end
    else
        if SIZE(B)>E then
            begin
            SPLIT( X, Y, B );
            INTERSECT( A, X, E );
            INTERSECT( A, Y, E );
            end
        else
            OUTPUT( B );
    end
end INTERSECT;
```

This simple example routine illustrates the simplicity of performing certain types of computation in the Bezier form if recursive subdivision techniques are used. Other types of

computation that become simple in the Bezier form are such things as surface-surface intersections (of any degree) and hidden surface removal. The importance of these features of the Bezier formulation will become more evident in the next section and in the next chapter.

Exercises.

2-3. Show that the recurrence relations given in (5) and (6) are consistent with the results obtained from applying the Bezier splitting matrices Hl and H_2.

1.3 Arbitrary Degree Curve Subdivision.

Although we could follow the same type of procedure used in section 1.1 to obtain the splitting rules for arbitrary degree bases, it is more convenient to derive the rules for arbitary degree Bezier curves in a slightly different way. The reason for chosing the Bezier form is again because of the properties of the Binomial distribution functions, which determine the behavior of the Bezier curve.

Consider the convex hull property. In order that a polynomial have it, as we mentioned before, the basis functions in terms of which it is expressed must both sum to 1 and be bounded in value between 0 and 1 for all values of t between 0 and 1. The binomial distribution functions obviously satisfy these two criteria. (Expand (t + (l-t))^n using the binomial theorem to prove that they sum to 1.)

The subcurve property also follows from the properties of the these functions. The essence of the subcurve behavior is that any polynomial C(t) expressed in terms of these functions can be

represented as

$$C(t) = C_1(t_1),$$

where

$$t = t_1/2 \quad , \quad 0 \le t_1 \le 1, \text{ for the first half}$$

and $C(t) = C_2(t_2)$,

where

$$t = (1+t_2)/2 \quad , 0 \le t_2 \le 1 \text{ for the second half.}$$

To see what relation gives the control points of $C_1(t_1)$ and $C_2(t_2)$ in terms of the control points of C(t), observe that

$$B_{n,i}(s/2) = \binom{n}{i} s^i \, 2^{-n} \, (2-s)^{n-i}$$

$$= \binom{n}{i} s^i \, 2^{-n} \, (s + 2(1-s))^{n-i} \quad ,$$

or

$$B_{n,i}(s/2) = \sum_{k=0}^{n} 2^{-k} \binom{k}{i} B_{n,k}(s) \qquad (7)$$

using the binomial theorem to expand the quantity in parentheses. Also,

$$B_{n,i}((1+s)/2) = B_{n,i}(1 - (1-s)/2)$$

$$= B_{n,n-i}((1-s)/2)$$

$$= \sum_{k=0}^{n} 2^{-k} \binom{k}{n-i} B_{n,k}(1-s) \text{ from (7)},$$

giving

$$B_{n,i}((1+s)/2) = \sum_{k=0}^{n} 2^{-k} \binom{k}{n-i} B_{n,n-k}(s), \qquad (8)$$

since

$$B_{n,n-k}(1-s) = B_{n,k}(s).$$

Therefore

$$C(t) = C(t_1/2) = \sum_{i=0}^{n} B_{n,i}(t_1/2)\ p_{i+1}$$

$$= \sum_{i=0}^{n} \sum_{k=0}^{n} 2^{-k} \binom{k}{i} B_{n,k}(t_1)\ p_{i+1}$$

$$= \sum_{k=0}^{n} B_{n,k}(t_1) \left\{ \sum_{i=0}^{n} \binom{k}{i} p_{i+1}/2^k \right\} \qquad (9)$$

$$= \sum_{k=0}^{n} B_{n,k}(t_1)\ r_{k+1}\ .$$

In other words, the curve C(t) expressed in terms of the parameter $t_1 = 2t$ has control points ri given by the expression in brackets. This expression has the recursive definition given in equations 5 and 6. A similar argument using equation 8 shows that the control points s_i are also given by these two equations.

Therefore, the control point groups of the two subcurves of degree n, ri for the t_1 subcurve and s_i for the t_2 subcurve, are obtained from the control points of the parent curve of degree n by a total of

$$n + n-1 + n-2 + \ldots +1 = n(n+1)/2$$

adds and divides by 2, times the number of components.

Interesting applications of these properties exist in areas other than computer graphics and geometric design. For example, a very simple recursive algorithm for finding the zeros of an

arbitrary degree polynomial consists in transforming the polynomial into Binomial form and recursively subdividing the curve until both the bounds of the control points are less than a specified tolerance and two or more of the control points have opposite signs. Contrast this simple approach with the traditional iterative Newton's method in which one must have a suitably accurate starting approximation to even find a zero. Moreover, using these techniques, also in contrast to the traditional approach, one can establish an upper bound on the number of steps required to accomplish the computation.

Exercises.

2-4. Show that Pascal's triangle is determined by a recurrence relation similar to that of equations 5 and 6.

2-5. Show that the (n+1) by (n+1) matrix M analogous to equation 14 of the previous chapter, that transforms the nth degree primitive basis $[\, t^n \; t^{n-1} \; \ldots \; 1 \,]$ into the nth degree Binomial (Bezier) basis has elements

$$M_{j+1,i+1} = \binom{n-j}{i} \binom{n}{n-j} (-1)^{i+n-j}$$

2-6. Show that the Binomial distribution functions are defined by the recurrence relation

$$B_{n+1,i}(s) = s\, B_{n,i-1}(s) + (1-s)\, B_{n,i}(s)$$

where $B_{0,n}(s) = \delta_{0,n}$,
and $\delta_{0,n}$ is the Kroeneker delta function.

2-7. Show that a Bezier curve can be defined recursively by

$$p_m^{n+1}(s) = s\, p_m^n(s) + (1-s)\, p_{m+1}^n(s) \;,$$

where $p_i^0(s)$ are the original control points and $p_0^n(s)$ is the generated curve. Use this fact to geometrically construct the point on a curve where s equals a particular value.

2-8. Write an ALGOL procedure that will find the zeros of an arbitrary degree polynomial expressed in Bezier form.

2. Forward Difference Equations for Curve Generation.

This section presents a method for drawing curves that is based on the calculus of finite differences. The advantage to using the techniques presented here is that once the initial arithmetic is done to obtain the difference equations, the actual generation of the curve requires only a small number of additions. Since the methods for generating the difference matrix are very straightforward, we will develop the matrix for cubics only.

