COMPUTER AIDED GEOMETRIC DESIGN

Thomas W. Sederberg

October 23, 2012

Preface

This semester is the 24^{th} time I have taught a course at Brigham Young University titled, "Computer Aided Geometric Design." When I first taught such a course in 1983, the field was young enough that no textbook covered everything that I wanted to teach, and so these notes evolved.

The field now has matured to the point that several semesters worth of valuable material could be compiled. These notes, admittedly biased towards my own interests, reflect my personal preferences as to which of that material is most beneficial to students in an introductory course.

I welcome anyone who has an interest in studying this fascinating topic to make free use of these notes. I invite feedback on typos and on material that could be written more clearly.

Thomas W. Sederberg
Department of Computer Science
Brigham Young University
tom@cs.byu.edu
January 2007

Contents

List of Figures

Chapter 1

Introduction

Computer aided geometric design (CAGD) concerns itself with the mathematical description of shape for use in computer graphics, manufacturing, or analysis. It draws upon the fields of geometry, computer graphics, numerical analysis, approximation theory, data structures and computer algebra.

CAGD is a young field. The first work in this field began in the mid 1960s. The term *computer aided geometric design* was coined in 1974 by R.E. Barnhill and R.F. Riesenfeld in connection with a conference at the University of Utah.

This chapter presents some basic background material such as vector algebra, equations for lines and conic sections, homogeneous coordinates.

1.1 Points, Vectors and Coordinate Systems

Consider the simple problem of writing a computer program which finds the area of any triangle. We must first decide how to uniquely describe the triangle. One way might be to provide the lengths l_1, l_2, l_3 of the three sides, from which Heron's formula yields

$$Area = \sqrt{s(s-l_1)(s-l_2)(s-l_3)}, \qquad s = \frac{l_1+l_2+l_3}{2}.$$

An alternative way to describe the triangle is in terms of its vertices. But while the lengths of the sides of a triangle are independent of its position, we can specify the vertices to our computer program only with reference to some *coordinate system* — which can be defined simply as any method for representing points with numbers.

Note that a coordinate system is an artificial devise which we arbitrarily impose for the purposes at hand. Imagine a triangle cut out of paper and lying on a flat table in the middle of a room. We could define a *Cartesian coordinate system* whose origin lies in a corner of the room, and whose coordinate axes lie along the three room edges which meet at the corner. We would further specify the unit of measurement, say centimeters. Then, each vertex of our triangle could be described in terms of its respective distance from the two walls containing the origin and from the floor. These distances are the Cartesian coordinates (x, y, z) of the vertex with respect to the coordinate system we defined.

Vectors A *vector* can be pictured as a line segment of definite length with an arrow on one end. We will call the end with the arrow the *tip* or *head* and the other end the *tail*.

Two vectors are equivalent if they have the same length, are parallel, and point in the same direction (have the same *sense*) as shown in Figure 1.1. For a given coordinate system, we can

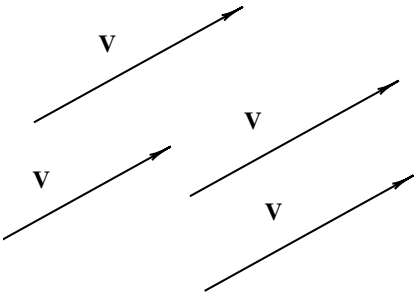
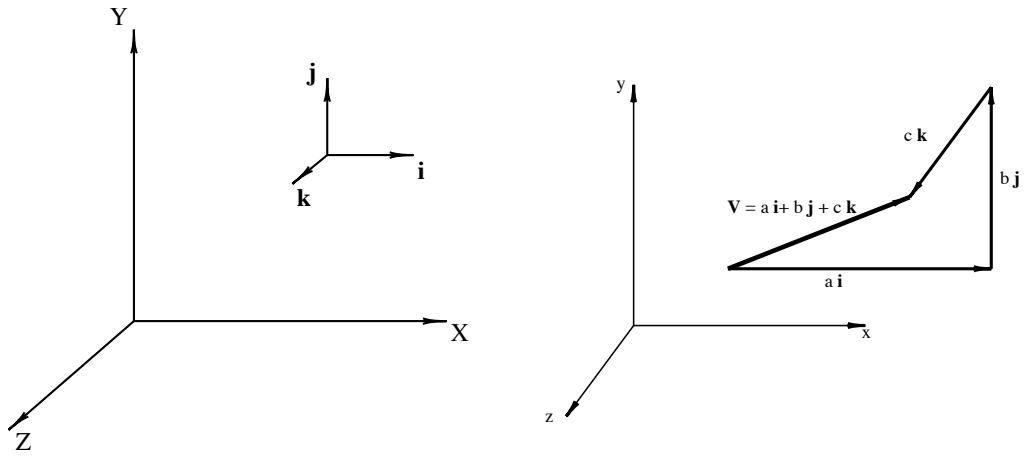


Figure 1.1: Equivalent Vectors

describe a three-dimensional vector in the form (a, b, c) where a (or b or c) is the distance in the x (or y or z) direction from the tail to the tip of the vector.



(a) Unit Vectors.

(b) Vector in Component Form.

Figure 1.2: Vectors.

Unit Vectors The symbols **i**, **j**, and **k** denote vectors of "unit length" (based on the unit of measurement of the coordinate system) which point in the positive x, y, and z directions respectively (see Figure 1.2.a).

Unit vectors allow us to express a vector in component form (see Figure 1.2.b):

$$\mathbf{P} = (a, b, c) = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}.$$

An expression such as (x, y, z) can be called a *triple* of numbers. In general, an expression $(x_1, x_2, \ldots, x_n)$ is an n-tuple, or simply a tuple. As we have seen, a triple can signify either a point or a vector.

Relative Position Vectors Given two points $\mathbf{P}_1$ and $\mathbf{P}_2$, we can define

$$\mathbf{P}_{2/1} = \mathbf{P}_2 - \mathbf{P}_1$$

as the vector pointing from $\mathbf{P}_1$ to $\mathbf{P}_2$. This notation $\mathbf{P}_{2/1}$ is widely used in engineering mechanics, and can be read "the position of point $\mathbf{P}_2$ relative to $\mathbf{P}_1$"

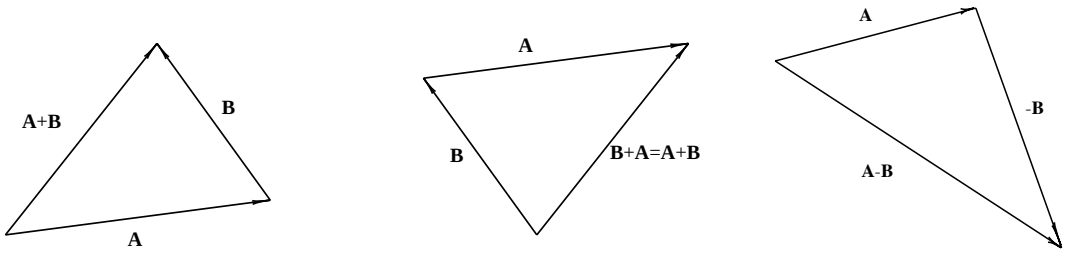
In our diagrams, points will be drawn simply as dots or small circles, and vectors as line segments with single arrows. Vectors and points will both be denoted by bold faced type.

1.2 Vector Algebra

Given two vectors $\mathbf{P}_1 = (x_1, y_1, z_1)$ and $\mathbf{P}_2 = (x_2, y_2, z_2)$, the following operations are defined:

Addition:

$$\mathbf{P}_1 + \mathbf{P}_2 = \mathbf{P}_2 + \mathbf{P}_1 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$



(a) Vector Addition. (b) Vector Subtraction.

Figure 1.3: Vector Addition and Subtraction.

Subtraction:

$$\mathbf{P}_1 - \mathbf{P}_2 = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$$

Scalar multiplication:

$$c\mathbf{P}_1 = (cx_1, cy_1, cz_1)$$

Length of a Vector

$$|\mathbf{P}_1| = \sqrt{x_1^2 + y_1^2 + z_1^2}$$

A vector of length one is called a unit vector.

$$\frac{\mathbf{P}_1}{|\mathbf{P}_1|}$$

is a unit vector in the direction of $\mathbf{P}_1$

Dot Product The dot product of two vectors is defined

$$\mathbf{P}_1 \cdot \mathbf{P}_2 = |\mathbf{P}_1||\mathbf{P}_2| \cos\theta \tag{1.1}$$

where θ is the angle between the two vectors. Since the unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are mutually perpendicular,

$$\mathbf{i} \cdot \mathbf{i} = \mathbf{j} \cdot \mathbf{j} = \mathbf{k} \cdot \mathbf{k} = 1$$

$$\mathbf{i} \cdot \mathbf{j} = \mathbf{i} \cdot \mathbf{k} = \mathbf{j} \cdot \mathbf{k} = 0.$$

The dot product obeys the distributive law

$$\mathbf{P}_1 \cdot (\mathbf{P}_2 + \mathbf{P}_3) = \mathbf{P}_1 \cdot \mathbf{P}_2 + \mathbf{P}_1 \cdot \mathbf{P}_3,$$

As a result of the distributive law,

$$\begin{aligned} \mathbf{P}_1 \cdot \mathbf{P}_2 &= (x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}) \cdot (x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}) \\ &= (x_1 * x_2 + y_1 * y_2 + z_1 * z_2) \end{aligned} \tag{1.2}$$

(1.2) enables us to compute the angle between any two vectors. From (1.1),

$$\theta = \cos^{-1}\left(\frac{\mathbf{P}_1 \cdot \mathbf{P}_2}{|\mathbf{P}_1||\mathbf{P}_2|}\right).$$

Example. Find the angle between vectors $(1, 2, 4)$ and $(3, -4, 2)$.

Answer.

$$\begin{aligned} \theta &= \cos^{-1}\left(\frac{\mathbf{P}_1 \cdot \mathbf{P}_2}{|\mathbf{P}_1||\mathbf{P}_2|}\right) \\ &= \cos^{-1}\left(\frac{(1,2,4) \cdot (3,-4,2)}{|(1,2,4)||(3,-4,2)|}\right) \\ &= \cos^{-1}\left(\frac{3}{\sqrt{21}\sqrt{29}}\right) \\ &\approx 83.02^\circ \end{aligned}$$

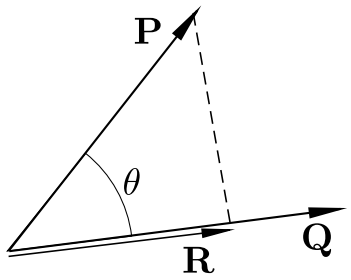


Figure 1.4: Vector Projection

An important application of dot products is in computing the projection of one vector onto another vector. As illustrated in Figure 1.4, vector $\mathbf{R}$ is the projection of vector $\mathbf{P}$ onto vector $\mathbf{Q}$. Since

$$|\mathbf{R}| = |\mathbf{P}| \cos\theta = \frac{\mathbf{P} \cdot \mathbf{Q}}{|\mathbf{Q}|}$$

we have

$$\mathbf{R} = |\mathbf{R}|\frac{\mathbf{Q}}{|\mathbf{Q}|} = \frac{\mathbf{P}\cdot\mathbf{Q}}{|\mathbf{Q}|}\frac{\mathbf{Q}}{|\mathbf{Q}|} = \frac{\mathbf{P}\cdot\mathbf{Q}}{\mathbf{Q}\cdot\mathbf{Q}}\mathbf{Q}.$$

Example. Find the projection of $\mathbf{P} = (3, 2, 1)$ onto $\mathbf{Q} = (3, 6, 6)$.

Answer.

$$\begin{aligned}\mathbf{R} &= \frac{\mathbf{P}\cdot\mathbf{Q}}{\mathbf{Q}\cdot\mathbf{Q}}\mathbf{Q}\\ &= \frac{(3,2,1)\cdot(3,6,6)}{(3,6,6)\cdot(3,6,6)}(3,6,6)\\ &= \frac{27}{81}(3,6,6)\\ &= (1,2,2)\end{aligned}$$

Cross Product: The cross product $\mathbf{P}_1 \times \mathbf{P}_2$ is a vector whose magnitude is

$$|\mathbf{P}_1 \times \mathbf{P}_2| = |\mathbf{P}_1||\mathbf{P}_2|\sin\theta$$

(where again θ is the angle between $\mathbf{P}_1$ and $\mathbf{P}_2$), and whose direction is mutually perpendicular to $\mathbf{P}_1$ and $\mathbf{P}_2$ with a sense defined by the right hand rule as follows. Point your fingers in the direction of $\mathbf{P}_1$ and orient your hand such that when you close your fist your fingers pass through the direction of $\mathbf{P}_2$. Then your right thumb points in the sense of $\mathbf{P}_1 \times \mathbf{P}_2$.

From this basic definition, one can verify that

$$\mathbf{P}_1 \times \mathbf{P}_2 = -\mathbf{P}_2 \times \mathbf{P}_1,$$

$$\mathbf{i}\times\mathbf{j} = \mathbf{k}, \quad \mathbf{j}\times\mathbf{k} = \mathbf{i}, \quad \mathbf{k}\times\mathbf{i} = \mathbf{j}$$

$$\mathbf{j}\times\mathbf{i} = -\mathbf{k}, \quad \mathbf{k}\times\mathbf{j} = -\mathbf{i}, \quad \mathbf{i}\times\mathbf{k} = -\mathbf{j}.$$

The cross product obeys the distributive law

$$\mathbf{P}_1 \times (\mathbf{P}_2 + \mathbf{P}_3) = \mathbf{P}_1 \times \mathbf{P}_2 + \mathbf{P}_1 \times \mathbf{P}_3.$$

This leads to the important relation

$$\begin{aligned}\mathbf{P}_1 \times \mathbf{P}_2 &= (x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}) \times (x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k})\\ &= (y_1z_2 - y_2z_1, x_2z_1 - x_1z_2, x_1y_2 - x_2y_1)\\ &= \begin{vmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k}\\ x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\end{vmatrix}\end{aligned} \tag{1.3}$$

Area of a Triangle. Cross products have many important uses, such as finding a vector which is mutually perpendicular to two other vectors and finding the area of a triangle which is defined by three points $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$.

$$Area = \frac{1}{2}|\mathbf{P}_{1/2}||\mathbf{P}_{1/3}|\sin\theta_1 = \frac{1}{2}|\mathbf{P}_{1/2} \times \mathbf{P}_{1/3}| \tag{1.4}$$

For example, the area of a triangle with vertices $\mathbf{P}_1 = (1,1,1)$, $\mathbf{P}_2 = (2,4,5)$, $\mathbf{P}_3 = (3,2,6)$ is

$$\begin{aligned} Area &= \frac{1}{2}|\mathbf{P}_{1/2} \times \mathbf{P}_{1/3}| \\ &= \frac{1}{2}|(1,3,4) \times (2,1,5)| \\ &= \frac{1}{2}|(11,3,-5)| = \frac{1}{2}\sqrt{11^2+3^2+(-5)^2} \\ &\approx 6.225 \end{aligned}$$

1.2.1 Points vs. Vectors

A point is a geometric entity which connotes position, whereas a vector connotes direction and magnitude. From a purely mathematical viewpoint, there are good reasons for carefully distinguishing between triples that refer to points and triples that signify vectors [Goldman '85]. However, no problem arises if we recognize that a triple connoting a point can be interpreted as a vector from the origin to the point. Thus, we could call a point an *absolute position vector* and the difference between two points a *relative position vector*. These phrases are often used in engineering mechanics, where vectors are used to express quantities other than position, such as velocity or acceleration.

1.3 Rotation About an Arbitrary Axis

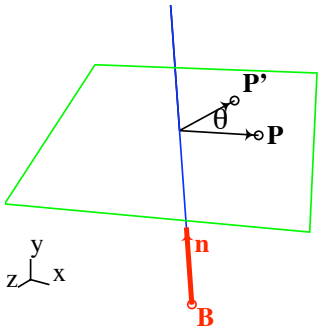
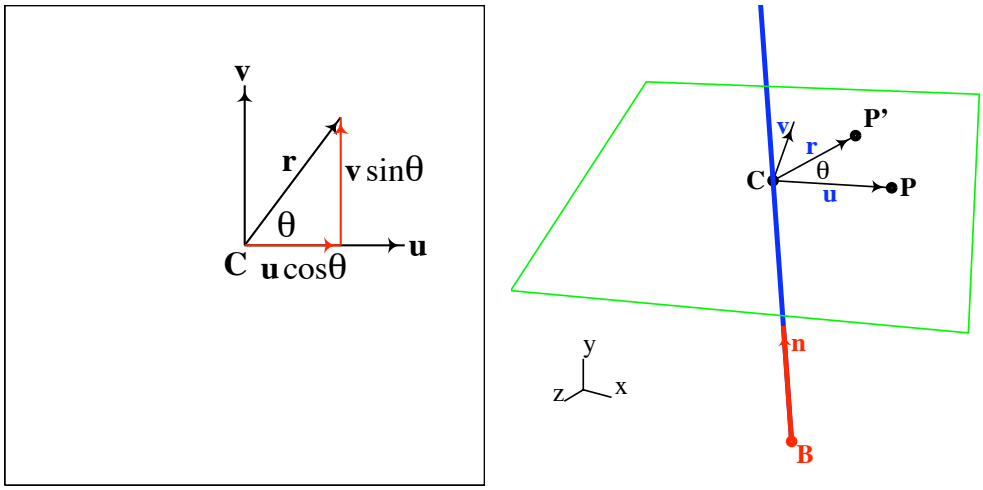


Figure 1.5: Rotation about an Arbitrary Axis

The problem of computing a rotation about an arbitrary axis is fundamental to CAGD and computer graphics. The standard solution to this problem as presented in most textbooks on computer graphics involves the concatenation of seven 4×4 matrices. We present here a straightforward solution comprised of the four simple vector computations in equations (1.6) through (1.9) — a compelling example of the power of vector algebra.

Figure 1.5 shows a point $\mathbf{P}$ which we want to rotate an angle θ about an axis that passes through $\mathbf{B}$ with a direction defined by unit vector $\mathbf{n}$. So, given the angle θ, the unit vector $\mathbf{n}$, and Cartesian coordinates for the points $\mathbf{P}$ and $\mathbf{B}$, we want to find Cartesian coordinates for the point $\mathbf{P}'$.

The key insight needed is shown in Figure 1.6.a.



(a) Key Insight. (b) Labels.

Figure 1.6: Rotation about an Arbitrary Axis Using Vector Algebra.

Let $\mathbf{u}$ and $\mathbf{v}$ be any two three-dimensional vectors that satisfy $u \cdot v = 0$ (that is, they are perpendicular) and $|\mathbf{u}| = |\mathbf{v}| \neq 0$ (that is, they are they same length but not necessarily unit vectors). We want to find a vector $\mathbf{r}$ that is obtained by rotating $\mathbf{u}$ an angle θ in the plane defined by $\mathbf{u}$ and $\mathbf{v}$. As suggested in Figure 1.6,

$$\mathbf{r} = \mathbf{u}\cos\theta + \mathbf{v}\sin\theta. \tag{1.5}$$

With that insight, it is easy to compute a rotation about an arbitrary axis. Note that $(\mathbf{C} - \mathbf{B})$ is the projection of vector $(\mathbf{P} - \mathbf{B})$ onto the unit vector $\mathbf{n}$. Referring to the labels in Figure 1.6.b, we compute

$$\mathbf{C} = \mathbf{B} + [(\mathbf{P} - \mathbf{B}) \cdot \mathbf{n}]\mathbf{n}. \tag{1.6}$$

$$\mathbf{u} = \mathbf{P} - \mathbf{C} \tag{1.7}$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} \tag{1.8}$$

Then, $\mathbf{r}$ is computed using equation (3.1), and

$$\mathbf{P}' = \mathbf{C} + \mathbf{r}. \tag{1.9}$$

Example
Find the coordinates of a point $(5, 7, 3)$ after it is rotated an angle $\theta = 90°$ about an axis that points in the direction $(2, 1, 2)$ and that passes through the point $(1, 1, 1)$.
Answer:
$\mathbf{n} = (\frac{2}{3}, \frac{1}{3}, \frac{2}{3})$.

$$\begin{aligned} \mathbf{C} &= \mathbf{B} + [(\mathbf{P} - \mathbf{B}) \cdot \mathbf{n}]\mathbf{n} \\ &= (1,1,1) + [((5,7,3) - (1,1,1)) \cdot (\frac{2}{3}, \frac{1}{3}, \frac{2}{3})](\frac{2}{3}, \frac{1}{3}, \frac{2}{3}) \\ &= (5,3,5) \end{aligned} \tag{1.10}$$

$$\mathbf{u} = \mathbf{P} - \mathbf{C} = (0, 4, -2)$$
$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (-\frac{10}{3}, \frac{4}{3}, \frac{8}{3}).$$
$$\mathbf{r} = \mathbf{u}\cos\theta + \mathbf{v}\sin\theta = \mathbf{u} \cdot 0 + \mathbf{v} \cdot 1 = \mathbf{v}.$$

1.3.1 Matrix Form

It is possible to take these simple vector equations and to create from them a single 4×4 transformation matrix for rotation about an arbitrary axis. While this is useful to do in computer graphics (where, in fact, this matrix is typically created by concatenating seven 4×4 matrices), the simple vector equations we just derived suffice for many applications. The derivation of this matrix is presented here for your possible reference. We will not be using it in this course.