Each of the cubic representations in the previous chapter reduce to a common algebraic representation by multiplying the basis matrix by the geometry matrix to get the algebraic coefficient matrix for cubics:

$$A = M_q \; G_q \; . \qquad (10)$$

Therefore, we will assume that this has been done, giving for the cubic curve

$$C(t) = [\, t^3 \;\; t^2 \;\; t \;\; 1 \,]\, A.$$

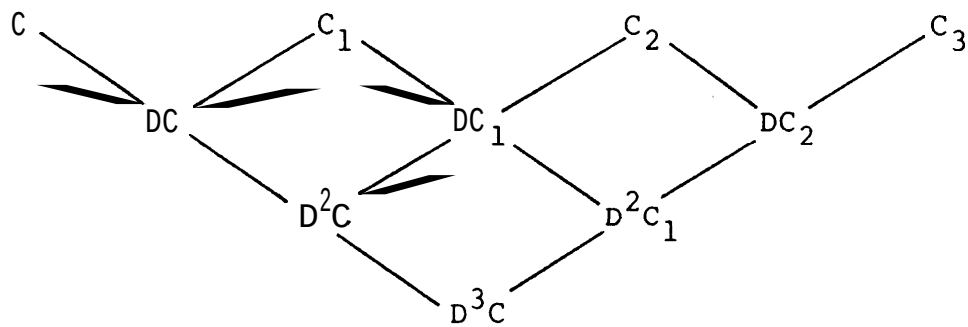
The number of spatial dimensions represented by this curve is the number of columns of A and does not alter the following development.

Suppose we wish to render a curve with a total of n straight 1 ine segments. Let us define e=1/n to be the change in the parameter t between successive positions along the curve at which the curve is to be evaluated. These will be the endpoints of the straight line segments used to represent the curve. The first step in generating the difference equations for a degree m curve is to evaluate the curve at e, 2e, 3e,...I me. This gives the zeroeth order differences. The first order differences are then computed as the differences between adjacent zeroeth order differences. Clearly, there will be only m of these values, rather than m+1, as with the zeroeth order differences. In general, the qth order values are computed as the differences between the m-q+1 values of order q-l.

Let us carry out this process for cubics. Defining the values

$$\begin{aligned} C(0) &= C, \\ C(e) &= C_1, \\ C(2e) &= C_2, \\ C(3e) &= C_3, \end{aligned} \qquad (11)$$

the differences are computed as in the following table:



The curve can be generated using only the D^nC's, that is, those along the left hand edge of this triangle. C represents the starting point of the curve. DC is the value that when added to C

will give the second point on the curve; it is the increment for C.

It is convenient to express these difference equations in a matrix form. It is easy to show that the following values result from the construction method given above:

$$C = [0 \quad 0 \quad 0 \quad 1]\ A,$$
$$DC = [e^3 \quad e^2 \quad e \quad 0]\ A,$$
$$D^2C = [6e^3 \quad 2e^2 \quad 0 \quad 0]\ A,$$
$$\text{and } D^3C = [6e^3 \quad 0 \quad 0 \quad 0]\ A.$$

Combining these equations into one matrix equation, we obtain

$$\begin{bmatrix} C \\ DC \\ D^2C \\ D^3C \end{bmatrix} = E(e)\ A, \qquad (12)$$

for the difference matrix, where

$$E = \begin{bmatrix} 0 & 0 & 0 & 1 \\ e^3 & e^2 & e & 0 \\ 6e^3 & 2e^2 & 0 & 0 \\ 6e^3 & 0 & 0 & 0 \end{bmatrix}.$$

The product indicated in equation (12) is therefore the matrix required to generate the curve. Assuming the same step size will always be used, the E matrix is computed once and used for each curve to be generated.

We see from the above development that the product

$$D = E(e)\ M\ G \qquad (13)$$

gives the difference matrix for the curve with geometry matrix G, using basis matrix M. Clearly, if the curve is to also undergo some viewing transformation before actually being drawn, then it is computationally more efficient to first transform G, which has only four vector components, and then bypass the viewing transformation mechanism when the curve is being generated. In other words, assuming vm is the viewing matrix, the difference matrix

$$D = E(e)\ M\ G\ vm$$

can be used to generate points on the curve in the transformed space defined by vm. However, since normal viewing transformations are 4X4, D has 4 components, and thus 4 components must be iterated (x, y, z and w).

The following algorithm describes the process involved in drawing a curve with nseg segments having geometry matrix G_q and basis matrix M_q.

Preliminary steps:

A. Form the difference matrix D of equation (13).

B. Transform the x, y and z components of all elements of D by the viewing matrix being used to view the curve. If the viewing transformation is 4X4, D now has 4 components. Use this transformed D in the algorithm. Bypass all viewing transformations in the algorithm while drawing the segments of the curve.

Algorithm DRCURVE(nseg, D):

1. Copy D into working registers R (to avoid destroying D, which was passed by reference.)

2. MOVE to position in first element of R.
(x,y,z,w position.)

3. Repeat steps 3.1 and 3.2 nseg times.

3.1 Add second element of R(all components) to first element; add third element to second; add fourth element to third.

3.2 DRAW to position in first element of R.

Exercises.

2-9. Prove that the nth order differences of a function C(t) are given by the expression

$$D^n C = \sum_{i=0}^{n} (-1)^{n+i} \binom{n}{i} C(ie) .$$

Use this to write down the matrix for degree 5 curves.

Chapter 3
Parametric Surfaces and Volumes

In this chapter we extend the methods of the previous chapter to include surfaces. In addition, formulations for reresenting volumes are presented, although the main emphasis is on the more commonly used surface representations such as Coons and Bezier patches. By analogy with equation 1-1, the surface is represented as a vector bivariate function of two parameter variables, t and u, each of which ranges between 0 and 1:

$$S(t,u) = [\, x(t,u)\ y(t,u)\ z(t,u)\ 1 \,] \qquad 0 \le t \le 1, \quad 0 \le u \le 1 \qquad (1)$$

Since we are concerned primarily with representations in physical space, we restrict ourselves to three dimensions, although similar equations hold for higher dimensions. Also, since the parameter ranges are restricted to lie between 0 and 1, we are concerned with just that segment (patch) of the surface corresponding to this range.

Bivariate representations can be obtained from univariate representations in several different ways. One of the more common methods employs the Tensor Product theorem from linear algebra[6]. Another method employs a form of surface known as the Boolean Sum surface[6]. In this chapter we will consider primarily the Tensor Product form, since it is the common way of expressing the general Coons patch, Bezier patches and bivariate splines. However, rather than simply stating this theorem as the basis for extending to two variables, we will instead develop a method for extending the curve formulations of the previous chapter to surfaces. Using this approach, both the Tensor Product

and Boolean Sum representations become more easily understood.

In Section 1, the bilinear form, or the bivariate extension to the straight line, is developed. In the first part of Section 2, the bicubic Coons patch as an extension of the Hermite form for bicubic curves is presented. Then the Bezier, or Binomial, cubic is extended to the bicubic representation. In section 2.2 the Bezier formulation is compared to the Coons form, and it is shown that the Bezier approach provides a very easy method for manipulation of the "twist" partition. After brief discussions of spline surfaces and mixed representations, we then give in Section 3 the bivariate form for arbitrary degree surfaces in Binomial form and show that the convex hull and subcurve property given in the previous chapters have analogous expressions for surfaces. Volumes, or trivariate representations, are then discussed as extensions to the surface representations. Section 5 then presents a mixture of the Bezier form with the Hermite form and indicates ways for mixing various other formulations.

1. Bilinear forms.

The main step in extending the straight line parametric form to two variables is realizing that in equation 5 of Chapter 1, there is no need to restrict the two endpoints of the Vine" to be constants. That is, one can just as simply let them be (arbitrary) functions of another parameter u. In particular, suppose we let them be functions of u of the same form as Equation 1-5. In that case, we obtain

$$V(t) = \begin{bmatrix} t & 1 \end{bmatrix} M \begin{bmatrix} p_1(u) \\ p_2(u) \end{bmatrix} , \quad (2)$$

where

$$p_1(u) = \begin{bmatrix} u & 1 \end{bmatrix} M \begin{vmatrix} p_{1,1} \\ p_{1,2} \end{vmatrix} , \qquad (3)$$

and

$$p_2(u) = \begin{bmatrix} u & 1 \end{bmatrix} M \begin{bmatrix} p_{2,1} \\ p_{2,2} \end{bmatrix} . \qquad (4)$$

The coefficients $p_{i,j}$ in equations 3 and 4 are now assumed to be constants (although the trilinear form is obtained by letting them be similar functions of another parameter v). Since V(t) as expressed in equation 2 implicitly depends upon the parameter u, it is best to explicitly indicate this by expressing it as V(t) = C(t,u). A small amount of matrix algebra allows us to express these three equations as

$$V(t) = C(t,u) = \begin{bmatrix} t & 1 \end{bmatrix} M \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} M^t \begin{bmatrix} u \\ 1 \end{bmatrix} . \qquad (5)$$

This is the Tensor Product bilinear form of equation 5 expressed conveniently as a matrix product. Chosing any particular value of u and varying t yields a straight line. By symmetry, chosing some value for t and varying u also yields a straight line. That is, the form of equation 5 allows us to interpret it in the form

$$C(t,u) = \begin{bmatrix} p_1(t) & p_2(t) \end{bmatrix} M^t \begin{bmatrix} u \\ 1 \end{bmatrix} ,$$

where

$$p_1(t) = \begin{bmatrix} t & 1 \end{bmatrix} M \begin{bmatrix} p_{11} \\ p_{21} \end{bmatrix} ,$$

and

$$p_2(t) = \begin{bmatrix} t & 1 \end{bmatrix} M \begin{bmatrix} p_{12} \\ p_{22} \end{bmatrix} .$$

Notice that the matrix of end, or corner, points in equation 5 is a matrix of vectors, i.e. a 2 by 2 by n tensor, where n is the number of components of the points $p_{i,j}$ and therefore also of C(t,u). This matrix can be visualized as a volume of scalars with the spacial components extending into the page, that is, with the x component layer in the page, the y component layer just underneath it, etc.

2. Bicubics.

Each of the cubic representations of the previous chapter has a bicubic representation, and all of them are widely used in graphics and geometric design. The bicubic hermite and bicubic Bezier forms will be presented in detail, followed by a presentation of a combination of the two. The methods used should provide a reasonable foundation for the reader to develop the representation that is most appropriate for the application at hand.

2.1 Hermite Surface Patches

The bicubic Hermite surface is developed just as in equation 5. We allow the components of the G_h vector of equation l-9 to be functions of another parameter u. That is,

$$G_h = G_h(u) = \begin{bmatrix} p_1(u) \\ p_2(u) \\ s_1(u) \\ s_2(u) \end{bmatrix}. \qquad (6)$$

It is worth noting that they can be arbitrary functions of u. For the particular case in which we let each of them be cubic func-

tions of u of the form of equation 1-9, we obtain the bicubic Tensor Product form. Let

$$[\ p_1(u)\ p_2(u)\ s_1(u)\ s_2(u)\ ] =$$

$$[\ u^3\ u^2\ u\ 1\ ]\ M_h \begin{bmatrix} q_{11} & q_{21} & q_{31} & q_{41} \\ q_{12} & q_{22} & q_{32} & q_{42} \\ q_{13} & q_{23} & q_{33} & q_{43} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{bmatrix} \qquad (7)$$

In other words, the first two columns of the above matrix describe as cubic functions of u the curves $p_1(u)$ and $p_2(u)$, which represent the end-"points" of the cubic in equation l-9. Similarly, the third and fourth columns of this matrix describe the end-"tangents" of the cubic in l-9 as cubic functions of u. Transposing this equation and substituting it for $G_h(u)$ in l-9, we get

$$C(t) = S(t,u) =$$

$$[\ t^3\ t^2\ t\ 1\ ]\ M_h\ G_{hh}\ M_h{}^t \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix} \qquad (8)$$

where G_{hh} is the 4 by 4 matrix of geometry information describing the patch, i.e. the transpose of the matrix in (7). This is the Tensor product bicubic form of the Coons patch. Inspection of equation 7 reveals the meaning of each of the elements of the G_{hh} matrix. The point q_{11} is the u=0 end of the $p_1(u)$ curve and hence is the t=0, u=0 corner of the surface patch of (8). From (7), the vector q_{13} is the tangent to the curve $p_1(u)$ at the u=0 endpoint and hence is the derivative of S(t,u) with respect to u at t=0 and u=0. Similarly,

$$s^t(t,u) = q_{31}$$

and $$s^{tu}(t,u) = q_{33} ,$$

where the superscripts refer to differentiation with respect to the corresponding variable.

The other four corners of the patch have similar expressions. Partitioning the G_{hh} tensor into four 2 by 2 partitions

$$G_{hh} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] , \qquad (9)$$

the A partition contains the four corners of the patch, the B partition is the derivative of A with respect to u, the C partition is the derivative of A with respect to t and the D partition is the cross partial derivative of A with respect to both t and u, commonly known as the twist partition. Figure 3-l shows a surface patch of this form.