Let $\mathbf{P} = (x, y, z)$, $\mathbf{P'} = (x', y', z')$, $\mathbf{B} = (B_x, B_y, B_z)$, and $\mathbf{n} = (n_x, n_y, n_z)$. We seek a 4×4 matrix M such that

$$M \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix} = \begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix}$$

$$(C_x, C_y, C_z) = (B_x, B_y, B_z) + [xn_x + yn_y + zn_z - \mathbf{B} \cdot \mathbf{n}](n_x, n_y, n_z) \tag{1.11}$$

$$\begin{aligned} C_x &= xn_x^2 + yn_xn_y + zn_xn_z + B_x - (\mathbf{B} \cdot \mathbf{n})n_x && (1.12)\\ C_y &= xn_xn_y + yn_y^2 + zn_yn_z + B_y - (\mathbf{B} \cdot \mathbf{n})n_y && (1.13)\\ C_z &= xn_xn_z + yn_yn_z + zn_z^2 + B_z - (\mathbf{B} \cdot \mathbf{n})n_z && (1.14) \end{aligned}$$

$$\mathbf{u} = (x, y, z) - (C_x, C_y, C_z) \tag{1.15}$$

$$\begin{aligned} u_x &= x(1 - n_x^2) - yn_xn_y - zn_xn_z + (\mathbf{B} \cdot \mathbf{n})n_x - B_x && (1.16)\\ u_y &= -xn_xn_y + y(1 - n_y^2) - zn_yn_z + (\mathbf{B} \cdot \mathbf{n})n_y - B_y && (1.17)\\ u_z &= -xn_xn_z - yn_yn_z + z(1 - n_z^2) + (\mathbf{B} \cdot \mathbf{n})n_z - B_z && (1.18) \end{aligned}$$

$$\begin{aligned} v_x &= n_yu_z - n_zu_y && (1.19)\\ v_y &= n_zu_x - n_xu_z && (1.20)\\ v_z &= n_xu_y - n_yu_x && (1.21) \end{aligned}$$

$$\begin{aligned} r_x &= u_x\cos\theta + (n_yu_z - n_zu_y)\sin\theta && (1.22)\\ r_y &= u_y\cos\theta + (n_zu_x - n_xu_z)\sin\theta && (1.23)\\ r_z &= u_z\cos\theta + (n_xu_y - n_yu_x)\sin\theta && (1.24) \end{aligned}$$

$$(x', y', z') = (C_x + r_x, C_y + r_y, C_z + r_z) \tag{1.25}$$

$$\begin{aligned} x' &= xn_x^2 + yn_xn_y + zn_xn_z + B_x - (\mathbf{B}\cdot\mathbf{n})n_x + \\ &\quad [x(1-n_x^2) - yn_xn_y - zn_xn_z + (\mathbf{B}\cdot\mathbf{n})n_x - B_x]\cos\theta + \\ &\quad n_y[-xn_xn_z - yn_yn_z + z(1-n_z^2) + (\mathbf{B}\cdot\mathbf{n})n_z - B_z]\sin\theta - \\ &\quad n_z[-xn_xn_y + y(1-n_y^2) - zn_yn_z + (\mathbf{B}\cdot\mathbf{n})n_y - B_y]\sin\theta \end{aligned} \tag{1.26}$$

$$\begin{aligned} x' &= x[n_x^2 + (1-n_x^2)\cos\theta] + y[n_xn_y(1-\cos\theta) - n_z\sin\theta] \\ &+ z[n_xn_z(1-\cos\theta) + n_y\sin\theta] + (B_x - (\mathbf{B}\cdot\mathbf{n})n_x)(1-\cos\theta) + (n_zB_y - n_yB_z)\sin\theta. \end{aligned}$$

Since $n_x^2 + n_y^2 + n_z^2 = 1$, $(1-n_x^2) = n_y^2 + n_z^2$. In like manner we can come up with an expression for y' and z', and our matrix M is thus

$$\begin{bmatrix} n_x^2 + (n_y^2+n_z^2)\cos\theta & n_xn_y(1-\cos\theta) - n_z\sin\theta & n_xn_z(1-\cos\theta) + n_y\sin\theta & T_1 \\ n_xn_y(1-\cos\theta) + n_z\sin\theta & n_y^2 + (n_x^2+n_z^2)\cos\theta & n_yn_z(1-\cos\theta) - n_x\sin\theta & T_2 \\ n_xn_z(1-\cos\theta) - n_y\sin\theta & n_yn_z(1-\cos\theta) + n_x\sin\theta & n_z^2 + (n_x^2+n_y^2)\cos\theta & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.27}$$

with

$$\begin{aligned} T_1 &= (B_x - (\mathbf{B}\cdot\mathbf{n})n_x)(1-\cos\theta) + (n_zB_y - n_yB_z)\sin\theta \\ T_2 &= (B_y - (\mathbf{B}\cdot\mathbf{n})n_y)(1-\cos\theta) + (n_xB_z - n_zB_x)\sin\theta \\ T_3 &= (B_z - (\mathbf{B}\cdot\mathbf{n})n_z)(1-\cos\theta) + (n_yB_x - n_xB_y)\sin\theta \end{aligned}$$

1.4 Parametric, Implicit, and Explicit Equations

There are basically three types of equations that can be used to define a planar curve: parametric, implicit, and explicit. The parametric equation of a plane curve takes the form

$$x = \frac{x(t)}{w(t)} \qquad y = \frac{y(t)}{w(t)}. \tag{1.28}$$

The implicit equation of a curve is of the form

$$f(x,y) = 0. \tag{1.29}$$

An explicit equation of a curve is a special case of both the parametric and implicit forms:

$$y = f(x). \tag{1.30}$$

In these notes, we restrict ourselves to the case where the functions $x(t)$, $y(t)$, $w(t)$, $f(x)$ and $f(x,y)$ are polynomials.

Any curve that can be expressed parametrically as in equation (1.28) is referred to as a **rational** curve. In the classical algebraic geometry literature, a rational curve is sometimes called a **unicursal** curve, which means that it can be sketched in its entirety without removing one's pencil from the paper. In computer aided geometric design, rational curves are often called **rational parametric** curves. The case where $w(t) \equiv 1$ is called a **polynomial** parametric curve (or a **non-rational** parametric curve). A curve that can be expressed in the form of equation (1.29) is known as a **planar algebraic** curve.

The parametric equation of a curve has the advantage of being able to quickly compute the (x, y) coordinates of points on the curve for plotting purposes. Also, it is simple to define a curve *segment* by restricting the parameter t to a finite range, for example $0 \leq t \leq 1$. On the other hand, the implicit equation of a curve enables one to easily determine whether a given point lies on the curve, or if not, which side of the curve it lies on. Chapter 17 shows that it is always possible to compute an implicit equation for a parametric curve. It is trivial to convert en explicit equation of a curve into a parametric equation ($x = t$, $y = y(x)$) or into an implicit equation ($f(x) - y = 0$). However, a curve defined by an implicit or parametric equation cannot in general be converted into explicit form.

A **rational surface** is one that can be expressed

$$x = \frac{x(s,t)}{w(s,t)} \qquad y = \frac{y(s,t)}{w(s,t)} \qquad z = \frac{z(s,t)}{w(s,t)} \tag{1.31}$$

where $x(s,t)$, $y(s,t)$, $z(s,t)$ and $w(s,t)$ are polynomials. Also, a surface that can be expressed by the equation

$$f(x, y, z) = 0 \tag{1.32}$$

where $f(x, y, z)$ is a polynomial is called an **algebraic surface.**

A **rational space curve** is one that can be expressed by the parametric equations

$$x = \frac{x(t)}{w(t)} \quad y = \frac{y(t)}{w(t)} \quad z = \frac{z(t)}{w(t)}. \tag{1.33}$$

The curve of intersection of two algebraic surfaces is an **algebraic space curve.**

1.5 Lines

The simplest case of a curve is a line. Even so, there are several different equations that can be used to represent lines.

1.5.1 Parametric equations of lines

Linear parametric equation

A line can be written in parametric form as follows:

$$x = a_0 + a_1 t; \qquad y = b_0 + b_1 t$$

In vector form,

$$\mathbf{P}(t) = \left\{ \begin{array}{c} x(t) \\ y(t) \end{array} \right\} = \left\{ \begin{array}{c} a_0 + a_1 t \\ b_0 + b_1 t \end{array} \right\} = \mathbf{A}_0 + \mathbf{A}_1 t. \tag{1.34}$$

In this equation, $\mathbf{A}_0$ is a point on the line and $\mathbf{A}_1$ is the direction of the line (see Figure 1.7)

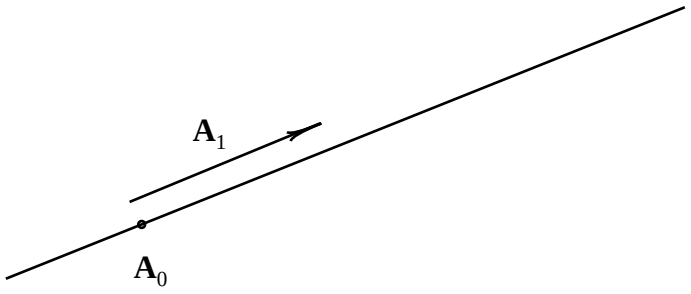
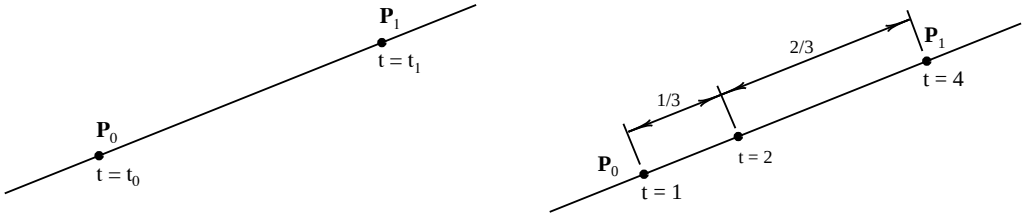


Figure 1.7: Line given by $\mathbf{A}_0 + \mathbf{A}_1 t$.



(a) Line given by $\mathbf{P}(t) = \frac{(t_1-t)\mathbf{P}_0+(t-t_0)\mathbf{P}_1}{t_1-t_0}$.

(b) Affine Example.

Figure 1.8: Affine parametric equation of a line.

Affine parametric equation of a line

A line can also be expressed

$$\mathbf{P}(t) = \frac{(t_1 - t)\mathbf{P}_0 + (t - t_0)\mathbf{P}_1}{t_1 - t_0} \tag{1.35}$$

where $\mathbf{P}_0$ and $\mathbf{P}_1$ are two points on the line and t_0 and t_1 are any parameter values. Note that $\mathbf{P}(t_0) = \mathbf{P}_0$ and $\mathbf{P}(t_1) = \mathbf{P}_1$. Note in Figure 1.8.a that the line *segment* $\mathbf{P}_0$–$\mathbf{P}_1$ is defined by restricting the parameter:

$$t_0 \le t \le t_1.$$

Sometimes this is expressed by saying that the line segment is the portion of the line in the *parameter interval* or *domain* $[t_0, t_1]$.

We will see that the line in Figure 1.8.a is actually a degree one Bézier curve. Most commonly, we have $t_0 = 0$ and $t_1 = 1$ in which case

$$\mathbf{P}(t) = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1. \tag{1.36}$$

Equation 1.36 is called an *affine* equation, whereas equation 1.34 is called a *linear* equation. An affine equation is coordinate system independent, and is mainly concerned with ratios and proportions. An affine equation can be thought of as answering the question: "If a line is defined through two points $\mathbf{P}_0$ and $\mathbf{P}_1$, and if point $\mathbf{P}_0$ corresponds to parameter value t_0 and point $\mathbf{P}_1$ corresponds to parameter value t_1, what point corresponds to an arbitrary parameter value t?" Figure 1.8.b shows a line on which $\mathbf{P}_0$ corresponds to parameter $t = t_0 = 1$ and $\mathbf{P}_1$ is assigned

parameter value $t = t_1 = 4$. For example, the point corresponding to $t = 2$ is one third of the way from $\mathbf{P}_0$ to $\mathbf{P}_1$.

Note that an affine equation can be derived from any two points on a line, given the parameter values for those points. If $\mathbf{P}(\alpha)$ is the point corresponding to parameter value $t = \alpha$ and if $\mathbf{P}(\beta)$ is the point corresponding to parameter value $t = \beta$ ($\alpha \neq \beta$), then the point corresponding to parameter value γ is

$$\mathbf{P}(\gamma) = \mathbf{P}(\alpha) + \frac{\gamma - \alpha}{\beta - \alpha}[\mathbf{P}(\beta) - \mathbf{P}(\alpha)] = \frac{(\beta - \gamma)\mathbf{P}(\alpha) + (\gamma - \alpha)\mathbf{P}(\beta)}{\beta - \alpha}$$

Rational parametric equations

A line can also be defined using the following parametric equations:

$$x = \frac{a_0 + a_1 t}{d_0 + d_1 t}; \quad y = \frac{b_0 + b_1 t}{d_0 + d_1 t}. \tag{1.37}$$

This is often called *rational* or *fractional* parametric equations.

Recall that the homogeneous Cartesian coordinates (X, Y, W) of a point are related to its Cartesian coordinates by

$$(x, y) = (\frac{X}{W}, \frac{Y}{W}).$$

Thus, we can rewrite equation 1.37 as

$$X = a_0 + a_1 t; \quad Y = b_0 + b_1 t; \quad W = d_0 + d_1 t.$$

This equation can be further re-written in terms of homogeneous parameters (T, U) where $t = \frac{T}{U}$. Thus,

$$X = a_0 + a_1 \frac{T}{U}; \quad Y = b_0 + b_1 \frac{T}{U}; \quad W = d_0 + d_1 \frac{T}{U}.$$

But since we can scale (X, Y, W) without changing the point (x, y) which it denotes, we can scale by U to give

$$X = a_0 U + a_1 T; \quad Y = b_0 U + b_1 T; \quad W = d_0 U + d_1 T.$$

1.5.2 Implicit equations of lines

A line can also be expressed using an *implicit* equation:

$$f(x, y) = ax + by + c = 0; \quad \text{or} \quad F(X, Y, W) = aX + bY + cW = 0.$$

The line defined by an implicit equation is the set of all points which satisfy the equation $f(x, y) = 0$.

An implicit equation for a line can be derived given a point $\mathbf{P}_0 = (x_0, y_0)$ on the line and the normal vector $\mathbf{n} = a\mathbf{i} + b\mathbf{j}$. As shown in Figure 1.9, a point $\mathbf{P} = (x, y)$ is on this line if

$$(\mathbf{P} - \mathbf{P}_0) \cdot \mathbf{n} = 0$$

from which

$$f(x, y) = (x - x_0, y - y_0) \cdot (a, b) = ax + by - (ax_0 + by_0) = 0. \tag{1.38}$$

From equation 1.38, a line whose implicit equation is $ax+by+c = 0$ has the normal vector $\mathbf{n} = a\mathbf{i}+b\mathbf{j}$.

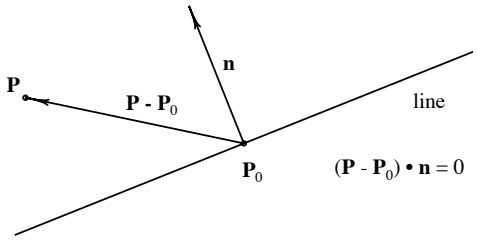


Figure 1.9: Line defined by point and normal.

Implicit equation of a line through two points

Three points (X_1, Y_1, W_1), (X_2, Y_2, W_2) and (X_3, Y_3, W_3) are collinear if

$$\begin{vmatrix} X_1 & Y_1 & W_1 \\ X_2 & Y_2 & W_2 \\ X_3 & Y_3 & W_3 \end{vmatrix} = 0.$$

Thus, the equation of the line through two points is

$$\begin{vmatrix} X & Y & W \\ X_1 & Y_1 & W_1 \\ X_2 & Y_2 & W_2 \end{vmatrix} = (Y_1W_2 - Y_2W_1)X + (X_2W_1 - X_1W_2)Y + (X_1Y_2 - X_2Y_1)W = 0.$$

1.5.3 Distance from a point to a line

If $\mathbf{n} = a\mathbf{i} + b\mathbf{j}$ is a unit vector (that is, if $a^2 + b^2 = 1$), then the value $f(x, y)$ in equation 1.38 indicates the signed perpendicular distance of a point (x, y) to the line. This can be seen from equation 1.38 and Figure 1.9. The dot product $(\mathbf{P} - \mathbf{P}_0) \cdot \mathbf{n}$ is the length of the projection of vector $(\mathbf{P} - \mathbf{P}_0)$ onto the unit normal $\mathbf{n}$, which is the perpendicular distance from $\mathbf{P}$ to the line.

Since the coefficients of an implicit equation can be uniformly scaled without changing the curve (because if $f(x, y) = 0$, then $c \times f(x, y) = 0$ also), the implicit equation of a line can always be *normalized*:

$$f(x, y) = a'x + b'y + c' = \frac{a}{\sqrt{a^2 + b^2}}x + \frac{b}{\sqrt{a^2 + b^2}}y + \frac{c}{\sqrt{a^2 + b^2}} = 0.$$

Then, $f(x, y)$ is the signed distance from the point (x, y) to the line, with all points on one side of the line having $f(x, y) > 0$ and the other side having $f(x, y) < 0$. Note that $|c'| = |f(0, 0)|$ is the distance from the origin to the line. Thus, if $c = 0$, the line passes through the origin. The coefficients a' and b' relate to the slope of the line. Referring to Figure 1.10, $a' = \cos(\theta)$, $b' = \sin(\theta)$, and $c' = -p$.

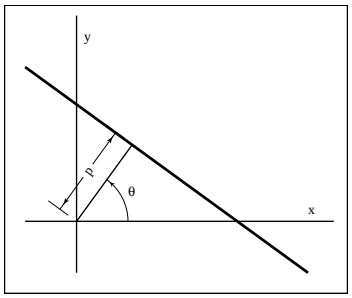


Figure 1.10: Normalized line equation.

1.6 Conic Sections

A conic section (or, simply *conic*) is any degree two curve. Any conic can be expressed using a degree two implicit equation:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

or, in homogeneous form:

$$aX^2 + bXY + cY^2 + dXW + eYW + fW^2 = 0. \tag{1.39}$$

Conics can be classified as hyperbolas, parabolas and ellipses (of which the circle is a special case). What distinguishes these cases is the number of real points at which the curve intersects the line at infinity $W = 0$. A hyperbola intersects $W = 0$ in two real points. Those points are located an infinite distance along the asymptotic directions. A parabola is tangent to the line at infinity, and thus has two coincident real intersection points. This point is located an infinite distance along the parabola's axis of symmetry. Ellipses do not intersect the line at infinity at any real point — all real points on an ellipse are finite.

To determine the number of real points at which a conic intersects the line at infinity, simply intersect equation 1.39 with the line $W = 0$ by setting $W = 0$ to get:

$$aX^2 + bXY + cY^2 = 0$$

from which

$$\frac{Y}{X} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2c}.$$

The two values Y/X are the slopes of the lines pointing to the intersections of the conic with the line at infinity. Thus, if $b^2 - 4ac > 0$, there are two distinct real intersections and the conic is a hyperbola. If $b^2 - 4ac = 0$, there are two coincident real intersections and the conic is a parabola, and if $b^2 - 4ac < 0$, there are no real intersections and the conic is an ellipse. The value $b^2 - 4ac$ is known as the *discriminant* of the conic.

1.6.1 Parametric equations of conics

The parametric equation of any conic can be expressed:

$$x = \frac{a_2t^2 + a_1t + a_0}{d_2t^2 + d_1t + d_0}; \qquad y = \frac{b_2t^2 + b_1t + b_0}{d_2t^2 + d_1t + d_0}.$$

or, in homogeneous form,

$$X = a_2T^2 + a_1TU + a_0U^2;$$
$$Y = b_2T^2 + b_1TU + b_0U^2;$$
$$W = d_2T^2 + d_1TU + d_0U^2.$$

It is also possible to classify a conic from its parametric equation. We again identify the points at which the conic intersects the line at infinity. In the parametric form, the only places at which (x, y) can be infinitely large is at parameter values of t for which

$$d_2t^2 + d_1t + d_0 = 0.$$

Thus, we note that $d_1^2 - 4d_0d_2$ serves the same function as the discriminant of the implicit equation. If $d_1^2 - 4d_0d_2 > 0$, there are two real, distinct values of t at which the conic goes to infinity and the curve is a hyperbola. If $d_1^2 - 4d_0d_2 < 0$, there are no real values of t at which the conic goes to infinity and the curve is an ellipse. If $d_1^2 - 4d_0d_2 = 0$, there are two real, identical values of t at which the conic goes to infinity and the curve is a parabola.

Chapter 2

Bézier Curves

Bézier curves are named after their inventor, Dr. Pierre Bézier, an engineer with the Renault car company who set out in the early 1960's to develop a curve formulation for use in shape design that would be intuitive enough for designers and artists to use, without requiring a background in mathematics.

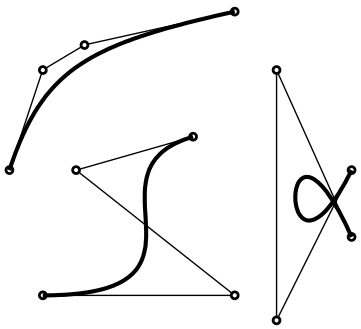


Figure 2.1: Examples of cubic Bézier curves.

Figure 2.1 shows three different Bézier curves, with their corresponding *control polygons*. Each control polygon is comprised of four *control points* that are connect with line segments. (These control polygons are not closed, and might more properly be called polylines.) The beauty of the Bézier representation is that a Bézier curve mimics the shape of its control polygon. A Bézier curve passes through its first and last control points, and is tangent to the control polygon at those endpoints. An artist can quickly master the process of designing shapes using Bézier curves by moving the control points, and most 2D drawing systems like Adobe Illustrator use Bézier curves.

Complicated shapes can be created by using a sequence of Bézier curves. Since Bézier curves are tangent to their control polygons, it is easy to join together two Bézier curves such that they are tangent continuous. Figure 2.2 shows the outline of a letter "g" created using several Bézier curves. All PostScript font outlines are defined using Bézier curves. Hence, as you read these notes, you are gazing upon Bézier curves!

Although Bézier curves can be used productively by artists who have little mathematical training,

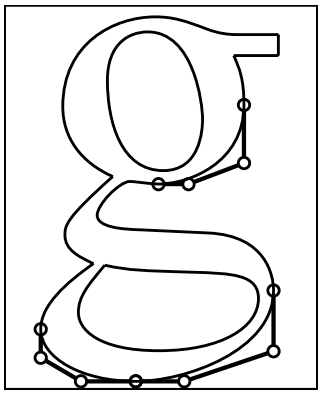
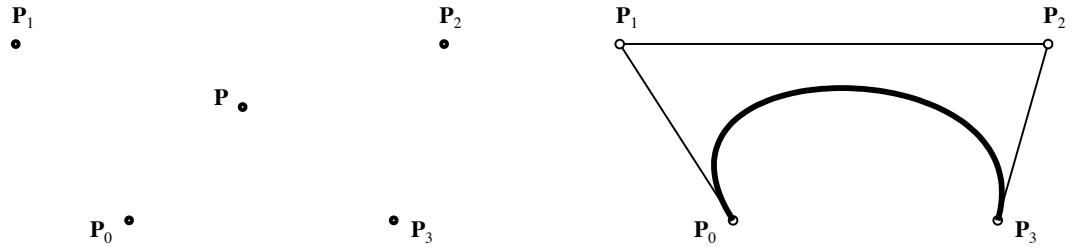


Figure 2.2: Font definition using Bézier curves.

one of the main objectives in this course is to study the underlying mathematics. These notes attempt to show that the power and elegance of Bézier curves are matched by the beauty of the underlying mathematics.

2.1 The Equation of a Bézier Curve

The equation of a Bézier curve is similar to the equation for the center of mass of a set of point masses. Consider the four masses m_0, m_1, m_2, and m_3 in Figure 2.3.a located at points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$.



(a) Center of mass of four points. (b) Cubic Bézier Curve.

Figure 2.3: Bézier Curves in Terms of Center of Mass.

The equation for the center of mass is

$$\mathbf{P} \;=\; \frac{m_0\mathbf{P}_0 \;+\; m_1\mathbf{P}_1 \;+\; m_2\mathbf{P}_2 \;+\; m_3\mathbf{P}_3}{m_0 \;+\; m_1 \;+\; m_2 \;+\; m_3}.$$

Next, imagine that instead of being fixed, constant values, each mass varies as a function of a parameter t. Specifically, let

$$m_0(t) \;=\; (1-t)^3, \quad m_1(t) \;=\; 3t(1-t)^2, \quad m_2(t) \;=\; 3t^2(1-t), \quad m_3(t) \;=\; t^3. \tag{2.1}$$

Now, for each value of t, the masses assume different weights and their center of mass changes continuously. As t varies between 0 and 1, a curve is swept out by the moving center of mass.

Figure 2.3.b shows the Bézier curve that results when the point masses in Figure 2.3.a are taken as control points. This curve is a cubic Bézier curve — *cubic* because the mass equations are degree three polynomials in t.

Notice that the mass equations in (2.1) sum identically to one:

$$(1-t)^3 + 3t(1-t)^2 + 3t^2(1-t) + t^3 = [(1-t)+t]^3 = 1^3 \equiv 1,$$

and so we can write the equation of this Bézier curve as $\mathbf{P}(t) = m_0(t)\mathbf{P}_0 + m_1(t)\mathbf{P}_1 + m_2(t)\mathbf{P}_2 + m_3(t)\mathbf{P}_3$.

The mass functions are plotted in the graph in Figure 2.4. Note that when $t = 0$, $m_0 = 1$ and

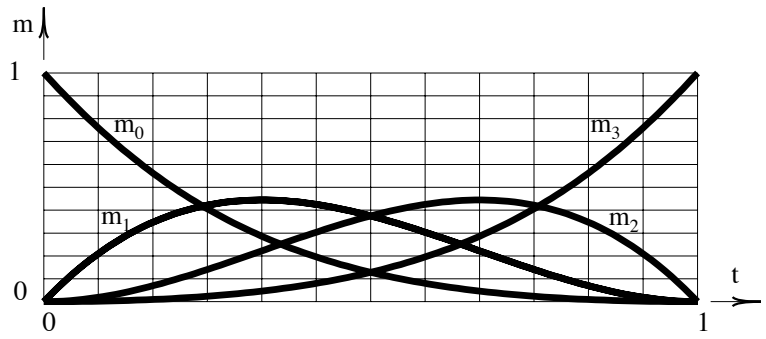


Figure 2.4: Cubic Bézier blending functions.

$m_1 = m_2 = m_3 = 0$. This explains why the curve passes through $\mathbf{P}_0$. When $t = 1$, $m_3 = 1$ and $m_0 = m_1 = m_2 = 0$, and the curve passes through point $\mathbf{P}_3$.

The variable masses $m_i(t)$ are usually called *blending functions* and, as noted before, the locations $\mathbf{P}_i$ are known as *control points*. The blending functions, in the case of Bézier curves, are known as *Bernstein polynomials*. We will later look at other curves formed with different blending functions.

Bézier curves of any degree can be defined. Figure 2.5 shows sample curves of degree one through four. A degree n Bézier curve has $n+1$ control points whose blending functions are denoted $B_i^n(t)$,

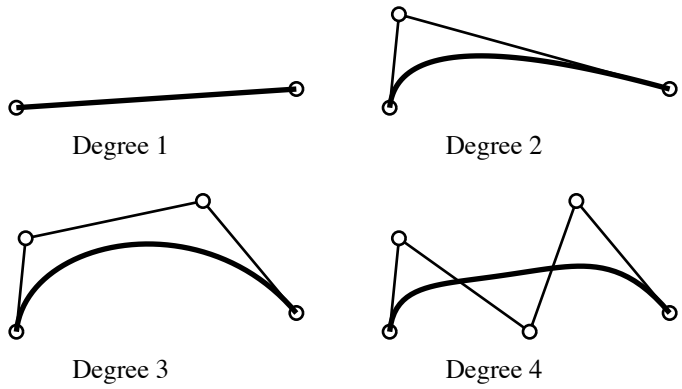


Figure 2.5: Bézier curves of various degree.

where

$$B_i^n(t) \;=\; \binom{n}{i}(1-t)^{n-i}t^i, \quad i = 0, 1, ..., n.$$

Recall that

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

$\binom{n}{i}$ is spoken "n – *choose* – i" and is called a *binomial coefficient* because it arises in the binomial expansion

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}.$$

In the degree three case, $n = 3$ and $B_0^3 = (1-t)^3$, $B_1^3 = 3t(1-t)^2$, $B_2^3 = 3t^2(1-t)$ and $B_3^3 = t^3$. $B_i^n(t)$ is also referred to as the ith Bernstein polynomial of degree n. The equation of a Bézier curve is thus:

$$\mathbf{P}(t) \;=\; \sum_{i=0}^{n} \binom{n}{i}(1-t)^{n-i}t^i\mathbf{P}_i. \tag{2.2}$$

2.2 Bézier Curves over Arbitrary Parameter Intervals

Equation 2.2 gives the equation of a Bézier curve which starts at $t = 0$ and ends at $t = 1$. It is useful, especially when fitting together a string of Bézier curves, to allow an arbitrary parameter interval:

$$t \in [t_0, t_1]$$

such that $\mathbf{P}(t_0) = \mathbf{P}_0$ and $\mathbf{P}(t_1) = \mathbf{P}_n$. We will denote the Bézier curve defined over an arbitrary parameter interval by $\mathbf{P}_{[t_0,t_1]}(t)$. It's equation is a modification of (2.2):

$$\mathbf{P}_{[t_0,t_1]}(t) \;=\; \frac{\sum_{i=0}^{n}\binom{n}{i}(t_1-t)^{n-i}(t-t_0)^i\mathbf{P}_i}{(t_1-t_0)^n} = \sum_{i=0}^{n}\binom{n}{i}(\frac{t_1-t}{t_1-t_0})^{n-i}(\frac{t-t_0}{t_1-t_0})^i\mathbf{P_i}. \tag{2.3}$$

If no parameter interval is specified, it is assumed to be $[0, 1]$. That is,

$$\mathbf{P}(t) = \mathbf{P}_{[0,1]}(t).$$

2.3 The de Casteljau Algorithm

The de Casteljau algorithm describes how to subdivide a Bézier curve $\mathbf{P}_{[t_0,t_2]}$ into two segments $\mathbf{P}_{[t_0,t_1]}$ and $\mathbf{P}_{[t_1,t_2]}$ whose union is equivalent to $\mathbf{P}_{[t_0,t_2]}$. This algorithm was devised in 1959 by Paul de Casteljau, a French mathematician at the Citroen automobile company. It is an example of a *geometric construction algorithm*.

To facilitate our description of the algorithm, we label the control points of a cubic Bézier curve $\mathbf{P}_{[t_0,t_2]}$ with $\mathbf{P}_0^0$, $\mathbf{P}_1^0$, $\mathbf{P}_2^0$, and $\mathbf{P}_3^0$ as illustrated in Figure 2.6. The algorithm involves computing the sequence of points

$$\mathbf{P}_i^j = (1-\tau)\mathbf{P}_i^{j-1} + \tau\mathbf{P}_{i+1}^{j-1}; \quad j = 1, \ldots, n; \quad i = 0, \ldots, n-j. \tag{2.4}$$

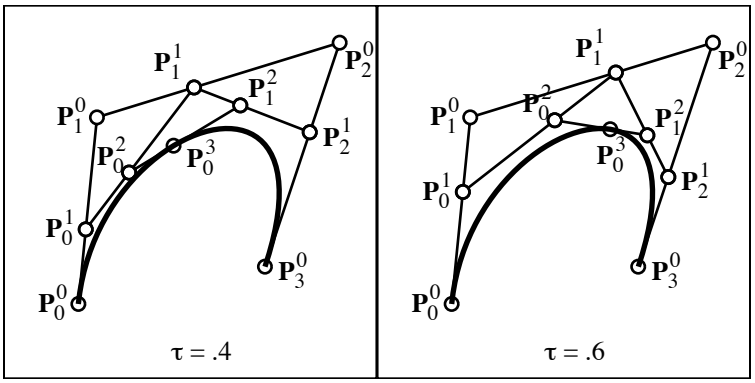


Figure 2.6: Subdividing a cubic Bézier curve.

where $\tau = \frac{t_1 - t_0}{t_2 - t_0}$. Then, the control points for $\mathbf{P}_{[t_0,t_1]}(t)$ are $\mathbf{P}_0^0, \mathbf{P}_0^1, \mathbf{P}_0^2, \ldots, \mathbf{P}_0^n$ and the control points for $\mathbf{P}_{[t_1,t_2]}(t)$ are $\mathbf{P}_0^n, \mathbf{P}_1^{n-1}, \mathbf{P}_2^{n-2}, \ldots, \mathbf{P}_n^0$. Although our example is for a cubic Bézier curve ($n = 3$), the algorithm works for any degree.

A practical application of the de Casteljau algorithm is that it provides a numerically stable means of computing the coordinates and tangent vector of any point along the curve, since $\mathbf{P}(t_1) = \mathbf{P}_0^n$ and the tangent vector is $\mathbf{P}_1^{n-1} - \mathbf{P}_0^{n-1}$.

Figure 2.7 shows that when a Bézier curve is repeatedly subdivided, the collection of control polygons converge to the curve. Thus, one way of plotting a Bézier curve is to simply subdivide it

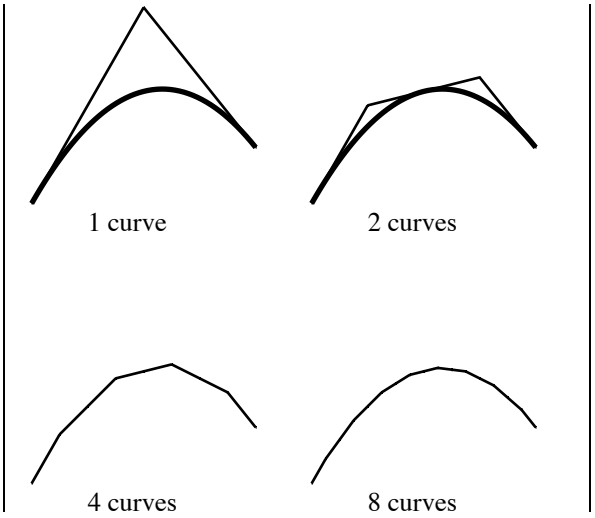


Figure 2.7: Recursively subdividing a quadratic Bézier curve.

an appropriate number of times and plot the control polygons (although a more efficient way is to use Horner's algorithm (see Chapter 3) or forward differencing (see Chapter 4)).

The de Casteljau algorithm works even if $\tau \notin [0, 1]$, which is equivalent to $t_1 \notin [t_0, t_2]$. Fig. 2.8 shows a quadratic Bézier curve "subdivided" at $\tau = 2$. The de Castelau algorithm is numerically stable as long as the parameter subdivision parameter is within the parameter domain of the curve.

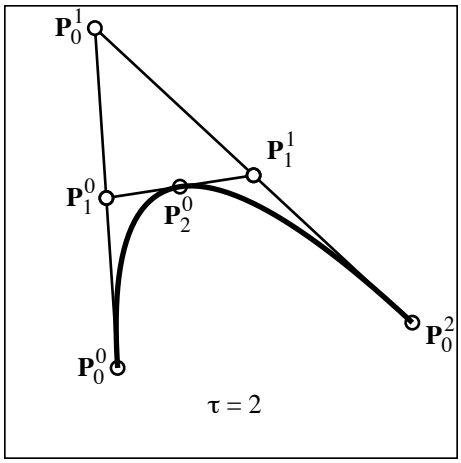


Figure 2.8: Subdividing a quadratic Bézier curve.

2.4 Degree Elevation

Any degree n Bézier curve can be exactly represented as a Bézier curve of degree $n+1$ (and hence, as a curve of any degree $m > n$). Given a degree n Bézier curve, the procedure for computing the control points for the equivalent degree $n+1$ Bézier curve is called *degree elevation*.

We now derive the degree elevation formula, using the identities:

$$\begin{aligned}
(1-t)B_i^n(t) &= (1-t)\binom{n}{i}(1-t)^{n-i}t^i \\
&= \binom{n}{i}(1-t)^{n-i+1}t^i \\
&= \frac{\binom{n}{i}}{\binom{n+1}{i}}\binom{n+1}{i}(1-t)^{n-i+1}t^i \\
&= \frac{\binom{n}{i}}{\binom{n+1}{i}}B_i^{n+1} = \frac{\frac{n!}{i!(n-i)!}}{\frac{(n+1)!}{i!(n+1-i)!}}B_i^{n+1} \\
&= \frac{n!i!(n+1-i)!}{i!(n-i)!(n+1)!}B_i^{n+1} = \frac{n!i!(n+1-i)(n-i)!}{i!(n-i)!(n+1)n!}B_i^{n+1} \\
&= \frac{n+1-i}{n+1}B_i^{n+1} \qquad (2.5)
\end{aligned}$$

and

$$tB_i^n(t) = \frac{i+1}{n+1}B_{i+1}^{n+1}(t). \qquad (2.6)$$

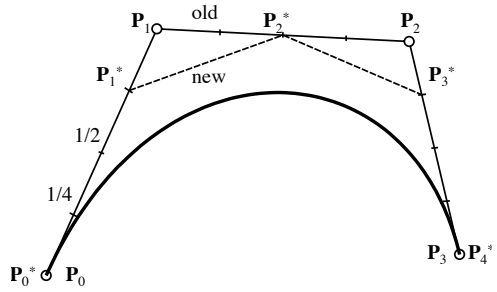
Degree elevation is accomplished by simply multiplying the equation of the degree n Bézier curve

by $[(1-t)+t]=1$:

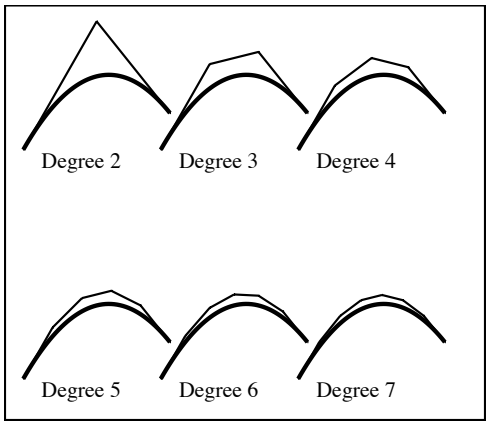
$$\begin{aligned}
\mathbf{P}(t) &= [(1-t)+t]\mathbf{P}(t) \\
&= [(1-t)+t]\sum_{i=0}^{n}\mathbf{P}_i B_i^n(t) \\
&= \sum_{i=0}^{n}\mathbf{P}_i[(1+t)B_i^n(t)+tB_i^n(t)] \\
&= \sum_{i=0}^{n}\mathbf{P}_i[\frac{n+1-i}{n+1}B_i^{n+1}+\frac{i+1}{n+1}B_{i+1}^{n+1}(t)] \\
&= \sum_{i=0}^{n}\frac{n+1-i}{n+1}\mathbf{P}_i B_i^{n+1}+\sum_{i=0}^{n}\frac{i+1}{n+1}\mathbf{P}_i B_{i+1}^{n+1}(t) \\
&= \sum_{i=0}^{n}\frac{n+1-i}{n+1}\mathbf{P}_i B_i^{n+1}+\sum_{i=1}^{n+1}\frac{i}{n+1}\mathbf{P}_{i-1} B_i^{n+1}(t) \\
&= \sum_{i=0}^{n+1}\frac{n+1-i}{n+1}\mathbf{P}_i B_i^{n+1}+\sum_{i=0}^{n+1}\frac{i}{n+1}\mathbf{P}_{i-1} B_i^{n+1}(t) \\
&= \sum_{i=0}^{n+1}[\frac{(n+1-i)\mathbf{P}_i+i\mathbf{P}_{i-1}}{n+1}]B_i^{n+1}(t) \\
&= \sum_{i=0}^{n+1}\mathbf{P}_i^* B_i^{n+1}(t) \qquad (2.7)
\end{aligned}$$

where

$$\mathbf{P}_i^* \;=\; \alpha_i\mathbf{P}_{i-1} \;+\; (1-\alpha_i)\mathbf{P}_i, \qquad \alpha_i \;=\; \frac{i}{n+1}. \qquad (2.8)$$



(a) Degree Elevation of a Cubic Bézier Curve.



(b) Repeated degree elevation.

Figure 2.9: Degree Elevation of a Bézier Curve.

For the case $n = 3$, the new control points $\mathbf{P}_i^*$ are:

$$\begin{aligned}
\mathbf{P}_0^* &= \mathbf{P}_0 \\
\mathbf{P}_1^* &= \frac{1}{4}\mathbf{P}_0 + \frac{3}{4}\mathbf{P}_1 \\
\mathbf{P}_2^* &= \frac{2}{4}\mathbf{P}_1 + \frac{2}{4}\mathbf{P}_2 \\
\mathbf{P}_3^* &= \frac{3}{4}\mathbf{P}_2 + \frac{1}{4}\mathbf{P}_3 \\
\mathbf{P}_4^* &= \mathbf{P}_3
\end{aligned}$$

Figure 2.9.a illustrates.

If degree elevation is applied repeatedly, as shown in Figure 2.9.b, the control polygon converges to the curve itself.