2.2 Bicubic Bezier Surfaces.

The Bezier Tensor product form is obtained in exactly the same way in which the Hermite form was obtained: the geometry vector of Equation l-13 is expressed as a function of another parameter u. In other words, the control "points" of l-13 are instead control “curves". Again, each of them can be arbitrary functions. Letting them be cubic functions of u of the form of equation 1-13, $G_b(u)$ can then be expressed as the transpose of

$$[\ p_1(u)\ p_2(u)\ p_3(u)\ p_4(u)\ I = \begin{bmatrix} p_{11} & p_{21} & p_{31} & p_{41} \\ p_{12} & p22 & p_{32} & p_{42} \\ p_{13} & p_{23} & p_{33} & p_{43} \\ p_{14} & p_{24} & p34 & p44 \end{bmatrix} \quad (10)$$

The points p_{11}, p_{12}, p_{13} and p_{14} are the control points of point(curve) $p_1(u)$ in equation 13, chapter 1. Transposing this and putting the result in Equation l-13, we obtain

$$C(t) = S(t,u) =$$

$$= [\ t^3\ t^2\ t\ \ 1\ ]\ M_b\ \ G_{bb}\ \ M_b{}^t \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}, \qquad (11)$$

where G_{bb} is the transpose of the coefficient tensor given in equation 10. This can also be expressed in summation form by carrying out the matrix multiplications:

$$S(t,u) = \sum_{i,j=0}^{3} b_i(t)\, b_j(u)\ p_{i+1,j+1},$$

where $b_i(t)$ and $b_j(u)$ are the degree 3 binomial distribution functions given in Chapter 1.

As with the bicubic Hermite form, we arrive at a matrix of geometry vectors. However, in contrast to the Hermite form, this matrix is only geometric point information and does not explicitly include terms for tangents and cross partial derivatives. This matrix is referred to as the control point array. From (9) and the properties of the Binomial basis it is clear that $3(p_{12}-p_{11})$ is the tangent to $p_1(u)$ at u=0 and hence is the tangent to S(t,u) in the u direction at t=0 and u=0 (cf. equation 1-11). Since the

tangent to the curve of equation l-13 at t=O is completely determined by p_1 and p_2, the tangent to the surface of (11) in the t direction at t=0 is completely determined by $p_1(u)$ and $p_2(u)$, or, since these two functions are determined by the first two columns of the matrix of (10), by the first two rows of the geometric matrix, G_{bb}, of equation 11. Symmetry in the binomial basis functions dictates that the third and fourth rows of G_{bb} determine the tangent to S(t,u) in the t direction at t=1. Also by symmetry, similar arguments hold for the u direction, i.e. the first two columns of G_{bb} determine the tangent to S(t,u) in the u direction at u=0, etc. Figure 3-2 shows the control point array for the patch in Figure 3-l.

It is interesting to compare this representation to the Hermite representation of the previous section. Since they are both cubic representaions, we can determine the transformation that expresses one in terms of the other by equating equations 8 and 11:

$$M_h \; G_{hh} \; M_h^t = M_b \; G_{bb} \; M_b^t$$

from which we get

$$G_{hh} = M_{hb} \; G_{bb} \; M_{hb}^t \quad , \qquad (12)$$

where M_{hb} is given by equation l-12. This expresses the Hermite geometry matrix for the surface in terms of the Bezier control points. That is, given a Bezier control point array, the Hermite coefficient matrix that produces the same surface is given by performing this transformation.

Let us consider just that portion of G_{hh} corresponding to the t=O and u=0 corner of the patch, i. e. the 1,1 elements of the four partitions of equation 9. After carrying out the matrix

arithmetic implied by equation 12, we find

$$
\begin{aligned}
A_{11} &= p_{11} , \\
B_{11} &= 3 (P12 - p_{11}) , \\
C_{11} &= 3 (p_{21} - p_{11}) , \\
D_{11} &= 3 [3(P22 - p_{12}) - 3(p_{21} - p_{11})] .
\end{aligned}
$$

We see that the point p_{22}, which is the interior control point of the Bezier form that affects the t=O, u=0 corner, occurs only in D_{11}. Since D is the twist partition of the Hermite form, and D_{11} is the "twist vector" for the t=O,u=O corner, it should be clear that the p_{22} point in the Bezier form affects only the twist at this corner. Therefore, this point provides a convenient geometric means for manipulating the "twist vector" at this corner. By symmetry, the points p_{23}, p_{32} and p_{33} affect only the twist vectors at their respective corners of the patch.

The properties that make the Bezier form extremely convenient in computational algorithms for curves also hold for surfaces. This means that the bounding volume of the patch's control points bounds the patch as well and that we can split a patch into two subpatches, i. e. express the control points of the subpatches in terms of the control points of their parent patch, by a very small amount of arithmetic. Recursive algorithms based on these properties make a number of difficult practical problems very simple. For example, an algorithm is described in Chapter 5 that finds, to within a specified tolerance, the boundary of intersection of two bivariate patches. Because the properties of curves expressed in Binomial form hold for arbitrary degree, the

algorithm holds for arbitrary degree.

2.3 Patch Continuity and Splines.

One of the principle reasons for using higher degree curves and surfaces is that they make it possible to join curve segments and surface patches with continuity of tangent as well as position. This section will first indicate how Bezier patches must be manipulated in order to obtain continuity between patches. Then the bivariate form of cubic Cardinal splines and B-splines will be presented.

Bezier curves are conceptually simpler to geometrically modify to obtain continuity than are Hermite curves. The reason for this is that they are represented in terms of geometric position information. Compared with Hermite patches, the conveniences of the Bezier form are amplified by the fact that in the Hermite form, the twist partition is not particularly easy to conceptualize.

Suppose we have four patches that join together with position and tangent continuity as shown in Figure 3-3. Now suppose we wish to be able to modify any of them, say by moving a Bezier control point, in such a way that the original continuity is maintained. Depending upon which of the points is moved, various points must be moved in patches adjacent to the one containing the original displaced point. Figure 3-4 illustrates that a total of 8 additional points must be moved through the same displacement when we move the corner point shared by all of the patches. This follows from the properties of the Bezier form. Figures 3-5 through 3-8 illustrate several other sets of displacements.

A set of high level subroutines has been implemented that allows manipulations such as these on bicubic surface patches. The routines are implemented in Fortran on a PDP-11/E&S Picture System. They are organized to provide enough flexibility to allow a very simple driving program to treat surface patches as low level primitives that can be edited and manipulated in much the same way that the program can manipulate and display ordinary geometric entities such as points and lines. These routines are described in the document of the Appendix.

Just as with curves, bivariate cubic spline surfaces can "automatically" be generated with C^1 or C^2 (B-spline) continuity. Since there are many different spline formulations and we do not try to be comprehensive in our discussion of them, we present only the bivariate extension of the Cardinal spline and B-spline described in the previous chapter.