2.5 The Convex Hull Property of Bézier Curves

An important property of Bézier curves is that they always lie within the convex hull of their control points. The convex hull can be envisioned by pounding an imaginary nail into each control point, stretching an imaginary rubber band so that it surrounds the group of nails, and then collapsing that rubber band around the nails. The polygon created by that imaginary rubber band is the convex hull. Figure 2.10 illustrates.

The fact that Bézier curves obey the convex hull property is assured by the center-of-mass definition of Bézier curves in Section 2.1. Since all of the control points lie on one side of an edge of the convex hull, it is impossible for the center of mass of those control points to lie on the other side of the line,

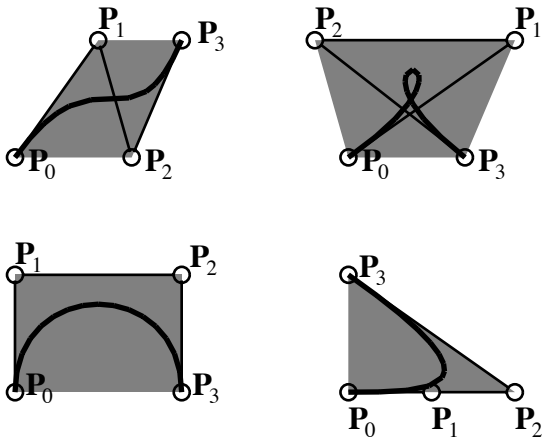


Figure 2.10: Convex Hull Property

2.6 Distance between Two Bézier Curves

The problem often arises of determining how closely a given Bézier curve is approximated by a second Bézier curve. For example, if a given cubic curve can be adequately represented by a degree elevated quadratic curve, it might be advantageous to replace the cubic curve with the quadratic.

Given two Bézier curves

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t); \qquad \mathbf{Q}(t) = \sum_{i=0}^{n} \mathbf{Q}_i B_i^n(t)$$

the vector $\mathbf{P}(t) - \mathbf{Q}(t)$ between points of equal parameter value on the two curves can itself be expressed as a Bézier curve

$$\mathbf{D}(t) = \mathbf{P}(t) - \mathbf{Q}(t) = \sum_{i=0}^{n} (\mathbf{P}_i - \mathbf{Q}_i) B_i^n(t)$$

whose control points are $\mathbf{D}_i = \mathbf{P}_i - \mathbf{Q}_i$. The vector from the origin to the point $\mathbf{D}(t)$ is $\mathbf{P}(t) - \mathbf{Q}(t)$. The convex hull property guarantees that the distance between the two curves is bounded by the largest distance from the origin to any of the control points $\mathbf{D}_i$.

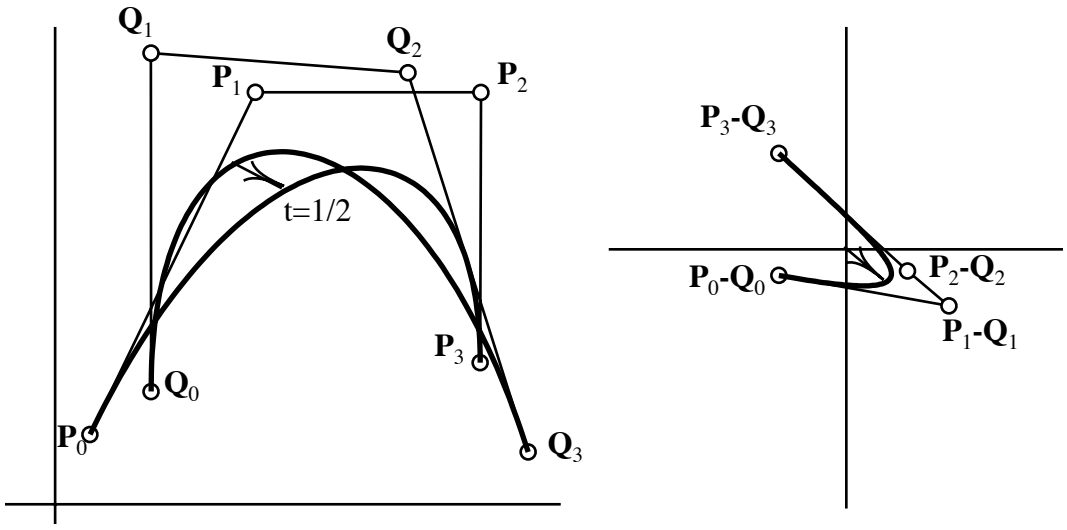


Figure 2.11: Difference curve.

This error bound is attractive because it is very easy to compute. However, it is not always a very tight bound because it is dependent on parametrization.

A more precise statement of the distance between two curves is the *Hausdorff* distance. Given two curves $\mathbf{P}_{[s_0,s_1]}(s)$ and $\mathbf{Q}_{[t_0,t_1]}(t)$. The Hausdorff distance $d(\mathbf{P}, \mathbf{Q})$ is defined

$$d(\mathbf{P}, \mathbf{Q}) = \max_{s \in [s_0, s_1]} \left\{ \min_{t \in [t_0, t_1]} |\mathbf{P}(s) - \mathbf{Q}(t)| \right\} \tag{2.9}$$

where $|\mathbf{P}(s) - \mathbf{Q}(t)|$ is the Euclidean distance between a point $\mathbf{P}(s)$ and the point $\mathbf{Q}(t)$. In practice, it is much more expensive to compute the Hausdorff distance than to compute a bound using the difference curve in Figure 2.11.

2.7 Derivatives

The parametric derivatives of a Bézier curve can be determined geometrically from its control points. For a polynomial Bézier curve $\mathbf{P}_{[t_0,t_1]}(t)$ of degree n with control points $\mathbf{P}_i$, the first parametric derivative can be expressed as a polynomial Bézier curve of degree $n-1$ with control points $\mathbf{D}_i$ where

$$\mathbf{D}_i \;=\; \frac{n}{t_1 - t_0}(\mathbf{P}_{i+1} \;-\; \mathbf{P}_i).$$

For example, the cubic Bézier curve in Fig. 2.12 has a first derivative curve as shown. Of course,

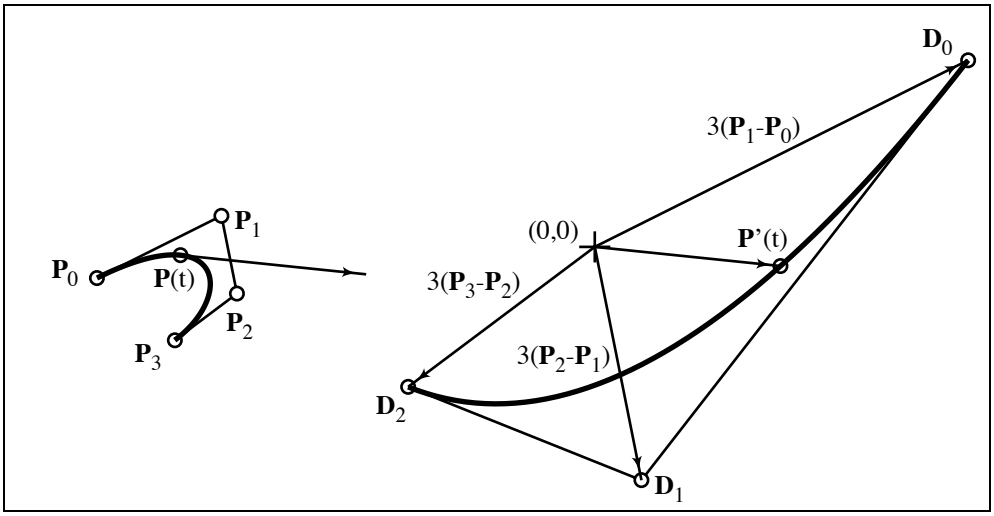


Figure 2.12: Hodograph.

the first derivative of a parametric curve provides us with a tangent vector. This is illustrated in Fig. 2.12 for the point $t \;=\; .3$. In this example, $t_0 = 0$ and $t_1 = 1$.

The first derivative curve is known as a *hodograph*. We can compute the second derivative of a Bézier curve by taking the hodograph of the hodograph (or, the second hodograph), etc.

It is interesting to note that if the hodograph passes through the origin, there is a cusp corresponding to that point on the original curve!

Note that the hodograph we have just described relates only to polynomial Bézier curves, *not* to rational Bézier curves or any other curve that we will study. The derivative of any other curve must be computed by differentiation. For a rational Bézier curve, that differentiation will involve the quotient rule. Consequently, the derivative of a degree n rational Bézier curve can be expressed as a rational Bézier curve of degree $2n$. As it turns out, the equations for those control points are rather messy.

2.8 Three Dimensional Bézier Curves

If Bézier control points are defined in three dimensional space, the resulting Bézier curve is three dimensional. Such a curve is sometimes called a *space* curve, and a two dimensional curve is called a *planar* curve. Our discussion of the de Casteljau algorithm, degree elevation, and hodographs extend to 3D without modification.

Since a degree two Bézier curve is defined using three control points, every degree two curve is planar, even if the control points are in a three dimensional coordinate system.

2.9 Rational Bézier Curves

A rational Bézier curve is one for which each control point $\mathbf{P}_i$ is assigned a scalar *weight*. The equation of a rational Bézier curve is

$$\frac{\sum_{i=0}^{n} w_i B_i^n(t)\mathbf{P}_i}{\sum_{i=0}^{n} w_i B_i^n(t)}.$$

The effect of changing a control point weight is illustrated in Fig. 2.13. This type of curve is known

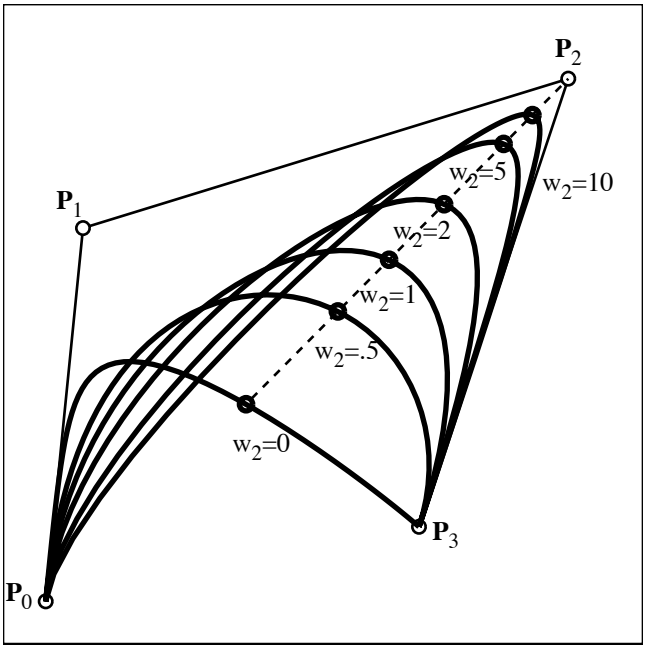


Figure 2.13: Rational Bézier curve.

as a *rational Bézier curve*, because the blending functions are rational polynomials, or the *ratio* of two polynomials. Note that if all weights are 1 (or if all weights are simply the same), a rational Bézier curve reduces to a *polynomial* Bézier curve.

Rational Bézier curves have several advantages over polynomial Bézier curves. Clearly, rational Bézier curves provide more control over the shape of a curve than does a polynomial Bézier curve. In addition, a perspective drawing of a 3D Bézier curve (polynomial or rational) is a rational Bézier curve, not a polynomial Bézier curve. Also, rational Bézier curves are needed to exactly express all conic sections. A degree two polynomial Bézier curve can only represent a parabola. Exact representation of circles requires rational degree two Bézier curves.

A rational Bézier curve can be interpreted as the perspective projection of a 3-D polynomial curve. Fig. 2.14 shows two curves: a 3-D curve and a 2-D curve. The 2-D curve lies on the plane $z = 1$ and it is defined as the projection of the 3-D curve onto the plane $z = 1$. One way to consider

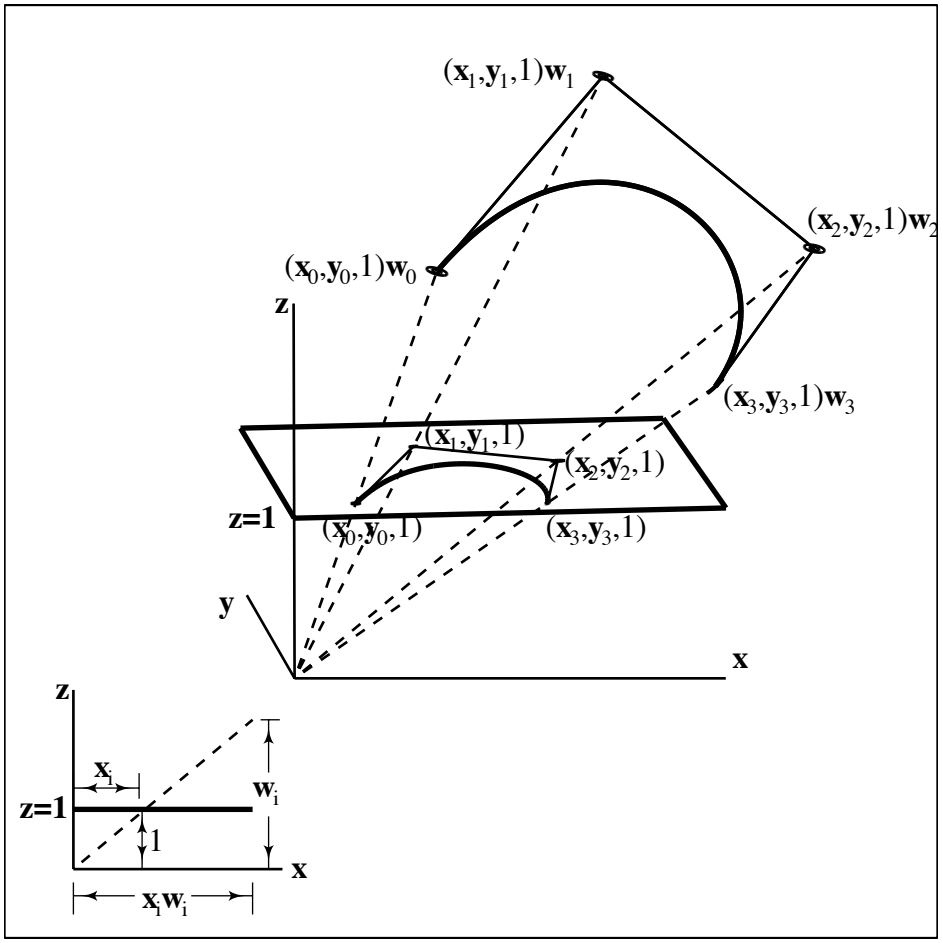


Figure 2.14: Rational curve as the projection of a 3-D curve.

this is to imagine a funny looking cone whose vertex is at the origin and which contains the 3-D curve. In other words, this cone is the collection of all lines which contain the origin and a point on the curve. Then, the 2-D rational Bézier curve is the intersection of the cone with the plane $z = 1$.

What we see here can be viewed as the geometric interpretation of homogeneous coordinates discussed in Section 7.1.1. The 3D Bézier control points $w_i(x_i, y_i, 1) = (x_i w_i, y_i w_i, w_i)$ can be thought of as homogeneous coordinates (X, Y, Z) that correspond to 2D Cartesian coordinates $(\frac{X}{Z}, \frac{Y}{Z})$. If the 2-D rational Bézier curve has control points (x_i, y_i) with corresponding weights w_i, then the homogeneous (X, Y, Z) coordinates of the 3-D control points are $w_i(x_i, y_i, 1) = (x_i w_i, y_i w_i, w_i)$. Denote points on the 3-D curve using upper case variables $(X(t), Y(t), Z(t))$ and on the 2-D curve using lower case variables $(x(t), y(t))$. Then, any point on the 2-D rational Bézier curve can be computed by computing the corresponding point on the 3-D curve, $(X(t), Y(t), Z(t))$, and projecting it to the plane $z = 1$ by setting

$$x(t) = \frac{X(t)}{Z(t)}, \quad y(t) = \frac{Y(t)}{Z(t)}.$$

2.9.1 De Casteljau Algorithm and Degree Elevation on Rational Bézier Curves

The de Casteljau algorithm and degree elevation algorithms for polynomial Bézier curves extend easily to rational Bézier curves as follows. First, convert the rational Bézier curve into its corresponding polynomial 3D Bézier curve as discussed in Section 2.9. Next, perform the de Casteljau algorithm or degree elevation on the 3D polynomial Bézier curve. Finally, map the resulting 3D Bézier curve back to 2D. The Z coordinates of the control points end up as the weights of the control points for the 2D rational Bézier curve.

This procedure does not work for hodographs, since the derivative of a rational Bézier curve requires the use of the quotient rule for differentiation. However, Section 2.9.2 describes how to compute the first derivative at the endpoint of a rational Bézier curve and Section 2.9.3 describes how to compute the curvature at the endpoint of a rational Bézier curve. Used in conjunction with the de Casteljau algorithm, this enable us to compute the first derivative or curvature at any point on a rational Bézier curve.

2.9.2 First Derivative at the Endpoint of a Rational Bézier Curve

The derivative equation for a rational Bézier involves the quotient rule for derivatives — it does not work to simply compute the tangent vector and curvature for the three dimensional non-rational Bézier curve and then project that value to the (x, y) plane. For a degree n rational Bézier curve $\mathbf{P}_{[0,1]}(t)$,

$$x(t) = \frac{x_n(t)}{d(t)} =$$

$$\frac{w_0 x_0 \binom{n}{0}(1-t)^n + w_1 x_1 \binom{n}{1}(1-t)^{n-1}t + w_2 x_2 \binom{n}{2}(1-t)^{n-2}t^2 + \ldots}{w_0 \binom{n}{0}(1-t)^n + w_1 \binom{n}{1}(1-t)^{n-1}t + w_2 \binom{n}{2}(1-t)^{n-2}t^2 + \ldots};$$

$$y(t) = \frac{y_n(t)}{d(t)} =$$

$$\frac{w_0 y_0 \binom{n}{0}(1-t)^n + w_1 y_1 \binom{n}{1}(1-t)^{n-1}t + w_2 y_2 \binom{n}{2}(1-t)^{n-2}t^2 + \ldots}{w_0 \binom{n}{0}(1-t)^n + w_1 \binom{n}{1}(1-t)^{n-1}t + w_2 \binom{n}{2}(1-t)^{n-2}t^2 + \ldots}$$

the equation for the first derivative vector at $t = 0$ must be found by evaluating the following equations:

$$\dot{x}(0) = \frac{d(0)\dot{x}_n(0) - \dot{d}(0)x_n(0)}{d^2(0)}; \qquad \dot{y}(0) = \frac{d(0)\dot{y}_n(0) - \dot{d}(0)y_n(0)}{d^2(0)}$$

from which

$$\mathbf{P}'(0) = \frac{w_1}{w_0} n(\mathbf{P}_1 - \mathbf{P}_0). \tag{2.10}$$

For a rational curve $\mathbf{P}_{[t_0,t_1]}(t)$, the first derivative at $t = t_0$ is:

$$\mathbf{P}'(0) = \frac{w_1}{w_0} \frac{n}{t_1 - t_0} (\mathbf{P}_1 - \mathbf{P}_0). \tag{2.11}$$

The second derivative of a rational Bézier curve at its endpoint is

$$\mathbf{P}''(0) = \frac{n(n-1)}{(t_1 - t_0)^2} \frac{w_2}{w_0} (\mathbf{P}_2 - \mathbf{P}_0) - \frac{2n}{(t_1 - t_0)^2} \frac{w_1}{w_0} \frac{nw_1 - w_0}{w_0} (\mathbf{P}_1 - \mathbf{P}_0) \tag{2.12}$$

2.9.3 Curvature at an Endpoint of a Rational Bézier Curve

The curvature (denoted by κ) of a curve is a measure of how sharply it curves. The curvature of a circle of radius ρ is defined to be $\kappa = 1/\rho$. A straight line does not curve at all, and its curvature is zero. For other curves, the curvature is generally different at each point on the curve. The meaning of curvature in this general case is illustrated by Figure 2.15 which shows a Bézier curve and the *osculating circle* with respect to a point on the curve. The word "osculate" means to kiss, and the osculating circle is the circle that "best fits" the curve at a particular point in the following sense. An arbitrary line that passes through a point on a curve intersects the curve with multiplicity one; the tangent line at that point intersects the curve with multiplicity two. An osculating circle is a circle of radius ρ that intersects the curve with multiplicity at least three, and the curvature κ is defined to be $\kappa = 1/\rho$. (See Section 7.3.1 for a description of intersection multiplicity.) ρ is called the radius of curvature for the curve at that point.

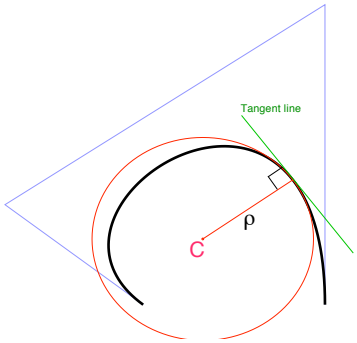


Figure 2.15: Osculating Circle.