Analogous to the sequence of points through which a composite curve passes in such a way that it is C^1 continuous, we have a rectangular array of points through which a composite surface is to pass such that it is everywhere C^1 continuous. By analogy with the previous developments of the Tensor product forms for Bezier and Hermite representations, the i,j segment, or patch, of the composite surface is given by the equation

$$S_{i,j}(t,u) =$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M \quad G^{i,j} \quad M^t \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}, \qquad (13)$$

where $G^{i,j} = \begin{bmatrix} p_{i,j} & p_{i,j+1} & p_{i,j+2} & p_{i,j+3} \\ p_{i+1,j} & p_{i+1,j+1} & p_{i+1,j+2} & p_{i+1,j+3} \\ p_{i+2,j} & p_{i+2,j+1} & p_{i+2,j+2} & p_{i+2,j+3} \\ p_{i+3,j} & p_{i+3,j+1} & p_{i+3,j+2} & p_{i+3,j+3} \end{bmatrix}$

is the geometric control point of the i,j patch for either the B-spline or the Cardinal spline, and M is the corresponding basis matrix. The properties of the spline curves imply that the composite surface has the required continuity, since it is nothing more than a spline curve of spline curves, in a manner of speaking. This can also easily be verified directly from (13). In the B-spline version, the surface has C^2 continuity in both directions, whereas the Cardinal version has C^1 continuity in both directions.

3. Arbitrary Degree Surfaces.

The Tensor Product form for arbitrary degree surfaces follows by analogy with that for cubics. The most straightforward extension for our purposes is the Bezier form, since we already know the arbitrary degree form of the basis. The degree n,n Bezier surface is

$$S(t,u) = \sum_{i,j=0}^{n} B_n\ i(t)\ Bn\ j(u)\ p_{i,j}\ ,$$

where the $B_{n,k}(s)$ are the degree n binomial distribution functions given in equation l-28.

The relation of this Bezier form to different geometric representations can be obtained very easily. Simply transform from one representaion to the other by inverting the appropriate basis matrix and performing the transformation analogous to that indicated in equation (12). Again, the reasons for favoring the

Bezier form are that the sub-patch and convex hull properties hold for arbitrary degree, thereby providing powerful computational tools with recursive algorithms.

4. Parametric Volumes.

All of the formulations presented in Chapter 1 have their trivariate extensions, which give rise to volume elements rather than surface patches. The procedure is the same as that used to extend curves to surfaces. That is, the geometric coefficients in the formulation being extended are treated as functions of a third parameter v, rather than constant vectors. As this third parameter varies between 0 and 1, a family of surfaces sweeps out a volume element, or "chunk". The particular function that is used is arbitrary. However, the standard tensor product form is generated if each of the new functions is of the same form as that used to represent the two parameter surfaces being extended to volumes.

In generating volume representations, we must give up the matrix notation used to represent curves and surfaces, since the geometry tensor is now a tensor of n-component vectors, where n is the dimension of the space. Instead, we will indicate the arithmetic involved by using the standard summation notation. The tensor product form of a tri-cubic volume representation is of the form

$$V(t,u,v) = \sum_{i,j,k=0}^{3} f_i(t)\, f_j(u)\, f_k(v)\, g_{i,j,k} \qquad (14)$$

where the $f_m(s)$ functions are any of the basis functions given in Chapter 1 and the $g_{i,j,k}$ are the components of the geometry tensor.

These volume representations are useful in representing approximations to three dimensional data, such as pressure fields in solutions to fluid-dynamic equations or density contours in three dimensional x-ray data. Since the patch splitting algorithms that have been so successful in rendering surface patches can be applied to volumes, these volumes can be easily rendered. Routines for displaying flow-fields, etc. using this approach are currently being developed. Chapter 5 gives the algorithms used for these routines.

5. Mixed Representations.

In the previous sections of this chapter, we extended curves to surfaces. In each case this was done by letting the geometry vector for the curve be a function of another parameter rather than a constant vector. Moreover, we always assumed that the various elements of the geometry vector were functions of the new parameter of the same form as the original curve; in doing so we obtained the normal form of the Tensor Product. We mentioned, however, that a surface can be generated by allowing the elements of the geometry vector to be arbitrary functions of the new parameter.

For example, suppose that in extending the Bezier form to a surface we allow each of the elements of the geometry vector of equation 10 to be, not Bezier cubic functions of u, but instead Hermite cubic functions of u. In that case, equation 10 assumes the form

$$[\ p_1(u)\ p_2(u)\ p_3(u)\ p_4(u)\ ] =$$

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} M_h \begin{bmatrix} p_{11} & p21 & p31 & p_{41} \\ p_{12} & p22 & p_{32} & p_{42} \\ s_{11} & s_{21} & s_{31} & s_{41} \\ s_{12} & s_{22} & s_{32} & s_{42} \end{bmatrix}, \qquad (15)$$

where p_{il} is the u=0 end of curve $p_i(u)$, s_{il} is its u=0 tangent, etc. Transposing the result and putting it into equation l-13, we obtain the mixed Bezier-Hermite bicubic surface:

$$S(t,u) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_b \; G_{bh} \; M_h{}^t \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix} \qquad (16)$$

where G_{bh} is the transpose of the matrix in (15).

Other mixed surfaces can be generated depending upon the particular needs at hand. A particularly fruitful approach might be to allow the four curves $p_i(u)$ to be non-polynomial functions of the parameter variable, for example circular arcs or complete circles.

Note that by following the procedure we have used to generate surface representations it is also possible to obtain formulations that have mixed degree. For example, the degree n,m Binomial surface is given by

$$S(t,u) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{n,i}(t) \; B_{m \;\; j,}(u) \; p_{i \;,j} \qquad (17)$$

That is, the surface is degree n in the t parameter and degree m in the u parameter.

Exercises.

3-l. Suppose you have a sequence of n circles in space, each given by and expression of the form $C_i(t)$; that is, each of them is a function of a parameter t such that the circle is swept out by the vector function as the parameter is varied betweem 0 and 1. Give an expression that will describe a surface passing through all of these circles in such a way that a topologically cylindrical object is generated.

Chapter 4
Computational Techniques for Surfaces

In this Chapter the techniques introduced in Chapter 2 for curves are extended to surfaces. In Section 1, patch subdivision is presented. These techniques are useful in generating shaded pictures of surfaces and volume elements. Section 2 is concerned with incermental techniques for generating line-drawn renderings of bivariate surface patches. Then in section 3, we present table driven methods that are useful in rapidly updating patch descriptions. These techniques are used in the routines for manipulating patch files that are described in the appendix.