In Figure 2.15, the center of the circle, $\mathbf{C}$, is located along the normal line a distance ρ from the point of contact.

The equation for the curvature of a parametric curve is

$$\kappa = \frac{|\dot{x}\ddot{y} - \dot{y}\ddot{x}|}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}$$

where the dots denote differentiation with respect to the curve parameter. If this equation is applied to an endpoint of a rational Bézier curve, a beautifully simple and geometrically meaningful equation results:

$$\kappa(t_0) = \frac{w_0 w_2}{w_1^2} \frac{n-1}{n} \frac{h}{a^2} \tag{2.13}$$

where n is the degree of the curve and a and h are as shown in Fig. 2.16 (a is the length of the first leg of the control polygon, and h is the perpendicular distance from $\mathbf{P}_2$ to the first leg of the control polygon). Curvature is independent of $[t_0, t_1]$.

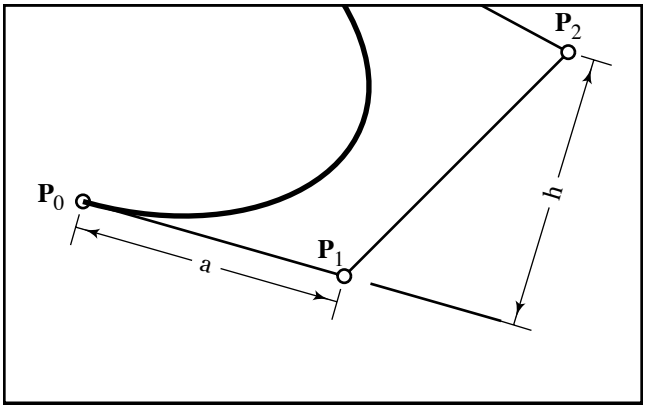


Figure 2.16: Endpoint curvature.

While is is mathematically possible to write down an equation for the first derivative vector and for the curvature of a rational Bézier curve as a function of t, such equations would be extremely complicated. The best way to find the curvature or first derivative vector at an arbitrary point along a rational Bézier curve is to first perform the de Casteljau algorithm (Section 2.3) to make the desired point an endpoint of the curve, and then apply (2.11) or (2.13).

Example A rational quadratic Bézier curve has control points and weights:

$$\mathbf{P}_0 = (0,0),\ w_0 = 1; \quad \mathbf{P}_1 = (4,3),\ w_1 = 2; \quad \mathbf{P}_2 = (0,5),\ w_2 = 4.$$

What is the curvature at $t = 0$?

Solution
The distance equation for the line $\mathbf{P}_0 - -\mathbf{P}_1$ is $\frac{3}{5}x - \frac{4}{5}y$, so plugging the Cartesian coordinates of $\mathbf{P}_2$ into this distance equation give $h = 4$. The value of a is found as the distance from $\mathbf{P}_0$ to $\mathbf{P}_1$, which is $\sqrt{4^2 + 3^2} = 5$. So, the curvature is

$$\kappa(0) = \frac{w_0 w_2}{w_1^2} \frac{n-1}{n} \frac{h}{a^2} = \frac{1 \cdot 4}{2^2} \frac{1}{2} \frac{4}{5^2} = \frac{2}{25}$$

2.10 Continuity

Two curve segments $\mathbf{P}_{[t_0,t_1]}$ and $\mathbf{Q}_{[t_1,t_2]}$ are said to be C^k continuous (or, to have k^{th} order parametric continuity) if

$$\mathbf{P}(t_1) = \mathbf{Q}(t_1),\ \mathbf{P}'(t_1) = \mathbf{Q}'(t_1),\ \ldots,\ \mathbf{P}^{(k)}(t_1) = \mathbf{Q}^{(k)}(t_1). \tag{2.14}$$

Thus, C^0 means simply that the two adjacent curves share a common endpoint. C^1 means that the two curves not only share the same endpoint, but also that they have the same tangent vector at their shared endpoint, in magnitude as well as in direction. C^2 means that two curves are C^1 and in addition that they have the same second order parametric derivatives at their shared endpoint, both in magnitude and in direction.

In Fig. 2.17, the two curves $\mathbf{P}_{[t_0,t_1]}(t)$ and $\mathbf{Q}_{[t_1,t_2]}(t)$ are at least C^0 because $\mathbf{p}_3 \equiv \mathbf{q}_0$. Furthermore,

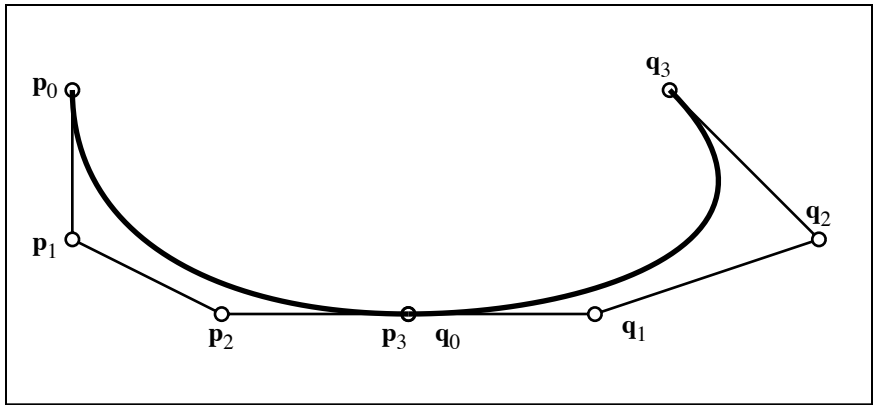


Figure 2.17: C^2 Bézier curves.

they are C^1 if

$$\frac{3}{t_1 - t_0}(\mathbf{p}_3 - \mathbf{p}_2) = \frac{3}{t_2 - t_1}(\mathbf{q}_1 - \mathbf{q}_0).$$

They are C^2 if they are C^1 and

$$\frac{6}{(t_1 - t_0)^2}(\mathbf{p}_3 - 2\mathbf{p}_2 + \mathbf{p}_1) = \frac{6}{(t_2 - t_1)^2}(\mathbf{q}_2 - 2\mathbf{q}_1 + \mathbf{q}_0).$$

A second method for describing the continuity of two curves, that is independent of their parametrization, is called geometric continuity and is denoted G^k. The conditions for geometric continuity (also known as *visual* continuity) are less strict than for parametric continuity. For G^0 continuity, we simply require that the two curves have a common endpoint, but we do not require that they have the same parameter value at that common point. For G^1, we require that line segments $\mathbf{p}_2 - \mathbf{p}_3$ and $\mathbf{q}_0 - \mathbf{q}_1$ are collinear, but they need not be of equal length. This means that they have a common tangent line, though the magnitude of the tangent vector may be different. G^2 (second order geometric continuity) means that the two neighboring curves have the same tangent line and also the same center of curvature at their common boundary. A general definition of G^n is that two curves that are G^n can be reparametrized (see Section 2.12) to make them be C^n. Two curves which are C^n are also G^n, as long as (2.15) holds.

$$\mathbf{P}(t_1) = \mathbf{Q}(t_1),\ \mathbf{P}'(t_1) = \mathbf{Q}'(t_1) \neq 0,\ \ldots,\ \mathbf{P}^{(k)}(t_1) = \mathbf{Q}^{(k)}(t_1) \neq 0. \tag{2.15}$$

Second order continuity (C^2 or G^2) is often desirable for both physical and aesthetic reasons. One reason that cubic NURBS curves and surfaces are an industry standard in CAD is that they are C^2. Two surfaces that are G^1 but not G^2 do not have smooth reflection lines. Thus, a car body made up of G^1 surfaces will not look smooth in a show room. Railroad cars will jerk where G^1 curves meet.

2.11 Circular Arcs

Circular arcs can be exactly represented using rational Bézier curves. Figure 2.18 shows a circular

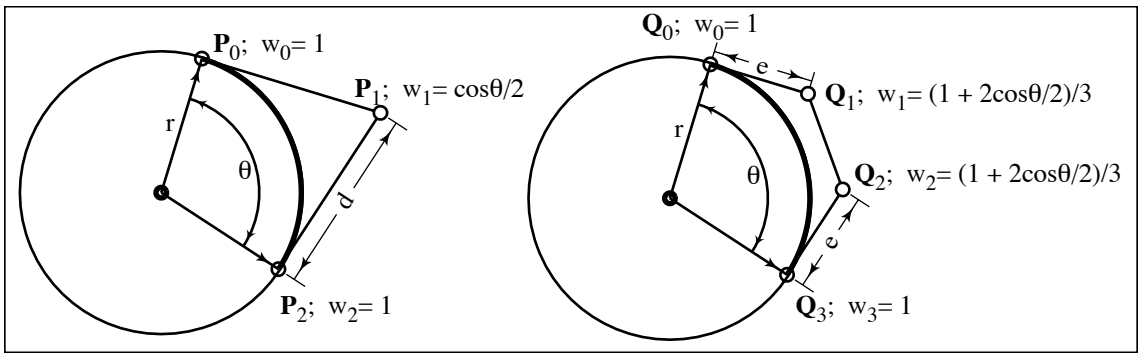


Figure 2.18: Circular arcs.

arc as both a degree two and a degree three rational Bézier curve. Of course, the control polygons are tangent to the circle. The degree three case is a simple degree elevation of the degree two case. The length e is given by

$$e = \frac{2\sin\frac{\theta}{2}}{1+2\cos\frac{\theta}{2}}r.$$

The degree two case has trouble when θ approaches 180° since $\mathbf{P}_1$ moves to infinity, although this can be remedied by just using homogeneous coordinates. The degree three case has the advantage that it can represent a larger arc, but even here the length e goes to infinity as θ approaches 240°. For large arcs, a simple solution is to just cut the arc in half and use two cubic Bézier curves. A complete circle can be represented as a degree five Bézier curve as shown in Figure 2.19. Here, the weights are $w_0 = w_5 = 1$ and $w_1 = w_2 = w_3 = w_4 = \frac{1}{5}$.

2.12 Reparametrization of Bézier Curves

Reparametrization of a curve means to change how a curve is parametrized, i.e., to change which parameter value is assigned to each point on the curve. Reparametrization can be performed by a parameter substitution. Given a parametric curve $\mathbf{P}(t)$, if we make the substitution $t = f(s)$, we end up with a curve $\mathbf{Q}(s)$ which is geometrically identical to $\mathbf{P}(t)$. For example, let

$$\mathbf{P}(t) = (t^2 + t - 1, 2t^2 - t + 3), \quad t = 2s + 1$$

yields

$$\mathbf{Q}(s) = ((2s+1)^2 + (2s+1) - 1, 2(2s+1)^2 - (2s+1) + 3) = (4s^2 + 6s + 1, 8s^2 + 6s + 4).$$

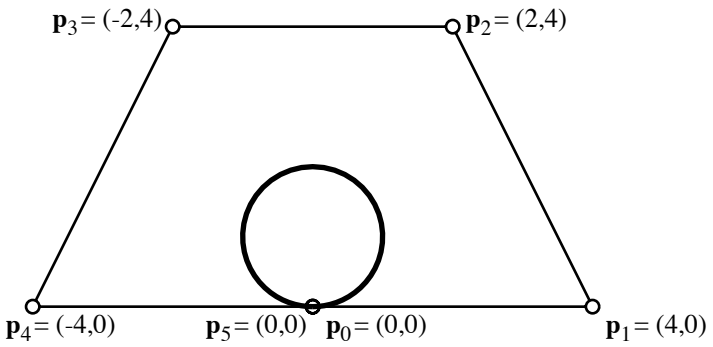


Figure 2.19: Circle as Degree 5 Rational Bézier Curve.

The curves $\mathbf{P}(t)$ and $\mathbf{Q}(s)$ consist of the same points (if the domains are infinite) but the correspondence between parameter values and points on the curve are different.

A simple (and obvious) way to reparametrize a Bézier curve $\mathbf{P}_{[t_0,t_1]}(t)$ is to simply change the values of t_0 and t_1 without changing the control points. Thus, two curves $\mathbf{P}_{[t_0,t_1]}(t)$ and $\mathbf{Q}_{[s_0,s_1]}(s)$ that have the same degree, control point positions, and weights, but for which $t_0 \neq s_0$ and/or $t_1 \neq s_1$ are reparametrizations of each other. The reparametrization function is

$$t = \frac{t_0 s_1 - s_0 t_1}{s_1 - s_0} + \frac{t_1 - t_0}{s_1 - s_0} s, \quad \text{or } s = \frac{s_0 t_1 - t_0 s_1}{t_1 - t_0} + \frac{s_1 - s_0}{t_1 - t_0} t.$$

Example
Bézier curves $\mathbf{P}_{[0,1]}(t)$ and $\mathbf{Q}_{[5,9]}(s)$ have the same degree, control point positions, and weights. The curves look the same, but have different parametrizations. For example,

$$\mathbf{P}(0) = \mathbf{Q}(5), \quad \mathbf{P}(1) = \mathbf{Q}(9), \quad \mathbf{P}(.5) = \mathbf{Q}(7).$$

In general,

$$\mathbf{P}(t) \equiv \mathbf{Q}(5 + 4t) \text{ and } \mathbf{Q}(s) \equiv \mathbf{P}(-\frac{5}{4} + \frac{1}{4}s).$$

A rational parametric curve can be reparametrized with the substitution $t = f(u)/g(u)$. In this case, it is possible to perform a rational-linear reparametrization which does not change the endpoints of the curve segment. If we let

$$t \;=\; \frac{a(1-u) \;+\; bu}{c(1-u) \;+\; du}$$

and want $u = 0$ when $t = 0$ and $u = 1$ when $t = 1$, then $a = 0$ and $b = d$. Since we can scale then numerator and denominator without affecting the reparametrization, set $c = 1$ and we are left with

$$t \;=\; \frac{bu}{(1-u) \;+\; bu}$$

A rational Bézier curve

$$\mathbf{X}(t) \;=\; \frac{\binom{n}{0} w_0 \mathbf{P}_0 (1-t)^n \;+\; \binom{n}{1} w_1 \mathbf{P}_1 (1-t)^{n-1} t \;+\; \ldots \;+\; \binom{n}{n} w_n \mathbf{P}_n t^n}{\binom{n}{0} w_0 (1-t)^n \;+\; \binom{n}{1} w_1 (1-t)^{n-1} t \;+\; \ldots \;+\; \binom{n}{n} w_n t^n}$$

can be raparametrized without changing its endpoints by making the substitutions

$$t \;=\; \frac{bu}{(1-u)\;+\;bu}, \quad (1-t) \;=\; \frac{(1-u)}{(1-u)\;+\;bu}.$$

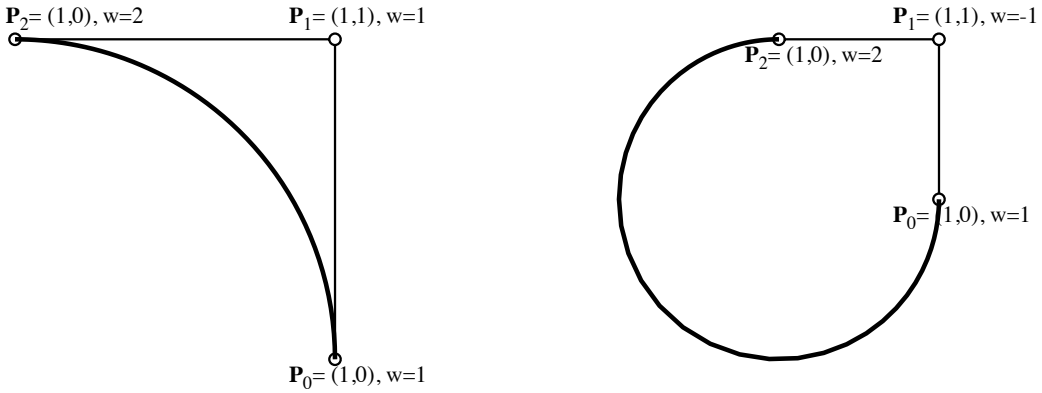
After multiplying numerator and denominator by $((1-u)\;+\;bu)^n$, we obtain

$$\mathbf{X}(u) \;=\; \frac{\binom{n}{0}(b^0 w_0)\mathbf{P}_0(1-u)^n \;+\; \binom{n}{1}(b^1 w_1)\mathbf{P}_1(1-u)^{n-1}u \;+\; \ldots \;+\; \binom{n}{n}(b^n w_n)\mathbf{P}_n u^n}{\binom{n}{0}b^0 w_0(1-u)^n \;+\; \binom{n}{1}b^1 w_1(1-u)^{n-1}u \;+\; \ldots \;+\; \binom{n}{n}b^n w_n u^n}$$

In other words, if we scale the weights w_i by b^i, the curve will not be changed!

Negative Weights

Normally, negative weights are discouraged for Bézier curves because they can cause a violation of the convex hull property. However, an interesting situation arises if we choose $b = -1$ for the rational reparametrization: the signs of the weights alternate, and the curve for $u \in [0,1]$ is the curve $t \in (\infty, 0] \cup [1, \infty)$. If the original curve is a quarter circle, the reparametrized curve will be the rest of the circle, as shown in Figure 2.20.



(a) Quarter circle.

(b) After reparametrization with $b = -1$.

Figure 2.20: Circle with negative weight.

2.13 Advantages of Rational Bézier Curves

As we have seen, rational Bézier curves have several advantages over polynomial Bézier curves. We here emphasize them.

1. Rational Bézier curves can represent conic sections exactly.
2. If you perform a perspective projection of a Bézier curve (either a polynomial or rational), the result is a rational Bézier curve.

3. Rational Bézier curves permit some additional shape control: by changing the weights, you change the shape of the curve.

4. It is possible to reparametrize the curve by simply changing the weights in a specific manner.

2.14 Explicit Bézier Curves

An explicit Bézier curve $\mathbf{P}_{[t_0,t_1]}(t)$ is one for which $x \equiv t$. For a polynomial Bézier curve, this happens when the x-coordinates of the control points are evenly spaced between t_0 and t_1. That is, $\mathbf{P}_i = (\frac{t_0(n-i)+t_1 i}{n}, y_i),\ i = 0, \ldots, n$. Such a Bézier curve takes on the form of a function

$$\begin{aligned} x &= t \\ y &= f(t) \end{aligned}$$

or simply

$$y = f(x).$$

An explicit Bézier curve is sometimes called a non-parametric Bézier curve. It is just a polynomial function expressed in the Bernstein polynomial basis. Figure 2.21 shows a degree five explicit Bézier curve over the domain $[t_0, t_1] = [0, 1]$.

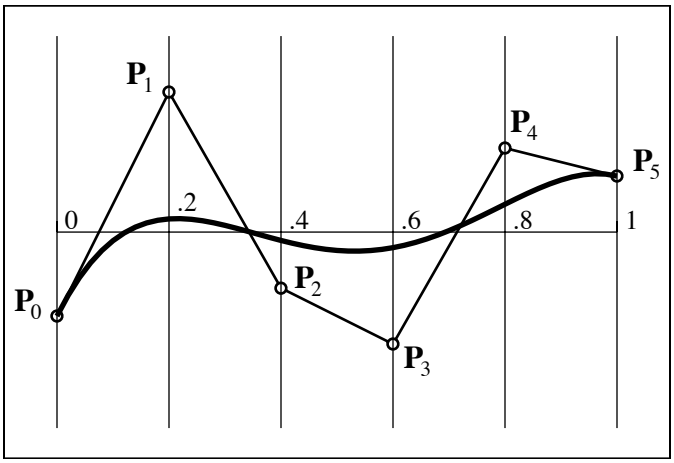


Figure 2.21: Explicit Bézier curve.

It is also possible to create an explicit rational Bézier curve. However, the representation of a degree-n rational function

$$x = t; \quad y = \frac{\sum_{i=0}^{n} w_i y_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}$$

as an explicit Bézier curve is degree $n+1$ because we must have

$$x = t = t\frac{\sum_{i=0}^{n} w_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}$$

which is degree $n+1$. Since $x(t)$ is degree $n+1$, we must likewise degree elevate $y(t)$.

2.15 Integrating Bernstein polynomials

Recall that the hodograph (first derivative) of a Bézier curve is easily found by simply differencing adjacent control points (Section 2.7). It is equally simple to compute the integral of a Bernstein polynomial. Since the integral of a polynomial in Bernstein form

$$p(t) = \sum_{i=0}^{n} p_i B_i^n(t) \tag{2.16}$$

is that polynomial whose derivative is $p(t)$. If the desired integral is a degree $n+1$ polynomial in Bernstein form

$$q(t) = \sum_{i=0}^{n+1} q_i B_i^{n+1}(t), \tag{2.17}$$

we have

$$p_i = (n+1)(q_{i+1} - q_i). \tag{2.18}$$

Hence, $q_0 = 0$ and

$$q_i = \frac{\sum_{j=0}^{i-1} p_j}{n+1}, \quad i = 1, n+1. \tag{2.19}$$

Note that if $p(t)$ is expressed as an explicit Bézier curve, $q(t)$ can be interpreted as the area under $p(t)$ between the lines $x = 0$ and $x = t$. Thus, the entire area under an explicit Bézier curve can be computed as simply the average of the control points! This is so because

$$q(1) = q_{n+1} = \frac{\sum_{j=0}^{n} p_j}{n+1}. \tag{2.20}$$

Chapter 3

Polynomial Evaluation and Basis Conversion

3.1 Horner's Algorithm in Power Basis

Horner's algorithm can evaluate a power basis polynomial

$$y(t) = \sum_{i=0}^{n} p_i t^i$$

using n multiplies and adds. The trick is to rewrite the polynomial in nested form:

$$y(t) = (\cdots((p_n t + p_{n-1})t + p_{n-2})t + \cdots + p_1)t + p_0$$

This can be written

$$\begin{aligned} h_n &= p_n \\ h_{i-1} &= t \cdot h_i + p_{i-1}, \quad i = n, \ldots, 1 \\ y(t) &= h_0 \end{aligned}$$

or, as a C funtion,

```
float p_horner(float *p,int n, float t)
{
float h;
int i;

   h = p[n];
   for(i=n-1; i>=0; i--)
      h = t*h + p[i];
   return  h;
}
```

This is the fastest algorithm for evaluating a power basis polynomial at a single point.

For evaluating a polynomial at several evenly spaced intervals ($f(0)$, $f(a)$, $f(2a)$, $f(3a)$, etc.) forward differencing works faster (see Chapter 4) but suffers from numerical instability.

3.2 Horner's Algorithm in Bernstein Basis

We have seen that one way to evaluate a Bézier curve (that is, to compute the Cartesian coordinates of a point on a Bézier curve) is to use the de Casteljau algorithm. Another option is to convert it to power basis (see Section 3.3) and then use Horner's algorithm in Section 3.1. Alternatively, Horner's algorithm can be modified to work directly on Bernstein polynomials. We can use the recurrence relation for binomial coefficients

$$\binom{n}{i} = \frac{n-i+1}{i}\binom{n}{i-1}$$

to derive a nested multiplication which does not involve the binomial coefficients directly:

```
float b_horner(float *b, int n, float t)
{
float u, bc, tn, tmp;
int i;

   u = 1.0 - t;
   bc = 1;
   tn = 1;
   tmp = b[0]*u;
   for(i=1; i<n; i++){
      tn = tn*t;
      bc = bc*(n-i+1)/i;
      tmp = (tmp + tn*bc*b[i])*u;
   }
    return (tmp + tn*t*b[n]);
}
```

3.3 Basis Conversion

A fundamental problem in CAGD is that of polynomial basis conversion. We here consider the common case of conversion between power and Bernstein bases. For example, given a Bézier curve of any degree, we want to find the equivalent power basis equation and vice versa.

Denote a polynomial in Bernstein form

$$B(t) = \sum_{i=0}^{n} b_i \binom{n}{i} (1-t)^{n-i} t^i$$

and a polynomial in power form

$$P(t) = \sum_{i=0}^{n} p_i t^i.$$

The problem we consider is, given the b_i of a polynomial in Bernstein basis, find the p_i for the equivalent power basis polynomial (Bernstein to power basis conversion) or given the p_i find the b_i (power to Bernstein conversion).

An elegant solution to this problem can be obtained by performing a Taylor's series expansion. The Taylor's expansion of $B(t)$ is:

$$B(t) \equiv B(0) + B'(0)t + \frac{B''(0)t^2}{2} + \ldots + \frac{B^{(i)}(0)t^i}{i} + \ldots.$$

A power basis polynomial is equivalent to a Bernstein basis polynomial ($P(t) \equiv B(t)$) if and only if

$$\frac{P^{(i)}(0)t^i}{i!} \equiv \frac{B^{(i)}(0)t^i}{i!}, \quad i = 0, \ldots, n.$$

But, for power basis,

$$\frac{P^{(i)}(0)}{i!} = p_i$$

so

$$p_i = \frac{B^{(i)}(0)}{i!}, \ i = 0, \ldots, n. \tag{3.1}$$

The terms $B^{(i)}(0)$ can be found by setting up a difference table. Remember from our study of hodographs that the coefficients of the derivative of a polynomial in Bernstein form (for example, the x or y component of a Bézier curve) are:

$$n(b_1 - b_0), \quad n(b_2 - b_1), \ldots, n(b_n - b_{n-1}).$$

The coefficients of the second derivative are:

$$n(n-1)(b_2 - 2b_1 + b_0), \quad n(n-1)(b_3 - 2b_2 + b_1), \ldots, n(n-1)(b_n - 2b_{n-1} + b_{n-2}).$$

Since $B(0) = \sum_{i=0}^{n} b_i \binom{n}{i}(1-0)^{n-i}0^i = b_0$, we have

$$B'(0) = n(b_1 - b_0), \quad B''(0) = n(n-1)(b_2 - 2b_1 + b_0)$$

$$B^{(i)}(0) = n(n-1)\cdots(n-i+1)\sum_{j=0}^{i}(-1)^{(i-j+1)}\binom{i}{j}b_j.$$

This can be written more neatly if we define the recurrence

$$b_i^j = b_{i+1}^{j-1} - b_i^{j-1}$$

with $b_i^0 \equiv b_i$. Then

$$B^{(i)}(0) = n(n-1)\cdots(n-i+1)b_0^i = \frac{n!}{(n-i)!}b_0^i.$$

From equation 3.1,

$$p_i = \frac{n!}{(n-i)!i!}b_0^i = \binom{n}{i}b_0^i.$$

Thus, the problem reduces to one of finding the values b_0^i. This is easily done using a difference table:

$b_0^0 = b_0 = p_0$	$b_1^0 = b_1$	$\ldots$	$b_n^0 = b_n$
$b_0^1 = b_1^0 - b_0^0 = p_1/\binom{n}{1}$	$b_1^1 = b_2^0 - b_1^0$	$\ldots$	$b_n^0 = b_n^0 - b_{n-1}^0$
$b_0^2 = b_1^1 - b_0^1 = p_2/\binom{n}{2}$	$b_1^2 = b_2^1 - b_1^1$	$\ldots$	$b_n^2 = b_n^1 - b_{n-1}^1$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$b_0^{n-1} = b_1^{n-2} - b_0^{n-2} = p_{n-1}/\binom{n}{n-1}$	$b_1^{n-1} = b_2^{n-2} - b_1^{n-2}$		
$b_0^n = b_1^{n-1} - b_0^{n-1} = p_n$			

Thus, to perform Bernstein to power basis conversion, load the Bernstein coefficients into the top row and compute the difference table. Scale the left column by $\binom{n}{i}$, and you have the power coefficients.

To perform power to Bernstein conversion, divide the power coefficients by $\binom{n}{i}$, load them into the left column, compute the difference table backwards, and read the Bernstein coefficients off the top row.

3.3.1 Example

Convert to power basis the degree 4 Bernstein form polynomial with coefficients $(1, 3, 4, 6, 8)$. This is done by setting up the difference table

```
 1    3    4    6    8
 2    1    2    2
-1    1    0
 2   -1
-3
```

so the power coefficient are taken from the left column, times the binomial coefficients:

$$p_0 = 1\binom{4}{0} = 1$$

$$p_1 = 2\binom{4}{1} = 8$$

$$p_2 = -1\binom{4}{2} = -6$$

$$p_3 = 2\binom{4}{3} = 8$$

$$p_4 = -3\binom{4}{4} = -3$$

3.3.2 Closed Form Expression

The conversion from Bernstein to power basis can be written concisely as follows:

$$p_i = \sum_{k=0}^{i} b_k \binom{n}{i}\binom{i}{k}(-1)^{i-k}.$$

Chapter 4

Forward Differencing

Horner's algorithm is the fastest method for evaluating a polynomial at a single point. For a degree n polynomial, it requires n multiplies and n adds.

If a polynomial is to be evaluated at several evenly spaced values $t, t+\delta, t+2\delta, \ldots, t+k\delta$, the fastest method is to use *forward differences*.

Consider a degree 1 polynomial

$$f(t) = a_0 + a_1 t.$$

The difference between two adjacent function values is

$$\Delta_1(t) = f(t+\delta) - f(t) = [a_0 + a_1(t+\delta)] - [a_0 + a_1 t] = a_1\delta.$$

Thus, $f(t)$ can be evaluated at several evenly spaced points using the algorithm:

$\Delta_1 = a_1\delta$
$t_0 = 0$
$f(0) = a_0$
for $i = 1$ **to** k **do**
$t_i = t_{i-1} + \delta$
$f(t_i) = f(t_{i-1}) + \Delta_1$
endfor

Thus, each successive evaluation requires only one add, as opposed to one add and one multiply for Horner's algorithm.

This idea extends to polynomials of any degree. For the quadratic case,

$$f(t) = a_0 + a_1 t + a_2 t^2.$$

The difference between two adjacent function values is

$$\Delta_1(t) = f(t+\delta) - f(t) = [a_0 + a_1(t+\delta) + a_2(t+\delta)^2] - [a_0 + a_1 t + a_2 t^2]$$

$$\Delta_1(t) = a_1\delta + a_2\delta^2 + 2a_2 t\delta.$$

We can now write

$t_0 = 0$
$f(0) = a_0$
for $i = 1$ **to** k **do**
$t_i = t_{i-1} + \delta$
$\Delta_1(t_i) = a_1\delta + a_2\delta^2 + 2a_2t_{i-1}\delta$
$f(t_i) = f(t_{i-1}) + \Delta_1(t_{i-1})$
endfor

In this case, $\Delta(t)$ is a linear polynomial, so we can evaluate it as above, by defining

$$\Delta_2(t) = \Delta_1(t + \delta) - \Delta_1(t) = 2a_2\delta^2$$

and our algorithm now becomes

$t_0 = 0$
$f(0) = a_0$
$\Delta_1 = a_1\delta + a_2\delta^2$
$\Delta_2 = 2a_2\delta^2$
for $i = 1$ **to** k **do**
$t_i = t_{i-1} + \delta$
$f(t_i) = f(t_{i-1}) + \Delta_1$
$\Delta_1 = \Delta_1 + \Delta_2$
endfor

It should be clear that for a degree n polynomial, each successive evaluation requires n adds and no multiplies! For a cubic polynomial

$$f(t) = a_0 + a_1t + a_2t^2 + a_3t^3,$$

the forward difference algorithm becomes

$t_0 = 0$
$f(0) = a_0$
$\Delta_1 = a_1\delta + a_2\delta^2 + a_3\delta^3$
$\Delta_2 = 2a_2\delta^2 + 6a_3\delta^3$
$\Delta_3 = 6a_3\delta^3$
for $i = 1$ **to** k **do**
$t_i = t_{i-1} + \delta$
$f(t_i) = f(t_{i-1}) + \Delta_1$
$\Delta_1 = \Delta_1 + \Delta_2$
$\Delta_2 = \Delta_2 + \Delta_3$
endfor

Several questions remain. First, what are the initial values for the Δ_i if we want to start at some value other than $t = 0$. Second, what is a general equation for the Δ_i for a general degree n polynomial $f(t)$. Also, what if our polynomial is not in power basis.

These questions can be answered almost trivially by observing the following. Since $t_{i+1} = t_i + \delta$, we have

$$\Delta_1(t_i) = f(t_{i+1}) - f(t);$$
$$\Delta_j(t_i) = \Delta_{j-1}(t_{i+1}) - \Delta_{j-1}(t_i), \quad j = 2, \ldots, n;$$
$$\Delta_n(t_i) = \Delta_n(t_{i+1}) = \Delta_n(t_{i+k}) = \text{a constant}$$
$$\Delta_{n+1} = 0$$

Thus, our initial values for $\Delta_j(t_i)$ can be found by simply computing $f(t_i),\ f(t_{i+1}), \ldots, f(t_{i+n})$ and from them computing the initial differences. This lends itself nicely to a table. Here is the table for a degree four case:

$f(t_i)$	$f(t_{i+1})$	$f(t_{i+2})$	$f(t_{i+3})$	$f(t_{i+4})$
$\Delta_1(t_i)$	$\Delta_1(t_{i+1})$	$\Delta_1(t_{i+2})$	$\Delta_1(t_{i+3})$	
$\Delta_2(t_i)$	$\Delta_2(t_{i+1})$	$\Delta_2(t_{i+2})$		
$\Delta_3(t_i)$	$\Delta_3(t_{i+1})$			
$\Delta_4(t_i)$				
0	0	0	0	0

To compute $f(t_{i+5})$, we simply note that every number R in this table, along with its right hand neighbor R_{right} and the number directly beneath it R_{down} obey the rule

$$R_{right} = R + R_{down}.$$

Thus, we can simply fill in the values

$$\Delta_4(t_{i+1}) = \Delta_4(t_i) + 0$$
$$\Delta_3(t_{i+2}) = \Delta_3(t_{i+1}) + \Delta_4(t_{i+1})$$
$$\Delta_2(t_{i+3}) = \Delta_2(t_{i+2}) + \Delta_3(t_{i+2})$$
$$\Delta_1(t_{i+4}) = \Delta_1(t_{i+3}) + \Delta_2(t_{i+3})$$
$$f(t_{i+5}) = f(t_{i+4}) + \Delta_1(t_{i+4})$$

Note that this technique is independent of the basis in which $f(t)$ is defined. Thus, even if it is defined in Bernstein basis, all we need to do is to evaluate it $n + 1$ times to initiate the forward differencing.

For example, consider the degree 4 polynomial for which $f(t_i) = 1$, $f(t_{i+1}) = 3$, $f(t_{i+2}) = 2$, $f(t_{i+3}) = 5$, $f(t_{i+4}) = 4$. We can compute $f(t_{i+5}) = -24$, $f(t_{i+6}) = -117$, and $f(t_{i+7}) = -328$ from the following difference table:

$t:$	t_i	t_{i+1}	t_{i+2}	t_{i+3}	t_{i+4}	t_{i+5}	t_{i+6}	t_{i+7}
$f(t):$	1	3	2	5	4	-24	-117	-328
$\Delta_1(t):$	2	-1	3	-1	-28	-93	-211	
$\Delta_2(t):$	-3	4	-4	-27	-65	-118		
$\Delta_3(t):$	7	-8	-23	-38	-53			
$\Delta_4(t):$	-15	-15	-15	-15	-15			
$\Delta_5(t):$	0	0	0	0	0	0	0	0

Example
For a certain cubic polynomial $f(t)$, we have:

$$f(1) = 1; \quad f(2) = 2; \quad f(3) = 4; \quad f(5) = 15.$$

Solve for $f(4)$ using forward differencing.

Solution

$t:$	1	2	3	4	5
$f(t):$	1	2	4	x	15
$\Delta_1(t):$	1	2	$x-4$	$15-x$	
$\Delta_2(t):$	1	$x-6$	$19-2x$		
$\Delta_3(t):$	$x-7$	$25-3x$			
$\Delta_4(t):$	$32-4x$				

For a cubic polynomial, $\Delta_4(t) = 0$, so $32 - 4x = 0$ and $x = 8$.

4.1 Choosing δ

This raises the important question of how to select an appropriate value for δ when using forward differencing to plot a curve. One way to determine δ is so that distance from the the curve to its polygonal approximation lies within a tolerance. A discussion of how to chose δ that will satisfy such a requirement is found in Section 10.6.

A second criterion that might be used to choose δ arises in the problem of rasterizing a curve. This means to "turn on" a contiguous series of pixels that the curve passes through, without any gaps. If the control points of the degree n Bézier curve are $\mathbf{P}_i = (x_i, y_i)$ in *pixel coordinates*, then let

$$d = \max\{x_i - x_{i-1}, y_i - y_{i-1}\} \quad i = 1, \ldots, n$$

If you now evaluate the curve at $n * d + 1$ evenly spaced values of t and paint each resulting pixel, there will be no gaps in the drawing of the curve. Another way to say this, compute

$$\delta = \frac{1}{n * d}.$$

Then, compute the points $\mathbf{P}(i * \delta)$, $i = 0 \ldots, n * d$. Note that $n * d$ is an upper bound on the magnitude of the x or y component of the first derivative of the curve.

Chapter 5

Properties of Blending Functions

We have just studied how the Bernstein polynomials serve very nicely as blending functions. We have noted that a degree n Bézier curve always begins at $\mathbf{P}_0$ and ends at $\mathbf{P}_n$. Also, the curve is always tangent to the control polygon at $\mathbf{P}_0$ and $\mathbf{P}_n$.

Other popular blending functions exist for defining curves. In fact, you can easily make up your own set of blending functions. And by following a few simple rules, you can actually create a new type of free-form curve which has desirable properties.

Consider a set of control points $\mathbf{P}_i$, $i = 0, ..., n$ and blending functions $f_i(t)$ which define the curve

$$\mathbf{P}(t) = \sum_{i=0}^{n} f_i(t)\mathbf{P}_i.$$

We can select our blending functions such that the curve has any or all of the following properties:

1. **Coordinate system independence**. This means that the curve will not change if the coordinate system is changed. In other words, imagine that the control points are drawn on a piece of paper and we move that piece of paper around so that the (x, y) coordinates of the control points change. It would be nice if the curve did not change relative to the control points. Actually, if we were to pick an arbitrary set of blending functions, the curve *would* change. In order to provide coordinate system independence, the blending functions must form a *partition of unity*, which is math jargon that means that the blending functions sum identically sum to one:

   $$\sum_{i=0}^{n} f_i(t) \equiv 1.$$

   The property of coordinate system independence is also called *affine invariance*.

2. **Convex hull property**. The convex hull property was introduced in Section 2.5. This property exists in curves which are coordinate system independent and for which the blending functions are all non- negative:

   $$\sum_{i=0}^{n} f_i(t) \equiv 1; \quad f_i(t) \geq 0, \quad 0 \leq t \leq 1, \quad i = 0, ..., n$$

3. **Symmetry**. Curves which are symmetric do not change if the control points are ordered in reverse sequence. For a curve whose domain is $[0, 1]$, symmetry is assured if and only if

$$\sum_{i=0}^{n} f_i(t)\mathbf{P}_i \equiv \sum_{i=0}^{n} f_i(1-t)\mathbf{P}_{n-i}.$$

This holds if

$$f_i(t) = f_{n-i}(1-t).$$

For a curve whose domain is $[t_0, t_1]$, symmetry requires

$$\sum_{i=0}^{n} f_i(t)\mathbf{P}_i \equiv \sum_{i=0}^{n} f_i(t_1+t_2-t)\mathbf{P}_{n-i}.$$

4. **Variation Diminishing Property**. This is a property which is obeyed by Bézier curves and B-spline curves. It states that if a given straight line intersects the curve in c number of points and the control polygon in p number of points, then it will always hold that

$$c = p - 2j$$

where j is zero or a positive integer. This has the practical interpretation that a curve which

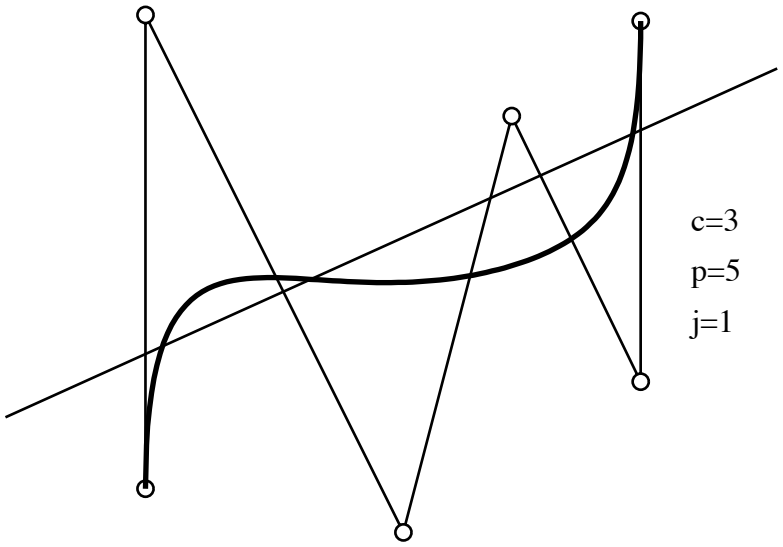


Figure 5.1: Variation Diminishing Property

obeys the variation diminishing property will "wiggle" no more than the control polygon.

The conditions under which a curve will obey the variation diminishing property are rather complicated. Suffice it to say that Bézier curves and B-spline curves obey this property, and most other curves do not.

5. **Linear Independence**. A set of blending functions is linearly independent if there do no exist a set of constants $c_0, \ldots, c_n$, not all zero, for which

$$\sum_{i=0}^{n} c_i f_i(t) \equiv 0.$$

Linearly independent blending functions are desirable for many reasons. For example,if they are not linearly independent, it is possible to express one blending function in terms of the other ones. This has the practical disadvantage that any given curve can be represented by infinitely many different control point positions. It also means that, for certain control point arrangements, the curve collapses to a single point, even though the control points are not all at that point.

If a set of blending functions are linearly independent, they can be called *basis functions*. Thus, Bernstein polynomials are basis functions for Bézier curves, and B-spline blending functions are basis functions.