1. Subdivision of Patches.

In Section 1.1 of Chapter 2, we showed that equation 2-4 gives the expression for obtaining the control points for the "first" subcurve of a curve whose geometry matrix is G. Also, in Chapter 3 we showed that a surface patch in the Tensor Product form is simply a "curve" whose controlling "points" are themselves curves. Because of this, we may use the H_1 matrix given by equation l-4 to "split" a geometry tensor for a surface patch in either of the two parametric directions of the patch. Similarly, the H_2 matrix may be used to split the tensor in either parametric direction. Therefore, if it is desired to split a patch into four subpatches, the control point tensors for the four patches are given by the expressions

$$
\begin{aligned}
G_{11} &= H_1 \; G \; H_1{}^t, \\
G_{12} &= H_1 \; G \; H_2{}^t, \\
G_{21} &= H_2 \; G \; H_1{}^t, \\
\text{and} \quad G_{22} &= H_2 \; G \; H_2{}^t.
\end{aligned} \tag{1}
$$

Similar expressions hold for higher degree patches. That is, after deriving the splitting matrices H_1 and H_2 corresponding to the basis used to represent the patch, the above expressions are applied to the geometry tensor. If the arbitrary degree patch is represented in Bezier form, the'expressions giving the new control points is the same as that given in equations 2-5,6,9, except that it is applied to only one parametric direction, i.e. to one of the subscripts of the control point tensor $p_{i,j}$ of the surface equation of Section 3 of Chapter 3.

Splitting volume representations represented in the Bezier form is done in the same way. Equation 3-14, with the binomial basis functions has its tensor $g_{i\,j\,k}$ split along one of the parametric directions by apply& the splitting rules of equations 2-5,6,9 to the corresponding subscript of $g_{i,j,k}$. The next chapter gives an example of splitting a volume representation so that equi-pressure contours representing solutions to flow-field computations can be displayed.

Exercises.

4-l. What expression gives the new geometry tensors if it is desired to split a patch in only one parametric direction? Extend this to splitting a volume element along one of its parametric directions.

2. Difference Equations for Surfaces.

The difference techniques introduced for curves in Section 2 of Chapter 2 can be extended to surfaces. The equation analogous to equation 2-12 is

$$\underline{S} = E(e_t) \; M_q \; G_{qq} \; M_q^t \; E(e_u)^t , \qquad (2)$$

where the q subscript denotes any of the bases we have discussed, e_t = 1/nsegt and e_u = 1/nsegu. Note that since the difference matrix E is 4 by 4, this equation is a 4 by 4 by n tensor, where n is the number of components.

Equation 2 represents a tensor of zero, first, second and third order differences in each parameter. Rather than write out the tensor in difference notation, it is simpler to describe how to use it. The first column of the tensor will generate the u=0 boundary curve of the patch if these coefficients are supplied to the curve drawing algorithm. That is, the second element in this first column is the first order difference in the t parameter, the third element in the column is the second order difference in the t parameter, etc. Likewise, the first row of the difference tensor will generate the curve along the t=0 boundary of the patch. In general, rows are to parameters (t,u) as columns are to parameters (u,t).

Suppose we wish to make a cross-hatched line-drawing of a surface patch represented by geometry tensor G_{qq} using basis matrix M_q. In addition, we want nsegu segments in each u curve and nsegt segments in each t curve. The following steps will generate it:

Preliminary steps:

A. Form S according to (2).

B. Transform the components (xyz) of S by the viewing matrix that expresses it in the viewing coordinate system. S may now have 4-vector homogeneous coordinates, i.e. x,y,z,and W.

Algorithm DRSURF(nsegt, nsegu, S):

1. Copy S into working registers R.

2. Repeat steps 2.1 and 2.2 nsegu times.

2.1 DRCURVE with nseg=nsegt and D=first column of R.

2.2 Add second column of R to first (all components), add third to second, add fourth to third.

3. Copy S transposed into working registers R.

4. Repeat steps 4.1 and 4.2 nsegt times.

4.1 DRCURVE with nseg=nsegu and D=first column of R.

4.2 Do step 2.2.

3. Table Driven Techniques.

It has often been demonstrated in computer science that efficiences in computation time can be traded with those of storage space. The author has shown that one example of such a tradeoff is applicable in making discrete renderings of curved surface patches[7]. Each of the surface formulations presented in Chapter

3 can be expressed in the form

$$S(t,u) = [\ q_1(t)\ q_2(t)\ q_3(t)\ q_4(t)\ ]\ G_{qq} \begin{matrix} q_1(u) \\ q_2(u) \\ q_3(u) \\ q_4(u) \end{matrix}$$

$$= \sum_{i,j=0}^{3} q_i(t)\ q_j(u)\ g_{i,j}\ ,$$

where the $q_k(s)$ are the basis functions and the $g_{i,j}$ are geometric control points.

Most of the arithmetic involved in computing this expression is in the evaluation of the basis functions. If we decide in advance to represent all patches only at certain values of the parameters, t and u, then we can precompute the basis functions, $q_i(t)*q_j(u)$, for all necessary values of the parameters and store them in tables. The tables can then be used to speed up the process of computing equation 3. Unless the step size in computing these tables is very small, this results in a small amount of storage.

Another time/space tradeoff can be made if we also keep the computed patch shape stored in a table. Suppose for each patch we precompute a table of x,y,z positions of the points being used to display the patch. Then any modification to the table resulting from the movement of a single control point through displacement $dg_{k,l}$ will result in a displacement of the corresponding patch (patches if we are dealing with one of the spline formulations) by

$$dS(t,u) = q_k(t)\ q_l(u)\ dg_{k,l}\ . \qquad (4)$$

Note that the displacement of a single control point has eliminated the need for summation.

These two schema are used to effect very rapid updates (near real-time) to patch displays using the routines described in the Appendix. .

Exercises.

4-2. Describe the algorithm required to generate updates to surfaces using equation 4. How does it differ if it is used with spline formulations.

4-3. Apply these incremental update ideas to the difference tensor of the previous section. Is it possible to get a savings in surface generation speed here as well?

Chapter 5
Applications

This chapter describes three different applications of the subdivision methods presented in the previous chapter. The first is an algorithm developed by Catmull[8] in 1974 for displaying quality shaded pictures of bivariate surface patches. This algorithm is described in sufficient detail to be easily implemented. Section 2 presents an extension of this algorithm to trivariate representations for the purpose of displaying equi-pressure contours in the solutions of flow-field computations. Section 3 presents an algorithm for determining, to within a specific tolerance, the boundaries of intersection of two bivariate surface patches of arbitrary degree.