6. **Endpoint Interpolation** If a curve over the domain $[t_0, t_1]$ is to pass through the first and last control points, as in the case of Bézier curves, the following conditions must be met:

$$f_0(t_0) = 1, \quad f_i(t_0) = 0, \ i = 1, ..., n$$

$$f_n(t_1) = 1, \quad f_i(t_1) = 0, \ i = 0, ..., n-1.$$

Any set of blending functions can be analyzed in terms of the above properties.

Example

Check which of the above-mentioned properties are obeyed by the sample curve

$$\mathbf{P}(t) = f_0(t)\mathbf{P}_0 + f_1(t)\mathbf{P}_1 + f_2(t)\mathbf{P}_2 = \frac{t^2 - 2t + 1}{t^2 + 1}\mathbf{P}_0 + \frac{2t - 2t^2}{t^2 + 1}\mathbf{P}_1 + \frac{2t^2}{t^2 + 1}\mathbf{P}_2$$

Answer:

This curve is **coordinate system independent** because

$$\frac{t^2 - 2t + 1}{t^2 + 1} + \frac{2t - 2t^2}{t^2 + 1} + \frac{2t^2}{t^2 + 1} = \frac{t^2 + 1}{t^2 + 1} \equiv 1.$$

The curve also obeys the **convex hull property** because its blending functions are non-negative for $t \in [0, 1]$. This is most easily seen by factoring the blending functions:

$$\frac{t^2 - 2t + 1}{t^2 + 1} = \frac{(t-1)^2}{t^2 + 1}$$

is non-negative for any real number because the square of a real number is always non-negative.

$$\frac{2t - 2t^2}{t^2 + 1} = \frac{2t(1-t)}{t^2 + 1}$$

is clearly non-negative for $t \in [0, 1]$.

The curve is **not symmetric** because

$$\frac{(1-t)^2 - 2(1-t) + 1}{(1-t)^2 + 1} = \frac{t^2}{t^2 - 2t + 2} \neq \frac{2t^2}{t^2 + 1}$$

Variation diminishing is very difficult to prove, and will not be attempted here.
Linear independence is assessed by checking if there exist constants c_0, c_1, c_2, not all zero, for which

$$c_0 \frac{t^2 - 2t + 1}{t^2 + 1} + c_1 \frac{2t - 2t^2}{t^2 + 1} + c_2 \frac{2t^2}{t^2 + 1} \equiv 0.$$

This is equivalent to

$$c_0(t^2 - 2t + 1) + c_1(2t - 2t^2) + c_2(2t^2) = (c_0 - 2c_1 + 2c_2)t^2 + (-2c_0 + 2c_1)t + c_0 \equiv 0$$

which can only happen if

$$c_0 - 2c_1 + 2c_2 = 0; \qquad -2c_0 + 2c_1 = 0; \qquad c_0 = 0.$$

There is no solution to this set of linear equations, other than $c_0 = c_1 = c_2 = 0$, so we conclude that these blending functions are linearly independent.
The curve does interpolate the endpoints, because

$$f_0(0) = 1, \quad f_1(0) = f_2(0) = 0; \qquad f_2(1) = 1, \quad f_0(1) = f_1(1) = 0.$$

The Bézier and B-spline curves are currently the most popular curve forms. Historically, other curve forms evolved independently at several different industrial sites, each faced with the common problem of making free-form curves accessible to designers. In this section, we will review three such curves: Timmer's Parametric Cubic, Ball's Cubic, and the Overhauser curve. Each of these curves is coordinate system independent and symmetric, but only Ball's cubic obeys the convex hull property.

5.1 Timmer's Parametric Cubic

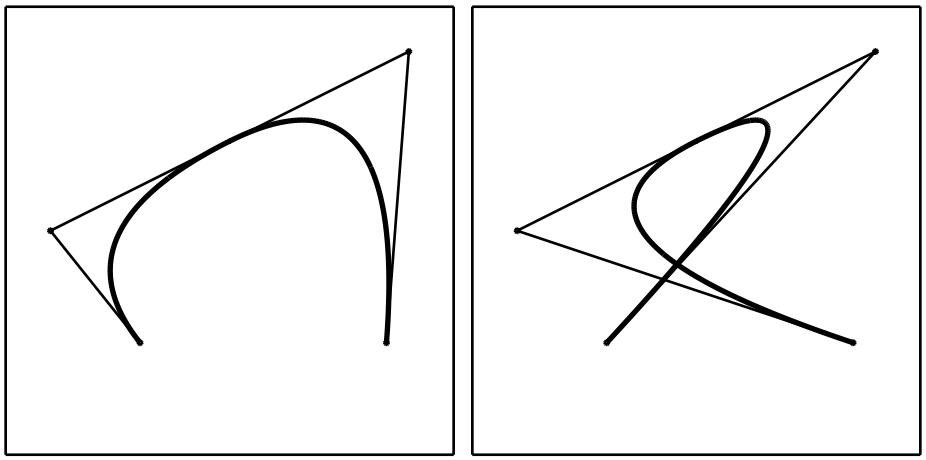


Figure 5.2: Timmer's PC

Timmer's Parametric Cubic (or PC) was created by Harry Timmer of McDonnell Douglas [Tim80]. It was modeled after the Bézier curve. Timmer felt that he could improve upon the Bézier curve

if he could make it follow the control polygon more tightly. This he did by forcing the curve to interpolate the endpoints of the control polygon and to be tangent to the control polygon at those points (just like Bézier curves) and in addition, he forced the curve to go through the midpoint of the line segment $\mathbf{P}_1$ - $\mathbf{P}_2$. The resulting blending functions are:

$$f_0(t) = (1-2t)(1-t)^2 = -2t^3 + 5t^2 - 4t + 1$$

$$f_1(t) = 4t(1-t)^2 = 4t^3 - 8t^2 + 4t$$

$$f_2(t) = 4t^2(1-t) = -4t^3 + 4t^2$$

$$f_3(t) = (2t-1)t^2 = 2t^3 - t^2$$

Figure 5.2 may mislead one into thinking that Timmer's curve is tangent to $\mathbf{P}_1 - \mathbf{P}_2$. This is not generally so (and is not exactly so even in Figure 5.2).

Example Problem
Given a Timmer curve with control points $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$, find the control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ of an equivalent cubic Bézier curve $\mathbf{P}(t)$.
Solution
Note that if $\mathbf{P}(t) \equiv \mathbf{Q}(t)$,then $\mathbf{P}(0) = \mathbf{Q}(0)$, $\mathbf{P}(1) = \mathbf{Q}(1)$, $\mathbf{P}'(0) = \mathbf{Q}'(0)$, and $\mathbf{P}'(1) = \mathbf{Q}'(1)$. This strategy provides a simple way to perform the conversion.

$$\mathbf{P}(0) = \mathbf{Q}(0) \rightarrow \mathbf{P}_0 = \mathbf{Q}_0.$$

$$\mathbf{P}(1) = \mathbf{Q}(1) \rightarrow \mathbf{P}_3 = \mathbf{Q}_3.$$

For derivatives, we differentiate the Timmer basis functions (it does not work to use the Bézier hodograph):

$$\mathbf{Q}'(t) = (-6t^2 + 10t - 4)\mathbf{Q}_0 + (12t^2 - 16t + 4)\mathbf{Q}_1 + (-12t^2 + 8t)\mathbf{Q}_2 + (6t^2 - 2t)\mathbf{Q}_3$$

so $\mathbf{Q}'(t) = 4(\mathbf{Q}_1 - \mathbf{Q}_0)$ and $\mathbf{Q}'(1) = 4(\mathbf{Q}_3 - \mathbf{Q}_2)$.

$$\mathbf{P}'(0) = \mathbf{Q}'(0) \rightarrow 3(\mathbf{P}_1 - \mathbf{P}_0) = 4(\mathbf{Q}_1 - \mathbf{Q}_0) \rightarrow \mathbf{Q}_1 = \frac{\mathbf{P}_0 + 3\mathbf{P}_1}{4}$$

$$\mathbf{P}'(1) = \mathbf{Q}'(1) \rightarrow 3(\mathbf{P}_3 - \mathbf{P}_2) = 4(\mathbf{Q}_3 - \mathbf{Q}_2) \rightarrow \mathbf{Q}_2 = \frac{\mathbf{P}_3 + 3\mathbf{P}_2}{4}$$

5.2 Ball's Rational Cubic

Alan Ball first published his cubic curve formulation in 1974 [Bal74]. Ball worked for the British Aircraft Corporation, and his cubic curve form was used in BAC's in-house CAD system.

Like Timmer's PC curve, Ball's cubic can be considered a variant of the cubic Bézier curve. It's distinguishing feature is that it handles conic sections as a special case in a natural way. The blending functions for the non-rational case are:

$$f_0(t) = (1-t)^2$$

$$f_1(t) = 2t(1-t)^2$$

$$f_2(t) = 2t^2(1-t)$$

$$f_3(t) = t^2$$

Notice that if $\mathbf{P}_1 = \mathbf{P}_2$, then the curve becomes a quadratic Bézier curve:

$$\begin{aligned}
&\mathbf{P}_0(1-t)^2 + \mathbf{P}_1 2t(1-t)^2 + \mathbf{P}_1 2t^2(1-t) + \mathbf{P}_3 t^2 \\
&= \mathbf{P}_0(1-t)^2 + \mathbf{P}_1[2t(1-t)^2 + 2t^2(1-t)] + \mathbf{P}_3 t^2 \\
&= \mathbf{P}_0(1-t)^2 + \mathbf{P}_1 2t(1-t) + \mathbf{P}_3 t^2
\end{aligned}$$

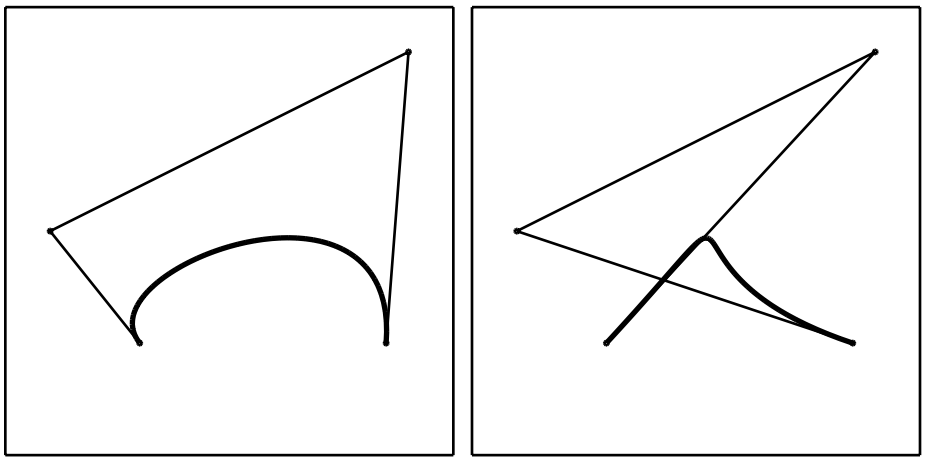


Figure 5.3: Ball's Cubic

5.3 Overhauser Curves

Overhauser curves were developed and used at Ford Motor Company [Ove68]. They are also known as cubic Catmull-Rom splines [CR74].

One complaint of Bézier curves is that the curve does not interpolate all of the control points. Overhauser curves do interpolate all control points in a piecewise string of curve segments. A single Overhauser curve is defined with the following blending functions:

$$\begin{aligned}
f_0(t) &= -\frac{1}{2}t + t^2 - \frac{1}{2}t^3 \\
f_1(t) &= 1 - \frac{5}{2}t^2 + \frac{3}{2}t^3 \\
f_2(t) &= \frac{1}{2}t + 2t^2 - \frac{3}{2}t^3 \\
f_3(t) &= -\frac{1}{2}t^2 + \frac{1}{2}t^3
\end{aligned}$$

A single Overhauser curve segment interpolates $\mathbf{P}_1$ and $\mathbf{P}_2$. Furthermore, the slope of the curve at $\mathbf{P}_1$ is only a function of $\mathbf{P}_0$ and $\mathbf{P}_2$ and the slope at $\mathbf{P}_2$ is only a function of $\mathbf{P}_1$ and $\mathbf{P}_3$:

$$\mathbf{P}'(0) = \frac{1}{2}(\mathbf{P}_2 - \mathbf{P}_0)$$

$$\mathbf{P}'(1) = \frac{1}{2}(\mathbf{P}_3 - \mathbf{P}_1)$$

This means that a second curve segment will be tangent to the first curve segment if its first three control points are identical to the last three control points of the first curve. This is illustrated in Fig. 5.4.

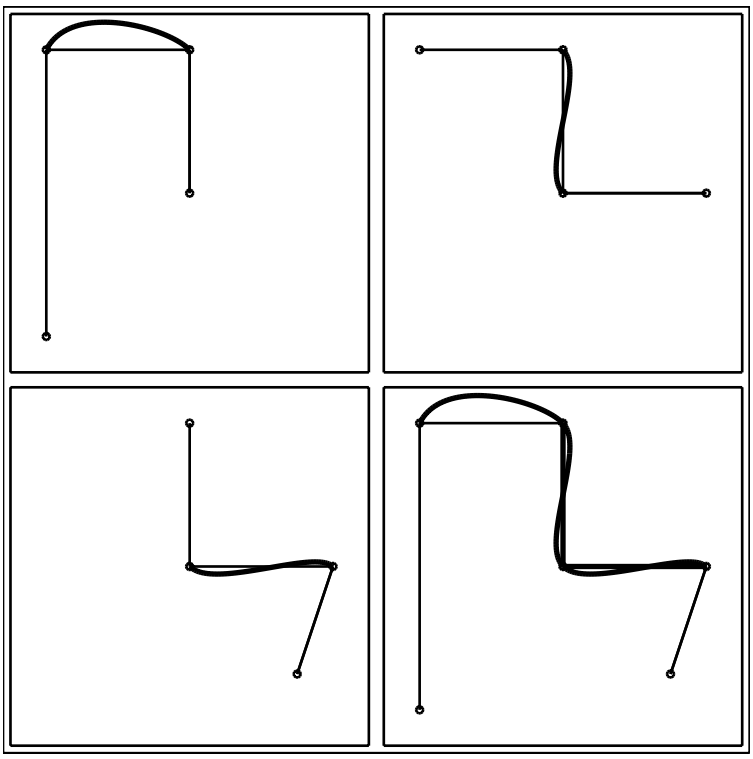


Figure 5.4: Overhauser curves

Chapter 6

B-Spline Curves

Most shapes are too complicated to define using a single Bézier curve. A spline curve is a sequence of curve segments that are connected together to form a single continuous curve. For example, a piecewise collection of Bézier curves, connected end to end, can be called a spline curve. Overhauser curves are another example of splines. The word "spline" is sometimes used as a verb, as in "Spline together some cubic Bézier curves." In approximation theory, spline is defined as a piecewise polynomial of degree n whose segments are C^{n-1}.

The word "spline" comes from the ship building industry, where it originally referred to a thin strip of wood which draftsmen would use like a flexible French curve. Metal weights (called ducks) were placed on the drawing surface and the spline was threaded between the ducks as in Figure 6.1. We know from structural mechanics that the bending moment M is an infinitely continuous function

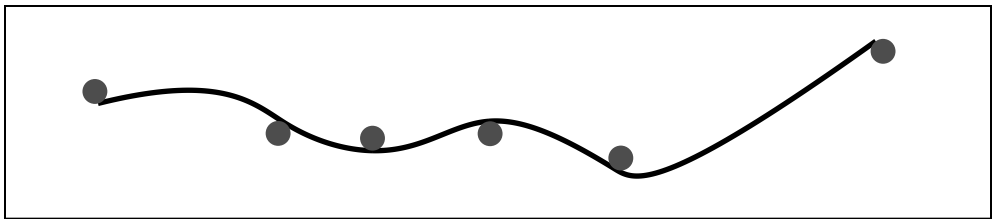


Figure 6.1: Spline and ducks.

along the spline except at a duck, where M is generally only C^0 continuous. Since the curvature of the spline is proportional to M ($\kappa = M/EI$), the spline is everywhere curvature continuous.

Curvature continuity is an important requirement for the ship building industry, as well as for many other applications. For example, railroad tracks are always curvature continuous, or else a moving train would experience severe jolts. Car bodies are G^2 smooth, or else the reflection of straight lines would appear to be G^0.

While C^1 continuity is straightforward to attain using Bézier curves (for example, popular design software such as Adobe Illustrator use Bézier curves and automatically impose tangent continuity as you sketch), C^2 and higher continuity is cumbersome. This is where B-spline curves come in. B-spline curves can be thought of as a method for defining a sequence of degree n Bézier curves that join automatically with C^{n-1} continuity, regardless of where the control points are placed.

Whereas an open string of m Bézier curves of degree n involve $nm + 1$ distinct control points (shared control points counted only once), that same string of Bézier curves can be expressed using

only $m+n$ B-spline control points (assuming all neighboring curves are C^{n-1}). The most important operation you need to understand to have a working knowledge of B-splines is how to extract the contituent Bézier curves. The traditional approach to teaching about B-Splines centered on basis functions and recurrence relations. Experience has shown that polar form (Section 6.1) and knot intervals (Section 6.13) provide students with such a working knowledge more quickly than recurrence relations.

6.1 Polar Form

Polar form, introduced by Dr. Lyle Ramshaw [Ram87, Ram89a, Ram89b], can be thought of as simply an alternative method of labeling the control points of a Bézier or B-Spline curve. These labels are referred to as *polar values*. These notes summarize the properties and applications of polar form, without delving into derivations. The interested student can study Ramshaw's papers.

All of the important algorithms for Bézier and B-spline curves can be derived from the following four rules for polar values.

1. For a degree n Bézier curve $\mathbf{P}_{[a,b]}(t)$, the control points are relabeled $\mathbf{P}_i = \mathbf{P}(u_1, u_2, \ldots u_n)$ where $u_j = a$ if $j \leq n - i$ and otherwise $u_j = b$. For a degree two Bézier curve $\mathbf{P}_{[a,b]}(t)$,

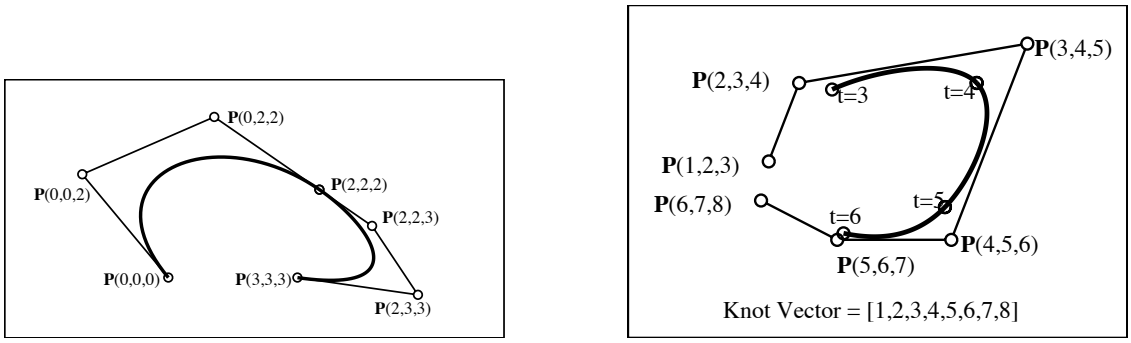
$$\mathbf{P}_0 = \mathbf{P}(a, a); \quad \mathbf{P}_1 = \mathbf{P}(a, b); \quad \mathbf{P}_2 = \mathbf{P}(b, b).$$

For a degree three Bézier curve $\mathbf{P}_{[a,b]}(t)$,

$$\mathbf{P}_0 = \mathbf{P}(a, a, a); \quad \mathbf{P}_1 = \mathbf{P}(a, a, b);$$

$$\mathbf{P}_2 = \mathbf{P}(a, b, b); \quad \mathbf{P}_3 = \mathbf{P}(b, b, b),$$

and so forth.



(a) Bézier curves with polar labels.

(b) B-Spline Curve with Polar Labels

Figure 6.2: Polar Labels.

Figure 6.2.a shows two cubic Bézier curves labeled using polar values. The first curve is defined over the parameter interval $[0, 2]$ and the second curve is defined over the parameter interval $[2, 3]$. Note that $\mathbf{P}(t, t, \ldots, t)$ is the point on a Bézier curve corresponding to parameter value t.