1. Surface Patch Rendering.

Catmull has developed what seems to be the simplest possible solution to the problem of making high quality shaded pictures of curved surfaces composed of collections of bivariate surface patches. The only major problem with the algorithm is that it requires excessive storage. However, given that memory continues to decrease in price, this can perhaps be overlooked. The algorithm makes use of the patch splitting methods introduced in Chapter 4 and uses a raster-scan display for output. For each pixel, sufficient memory is required to represent a shading value, which is a minimum of 6 bits for monochrome and 18 bits for full color, and a depth coordinate, which typically requires a minimum of 16 bits. We will refer to the depth coordinate buffer as a z-buffer. These requirements may vary by optimizing

the algorithm, but for our purposes, we will assume that this memory is available.

The algorithm is non-deterministic and recursive. It can be described in a few simple steps. Suppose we have a patch described in Bezier form, i.e. we have a geometry tensor G of the Bezier control points of the patch. (If the patch is described in another form, it is simply transformed into Bezier form by the methods described in Chapter 3.) We may wish to render the patch in perspective. If so, to avoid lengthy computations we approximate perspective by transforming the control points into perspective space. True perspective is obtained only by different means[8], but this gives a good approximation to it. We will assume that all such transformations have been performed on the points and that the point coordinates lie in the range

$$0 \leq x \leq RES,$$
$$0 \leq y \leq RES,$$
$$\text{and} \quad 0 \leq z \leq 1, \qquad (1)$$

where z=0 is the nearest and z=1 is the maximum distance of points from the observer.

Preliminary Step: Initialize all z-buffer values to 1 (most distant) and initialize all pixel shade values to the desired background shade or color. Apply the following algorithm to each patch tensor G:

Algorithm SUBDIV(G):

1. Determine if G is larger in extent than a single pixel. That is, SIZE(G)>1, where SIZE computes the maximum of the two sizes of the x and y displacements of the bounding volume around G. It if is larger that 1, goto step 2;

otherwise, call RENDER(G) and return.

2. Determine the parametric direction in which G experiences the greatest change in size. Split G along this direction. For example, if varying the first subscript of the control points gives rise to the largest change in the extent of G, it changes most in the t direction and should be split as follows:

$$G_1 = H_1 G \quad \text{and} \quad G_2 = H_2 G.$$

If the other direction gives rise to the most change in x,y extent of G, the patch should be split as:

$$G_1 = G H_1^t \quad \text{and} \quad G_2 = G H_2^t.$$

(It is also possible to simply divide the patch into 4 subpatches according to equation 4-1, but for simplicity we will assume a binary subdivision.) Recursively apply SUBDIV to these patches: SUBDIV(G_1) and SUBDIV(G_2), then return from this routine.

Since less arithmetic is involved if the new control points are obtained by equations 2-5,6,9 properly applied to the bivariate geometry tensor, the actual splitting should be done according to that scheme. The above method is given only because it is conceptually the simplest.

Algorithm RENDER(G):

1. Find the (x,y) and z coordinates of the patch. (x,y) is the pixel it covers and z is its depth at the pixel (see note 1 below).

2. If the z coordinate of this patch is greater than the z-buffer contents for pixel (x,y), return. (The patch is not visible here.)

3. Compute the unit normal to the patch represented by G. (See note 2 below.)

4. Compute the desired shading value of the patch based on this normal, and write it into the shade buffer for the pixel. Store the z coordinate for the patch in the z buffer for the pixel. Return.

Note 1. The (x,y) and z coordinates for the patch can be found by simply taking the coordinates of one of the points in the patch. Since the entire patch is assumed to cover the pixel, this is normally satisfactory.

Note 2. This can be approximated in a variety of ways. Simply taking the plane defined by 3 appropriate control points might be sufficient in most cases. More care can be, and normally should be, taken to find an average normal for the patch, e.g. the average of the normals at the four corners.

The above descriptions will yield a straightforward implementation of Catmull's algorithm. However, various modifications to the RENDER portion of the algorithm are usually implemented in order to decrease the effects of sampling the object on a fairly coarse raster, such as 512 by 512. Also, step 4 of the RENDER algorithm may involve a variety of different computations. For example, multiple light sources, specular reflection and mapping of textures onto the surfaces are techniques that are commonly em-

ployed in applications requiring very high quality images. To generate adequate shaded pictures, however, a simple model to use for the lighting in step 4 is

$$I = I_0 N_z,$$

where I_0 is the basic color (or relative shade) of the surface at the pixel being considered, and N_z is the z component of the normal vector to the surface at this point. The reason for chosing N_z is that if the normal vector is normalized, this represents the cosine of the angle between the normal and the z axis, which extends outward from the observer; if the observer carries his light source with him, this approximates Lambert's law of illumination for light sources located at the observer.

It is worth pointing out that this algorithm can be used to display translucent surfaces by changing step 4 of RENDER so that it does not modify the z-buffer contents, and so that it only alters the shade value for the pixel according to the translucency factor for the surface. However, this approach will be a valid approximation for translucency only if all surfaces are translucent. It is easy to show that a depth first rendering order must be followed if translucency is to be handled correctly in the general case.

Figure 5-l shows a rendering of part of the fuselage of a fighter plane. This picture was produced using the algorithm described above. Figure 5-2 shows the same object viewed from the

other side.

2. Display of Pressure Contours of Flow-field Solutions.

Discrete solutions to fluid dynamics equations give rise to a grid of points in 3-D space. At each of the points an approximate solution to the fluid-flow equations is determined from the boundary conditions using the finite differencing scheme. Typical quantities obtained are the pressure and velocity of the fluid as it flows past the points.

This section describes an algorithm that will generate high quality shaded pictures of contours of constant pressure. The algorithm has not actually been implemented. However, it is a very straightforward extension of the algorithm of the previous section, which was used to produce Figures 5-l and 5-2. NASA Ames Research Center does 'not presently have the capability of displaying such pictures. However, plans are underway to acquire the equipment necessary to produce them. Therefore, this section presents sufficient information to allow an implementation of the algorithm.

Since the pressure of the contour that is to be displayed by the algorithm may be specified as a parameter, repeated applications of it can be made to generate a family of surfaces that represent a range of pressure values in the flow field. If these contours are in addition represented by varying translucent colors, this procedure may be used to display several of them in the same picture.

The output of the finite differencing mechanism is a spatial grid of points. It is a topologically rectangular grid that is ordered in three directions according to three subscripts. At

each of these points we will assume that the 3-D coordinate and pressure are known, and we will represent each point by a 4-vector of the form

$$g_{i,j,k} = [\, x_{i,j,k} \;\; y_{i,j,k} \;\; z_{i,j,k} \;\; p_{i,j,k} \,1.$$

Although other quantities are known at each point, such as the velocity of the flow, we address only the problem of displaying the pressure field.

Refering to the volume formulation of equation 14, Chapter 3, we can fit a set of Cardinal spline "chunks", or volume elements, to the set of points $g_{i,j,k}$. Each chunk will be of the form

$$V_{l,m,n}(t,u,v) = \sum_{i,j,k=0}^{3} C_i(t)\, C_j(u)\, C_k(v)\; g_{l+i,m+j,n+k}.$$

where l,m and n range through all values necessary to include all of the $g_{i,j,k}$ in the grid of solution points. The composite volume generated by this equation provides a cubic interpolant volume that is C^1 continuous; it is a cubic spline in 3 variables. This cardinal spline generates a continuous field of pressure and coordinate values from a discrete field of points, just as a bicubic spline generates a continuous surface from a discrete set of points.

Given a particular value of pressure, p_0, our task is to determine through which of these volume elements, or "chunks", the contour corresponding to this pressure passes. All other volume elements may be ignored. Those through which it passes are recursively subdivided, just as in the surface subdivision algorithm. The results of the subdivision are chunks that are dis-

carded if and only if the pressure contour does not pass through them. When the remaining chunks are small enough that their lateral extent is no larger than a single pixel, each is written into the z-buffer according to the same rules as in the previous algorithm, except for a difference in the way that the render algorithm computes the plane representing the chunk.

Although the data management of such a scheme seems incomprehensible to anyone not familiar with recursion, it is in fact a very simple algorithm. Fortunately, the data stack on which recursion is based provides the necessary data management.

The following description assumes that the z-buffer has been initialized as in the patch subdivision algorithm. We also assume that all transformations have been done to put the x, y and z coordinates in the range indicated by (1). The following algorithm should be applied to each of the chunks given by equation 2, each of which has already been transformed into Bezier form.

Algorithm SUBDIV(G) -

1. Determine if the pressure, p_0, is within the bounds of the pressures of the 64 points of G. If not, return.

2. Determine the parametric direction, i.e. which subscript of the 3 of G, gives rise to the largest change in lateral extent of the chunk. Split it into two chunks, G_1 and G_2. Then recursively call SUBDIV(G_1) and SUBDIV(G_2).

3. If the chunk does not cover more than one pixel, RENDER(G) and return.

The RENDER algorithm is the same as in the surface patch algorithm with one exception. In this case, we have a (small) volume G of 64 points, some of which have pressure values greater than p_0 and others of which have pressure values less than p_0. We must find a plane (really only normal vector) which slices this small volume of points such that those greater than p_0 are on one side of it and those less than p_0 are on the other side of it.

3. Patch-patch intersections.

Another very important application of the subdivision methods introduced in Chapter 4 is that of finding the boundaries of intersection of two bivariate patches. Other attempts have been made at solving this problem[9], but they involve numerical approximation schema based on Newton iteration methods and consequently are sometimes stability dependent. Also, they are designed around bicubic patches, and although these are currently the most widely used bivariate surfaces, there is no reason to restrict ourselves to bicubics if it is avoidable. The subdivison algorithms work for arbitrary degree patches with no alteration.

The algorithm presented here, ISECT, will determine all of the points lying on the boundary (or boundaries) of intersection of two patches in Bezier form, to within a specified tolerance. However, because of the recursive nature of the algorithm, the points will in general not be found in an ordered sequence along the boundary. Therefore, the output of ISECT must be supplied to another algorithm, which is not described here but is available in a C implementation. The ordering of the points is straightforwardly done by a sort, but the sort is much more efficient if information about the recursive entries and exits to ISECT is provided as part of its output.

The algorithm ISECT assumes that two patches with control point tensors given by A and B are to be split; the parameters of the two patches are (t,u) and (v,w), respectively. ISECT first tests the bounding volumes of the two geometry tensors for the patches, A and B. If they do not overlap, then because of the convex hull property, the patches do not intersect. If they do, but both are smaller than the specified tolerance, then a point is output, i.e. the t,u,v,w parameter values at the point of "intersection" and the coordinates of the point of intersection are output. If either is larger than the tolerance, the larger of them is split along its largest dimension and the results are passed to ISECT for recursive processing.

Algorithm ISECT(A, B, WHICH)

1. Compute the new value of the parameter indicated by WHICH. (See Note 1.)

2. Compute the sizes of the two tensors A and B. If the larger of them is greater than tolerance, goto step 4.

3. Output t,u,v and w and the position of one point on A or B.

3.1 Restore the parameter indicated by WHICH to its value on entering the routine. Return.

4. If A is larger than B, goto step 5. Otherwise, split B along its larger direction into B_1 and B_2.

   If split in the v direction then

   ISECT(A,B_1,5); ISECT(A,B_2,6); goto step 3.1

   Otherwise,

   ISECT(A,B_1,7); ISECT(A,B_2,8); goto step 3.1

5. Split A alone its largest direction into A_1 and A_2.

If split in the t direction then

ISECT(A_1,B,1); ISECT(A_2,B,2); goto step 3.1.

Otherwise,

ISECT(A_1,B,3); ISECT(A_2,B,4); goto step 3.1.

Note 1. The parameter WHICH is used to tell which of the global parameters t,u,v,w to modify. WHICH=1 or 2 indicates dt is to be divided by 2 because it is going to be either added or subtracted to t. WHICH=1 indicates t, which is initially 0.5, should have dt, which is initially 0.5, subtracted from it. WHICH=2 indicates t should have dt added to it. Each pair of values for WHICH indicate similar operations should be performed for u,v and w. The reverse of these operations should be performed when a return is made from ISECT.

References

1. Coons, S. A., Surfaces for Computer-Aided Design of Space Forms, MIT Project MAC TR-41, June 1967.

2. Forrest, A. R., On Coons and Other Methods for the Representation of Curved Surfaces, Computer Graphics and Image Processing 1, pp. 341-359 (1972)

3. Riesenfeld, R. F., Applications of B-spline Approximation to Geometric Problems of Computer Aided Design, University of Utah UTEC-CSc-73-126.

4. Schoenberg, I. J., "On Variation Diminishing Approximation Methods", On Numerical Approximation, Ed. by R.E. Langer, University of Wisconsin press, 1959, pp. 249-274.

5. deBoor, C. "On Uniform Approximation with Splines", J. Approximation Theory, Vol 1, 1968, pp.219-235.

6. Forrest, A. R., "Computational Geometry", Proceedings of the Royal Society of London, London, A321, pp.187-195

7. Clark, J. H., Designing Surfaces in 3-D, CACM, Vol. 19, No. 11, October, 1976.

8. Catmull, E., A Subdivision Algorithm fo Computer Display of Curved Surfaces, PhD Thesis, University of Utah UTEC-csc-74-133.