2. For a degree n B-spline with a *knot vector* (see Section 6.2) of

$$[t_1, t_2, t_3, t_4, \ldots],$$

the arguments of the polar values consist of groups of n adjacent knots from the knot vector, with the i^{th} polar value being $\mathbf{P}(t_i, \ldots, t_{i+n-1})$, as in Figure 6.2.b.
3. A polar value is symmetric in its arguments. This means that the order of the arguments can be changed without changing the polar value. For example,

$$\mathbf{P}(1,0,0,2) = \mathbf{P}(0,1,0,2) = \mathbf{P}(0,0,1,2) = \mathbf{P}(2,1,0,0), \ \textit{etc.}$$

4. Given $\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, a)$ and $\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, b)$ we can compute $\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, c)$ where c is any value:

$$\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, c) =$$
$$\frac{(b-c)\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, a) + (c-a)\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, b)}{b-a}$$

$\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, c)$ is said to be an *affine combination* of $\mathbf{P}(u_1, u_2, \ldots, u_{n-1}, a)$ and $\mathbf{P}(u_1,\ u_2,\ \ldots, u_{n-1}, b)$. For example,

$$\mathbf{P}(0,t,1) = (1-t) \times \mathbf{P}(0,0,1) + t \times \mathbf{P}(0,1,1),$$
$$\mathbf{P}(0,t) = \frac{(4-t) \times \mathbf{P}(0,2) + (t-2) \times \mathbf{P}(0,4)}{2},$$
$$\mathbf{P}(1,2,3,t) = \frac{(t_2-t) \times \mathbf{P}(2,1,3,t_1) + (t-t_1) \times \mathbf{P}(3,2,1,t_2)}{(t_2-t_1)}.$$

What this means geometrically is that if you vary one parameter of a polar value while holding all others constant, the polar value will sweep out a line at a constant velocity, as in Figure 6.3.

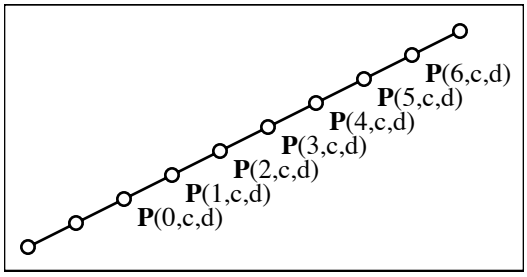


Figure 6.3: Affine map property of polar values.

6.1.1 Subdivision of Bézier Curves

To illustrate how polar values work, we now show how to derive the de Casteljau algorithm using only the first three rules for polar values.

Given a cubic Bézier curve $\mathbf{P}_{[0,1]}(t)$, we wish to split it into $\mathbf{P}_{[0,t]}$ and $\mathbf{P}_{[t,1]}$. The control points of the original curve are labeled

$$\mathbf{P}(0,0,0), \quad \mathbf{P}(0,0,1), \quad \mathbf{P}(0,1,1), \quad \mathbf{P}(1,1,1).$$

The subdivision problem amounts to finding polar values

$$\mathbf{P}(0,0,0), \quad \mathbf{P}(0,0,t), \quad \mathbf{P}(0,t,t), \quad \mathbf{P}(t,t,t),$$

and

$$\mathbf{P}(t,t,t), \quad \mathbf{P}(t,t,1), \quad \mathbf{P}(t,1,1), \quad \mathbf{P}(1,1,1).$$

These new control points can be derived by applying the symmetry and affine map rules for polar values. Refering to Figure 6.4, we can compute

STEP 1.

$$\mathbf{P}(0,0,t) = (1-t) \times \mathbf{P}(0,0,0) + (t-0) \times \mathbf{P}(0,0,1);$$
$$\mathbf{P}(0,1,t) = (1-t) \times \mathbf{P}(0,0,1) + (t-0) \times \mathbf{P}(0,1,1);$$
$$\mathbf{P}(t,1,1) = (1-t) \times \mathbf{P}(0,1,1) + (t-0) \times \mathbf{P}(1,1,1).$$

STEP 2.

$$\mathbf{P}(0,t,t) = (1-t) \times \mathbf{P}(0,0,t) + (t-0) \times \mathbf{P}(0,t,1);$$
$$\mathbf{P}(1,t,t) = (1-t) \times \mathbf{P}(0,t,1) + (t-0) \times \mathbf{P}(t,1,1);$$

STEP 3.

$$\mathbf{P}(t,t,t) = (1-t) \times \mathbf{P}(0,t,t) + (t-0) \times \mathbf{P}(t,t,1);$$

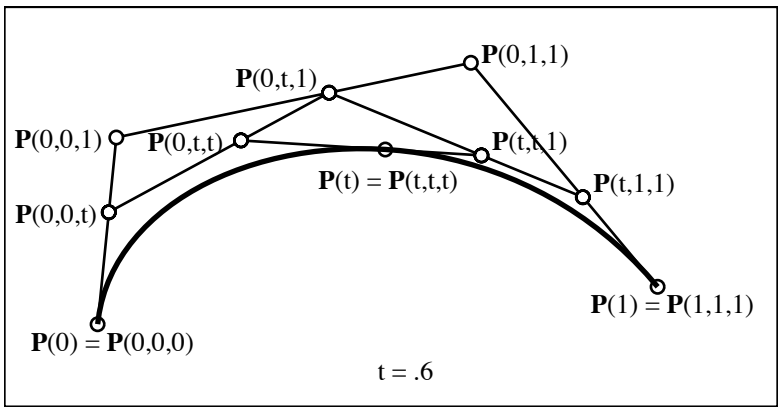


Figure 6.4: Subdividing a cubic Bézier curve.

6.2 Knot Vectors

A knot vector is a list of parameter values, or *knots*, that specify the parameter intervals for the individual Bézier curves that make up a B-spline. For example, if a cubic B-spline is comprised of four Béziercurves with parameter intervals $[1,2]$, $[2,4]$, $[4,5]$, and $[5,8]$, the knot vector would be

$$[t_0, t_1, 1, 2, 4, 5, 8, t_7, t_8].$$

Notice that there are two (one less than the degree) extra knots prepended and appended to the knot vector. These knots control the *end conditions* of the B-spline curve, as discussed in Section 6.6.

For historical reasons, knot vectors are traditionally described as requiring n end-condition knots, and in the real world you will always find a (meaningless) additional knot at the beginning and end of a knot vector. For example, the knot vector in Figure 6.2.b would be $[t_0, 1, 2, 3, 4, 5, 6, 7, 8, t_9]$, where the values of t_0 and t_9 have absolutely no effect on the curve. Therefore, we ignore these dummy knot values in our discussion, but be aware that they appear in B-spline literature and software.

Obviously, a knot vector must be non-decreasing sequence of real numbers. If any knot value is repeated, it is referred to as a *multiple knot*. More on that in Section 6.4. A B-spline curve whose knot vector is evenly spaced is known as a *uniform* B-spline. If the knot vector is not evenly spaced, the curve is called a *non-uniform* B-spline.

6.3 Extracting Bézier Curves from B-splines

We are now ready to discuss the central practical issue for B-splines, namely, how does one find the control points for the Bézier curves that make up a B-spline. This procedure is often called the Böhm algorithm after Professor Wolfgang Böhm [B81].

Consider the B-spline in Figure 6.2.b consisting of Bézier curves over domains $[3, 4]$, $[4, 5]$, and $[5, 6]$. The control points of those three Bézier curves have polar values

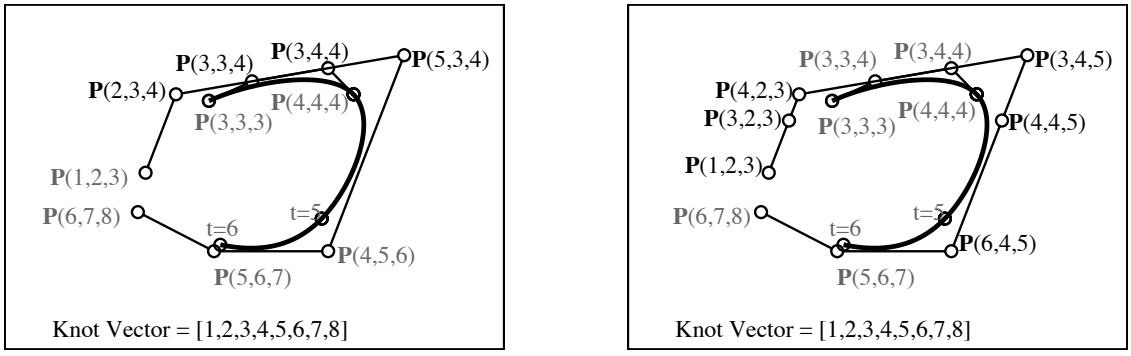
$$\mathbf{P}(3,3,3), \quad \mathbf{P}(3,3,4), \quad \mathbf{P}(3,4,4), \quad \mathbf{P}(4,4,4)$$

$$\mathbf{P}(4,4,4), \quad \mathbf{P}(4,4,5), \quad \mathbf{P}(4,5,5), \quad \mathbf{P}(5,5,5)$$

$$\mathbf{P}(5,5,5), \quad \mathbf{P}(5,5,6), \quad \mathbf{P}(5,6,6), \quad \mathbf{P}(6,6,6)$$

respectivly. Our puzzle is to apply the affine and symmetry properties to find those polar values given the B-spline polar values.

For the Bézier curve over $[3, 4]$, we first find that $\mathbf{P}(3,3,4)$ is $1/3$ of the way from $\mathbf{P}(2,3,4)$ to $\mathbf{P}(5,3,4) = \mathbf{P}(3,4,5)$. Likewise, $\mathbf{P}(3,4,4)$ is $2/3$ of the way from $\mathbf{P}(3,4,2) = \mathbf{P}(2,3,4)$ to $\mathbf{P}(3,4,5)$. See Figure 6.5.a.



(a) First Step.

(b) Second Step.

Figure 6.5: Böhm algorithm.

Before we can locate $\mathbf{P}(3,3,3)$ and $\mathbf{P}(4,4,4)$, we must find the auxilliary points $\mathbf{P}(3,2,3)$ ($2/3$ of the way from $\mathbf{P}(1,2,3)$ to $\mathbf{P}(4,2,3)$) and $\mathbf{P}(4,4,5)$ ($2/3$ of the way from $\mathbf{P}(3,4,5)$ to $\mathbf{P}(6,4,5)$)

as shown in Figure 6.5.b. Finally, $\mathbf{P}(3,3,3)$ is seen to be half way between $\mathbf{P}(3,2,3)$ and $\mathbf{P}(3,3,4)$, and $\mathbf{P}(4,4,4)$ is seen to be half way between $\mathbf{P}(3,4,4)$ and $\mathbf{P}(4,4,5)$.

Note that the four Bézier control points were derived from exactly four B-spline control points; $\mathbf{P}(5,6,7)$ and $\mathbf{P}(6,7,8)$ were not involved. This means that $\mathbf{P}(5,6,7)$ and $\mathbf{P}(6,7,8)$ can be moved without affecting the Bézier curve over $[3,4]$. In general, the Bézier curve over $[t_i, t_{i+1}]$ is only influenced by B-spline control points that have t_i or t_{i+1} as one of the polar value parameters. For this reason, B-splines are said to possess the property of *local control*, since any given control point can influence at most n curve segments.

6.4 Multiple knots

If a knot vector contains two identical non-end-condition knots $t_i = t_{i+1}$, the B-spline can be thought of as containing a zero-length Bézier curve over $[t_i, t_{i+1}]$. Figure 6.9 shows what happens when two knots are moved together. The Bézier curve over the degenerate interval $[5,5]$ has polar values $\mathbf{P}(5,5,5)$, $\mathbf{P}(5,5,5)$, $\mathbf{P}(5,5,5)$, $\mathbf{P}(5,5,5)$, which is merely the single point $\mathbf{P}(5,5,5)$. It can be

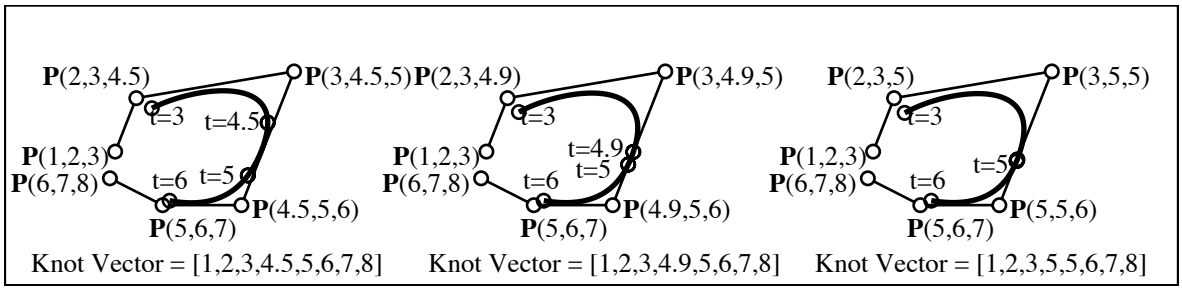


Figure 6.6: Double knot.

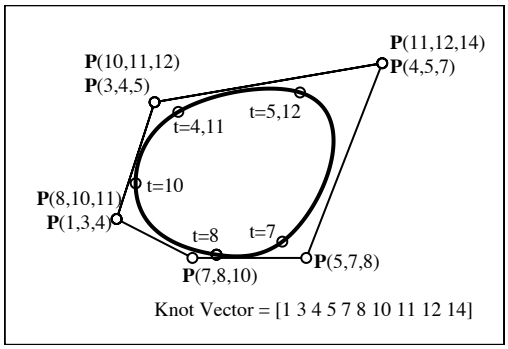
shown that a multiple knot diminishes the continuity between adjacent Bézier curves. The continuity across a knot of multiplicity k is generally $n - k$.

6.5 Periodic B-splines

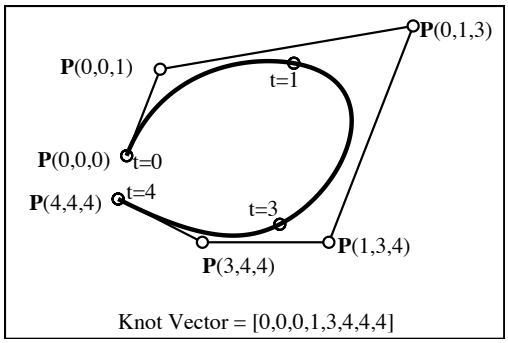
A periodic B-spline is a B-spline which closes on itself. This requires that the first n control points are identical to the last n, and the first n parameter intervals in the knot vector are identical to the last n intervals as in Figure 6.7.a.

6.6 Bézier end conditions

We earlier noted that a knot vector always has $n - 1$ extra knots at the beginning and end which do not signify Bézier parameter limits (except in the periodic case), but which influence the shape of the curve at its ends. In the case of an open (i.e., non-periodic) B-spline, one usually chooses an n-fold knot at each end. This imposes a Bézier behavior on the end of the B-spline, in that the curve interpolates the end control points and is tangent to the control polygon at its endpoints. One can verify this by noting that to convert such a B-spline into Bézier curves, the two control points at each end are already in Bézier form. This is illustrated in Figure 6.7.b.



(a) Periodic B-Spline.



(b) Bézier End Conditions.

Figure 6.7: Special B-Spline Curves.

6.7 Knot insertion

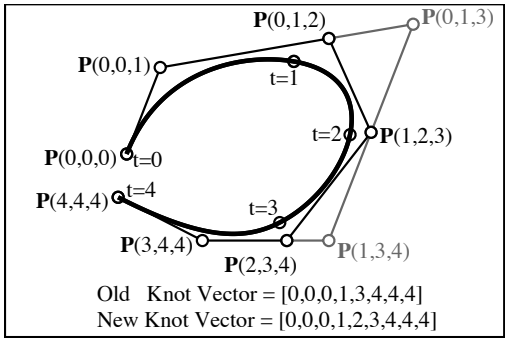
A standard design tool for B-splines is knot insertion. In the knot insertion process, a knot is added to the knot vector of a given B-spline. This results in an additional control point and a modification of a few existing control points. The end result is a curve defined by a larger number of control points, but which defines exactly the same curve as before knot insertion.

Knot insertion has several applications. One is the de Boor algorithm for evaluating a B-spline (discussed in the next section). Another application is to provide a designer with the ability to add local details to a B-spline. Knot insertion provides more local control by isolating a region to be modified from the rest of the curve, which thereby becomes immune from the local modification.

Consider adding a knot at $t = 2$ for the B-spline in Figure 6.7.b. As shown in Figure 6.8.a, this involves replacing $\mathbf{P}(0,1,3)$ and $\mathbf{P}(1,3,4)$ with $\mathbf{P}(0,1,2)$, $\mathbf{P}(1,2,3)$, and $\mathbf{P}(2,3,4)$. Figure 6.8.b shows the new set of control points, which are easily obtained using the affine and symmetry properties of polar values.

	Initial	After Knot Insertion
Knot Vector:	[(0,0,0,1,3,4,4,4)]	[(0,0,0,1,2,3,4,4,4)]
Control Points:	**P**(0,0,0)	**P**(0,0,0)
	P(0,0,1)	**P**(0,0,1)
		P(0,1,2)
	P(0,1,3)	
		P(1,2,3)
	P(1,3,4)	
		P(2,3,4)
	P(3,4,4)	**P**(3,4,4)
	P(4,4,4)	**P**(4,4,4)

(a) Knots and Polar Labels.



(b) Control Points.

Figure 6.8: Knot Insertion.

Note that the continuity at $t = 2$ is C^∞.

Example
This degree four polynomial Bézier curve begins at $t = 1$ and ends at $t = 5$, and is therefore also a B-spline with knot vector [11115555]. Insert a knot at $t = 3$. State the control point coordinates and knot vector after this knot insertion is performed.

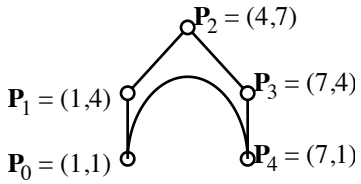


Figure 6.9: B-spline with knot vector [11115555].

Solution
The control points whose polar labels are $f(1,1,1,5) = (1,4)$, $f(1,1,5,5) = (4,7)$ and $f(1,5,5,5) = (7,4)$ will be replaced with polar labels $f(1,1,1,3) = (1,2.5)$, $f(1,1,3,5) = (2.5,5.5)$, $f(1,3,5,5) = (5.5,5.5)$, and $f(3,5,5,5) = (7,2.5)$.

6.8 The de Boor algorithm

The de Boor algorithm provides a method for evaluating a B-spline curve. That is, given a parameter value, find the point on the B-spline corresponding to that parameter value.

Any point on a B-spline $\mathbf{P}(t)$ has a polar value $\mathbf{P}(t, t, \ldots, t)$, and we can find it by inserting knot t n times. This is the de Boor algorithm. Using polar forms, the algorithm is easy to figure out.

The de Boor algorithm is illustrated in Figure 6.10.

6.9 Explicit B-splines

Section 2.14 discusses explicit Bézier curves, or curves for which $x(t) = t$. We can likewise locate B-spline control points in such a way that $x(t) = t$. The x coordinates for an explicit B-spline are known as *Greville abscissae*. For a degree n B-spline with m knots in the knot vector, the Greville abscissae are given by

$$x_i = \frac{1}{n}(t_i + t_{i+1} + \ldots + t_{i+n-1}); \quad i = 0 \ldots m - n. \tag{6.1}$$

6.10 B-spline hodographs

The first derivative (or hodograph) of a B-spline is obtained in a manner similar to that for Bézier curves. The hodograph has the same knot vector as the given B-spline except that the first and last knots are discarded. The control points are given by the equation

$$\mathbf{H}_i = n\frac{(\mathbf{P}_{i+1} - \mathbf{P}_i)}{t_{i+n} - t_i} \tag{6.2}$$

where n is the degree.

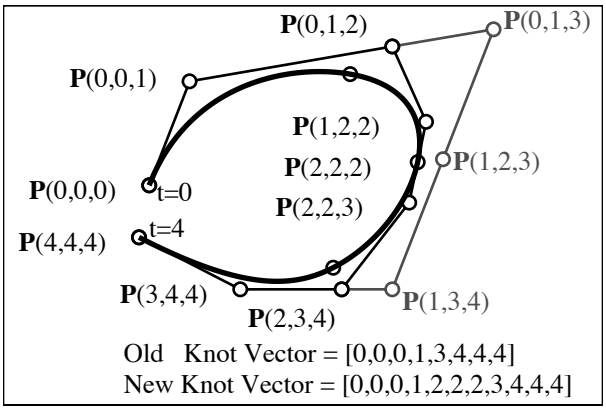


Figure 6.10: De Boor algorithm.

6.11 Symmetric polynomials

We introduced polar form as simply a labeling scheme, with rules defined for creating new control points with different labels. As we have seen, this level of understanding is sufficient to perform many of the basic operations on B-Spline curves. However, there is some beautiful mathematics behind these labels, based on symmetric polynomials.

A symmetric polynomial represents a degree m polynomial in one variable, $p(t)$, as a polynomial in $n \geq m$ variables, $p[t_1, \ldots, t_n]$, that is degree one in each of those variables and such that

$$p[t, \ldots, t] = p(t).$$

$p[t_1, \ldots, t_n]$ is said to be symmetric because the value of the polynomial will not change if the arguments are permuted. For example, if $n = 3$, $p[a, b, c] = p[b, c, a] = p[c, a, b]$ etc.

A symmetric polynomial has the form

$$p[t_1, \ldots, t_n] = \sum_{i=0}^{n} c_i p_i[t_1, \ldots, t_n]$$

where

$$p_0[t_1, \ldots, t_n] = 1; \quad p_i[t_1, \ldots, t_n] = \frac{\sum_{j=1}^{n} t_j p_{i-1}[t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_n]}{n}, \quad i = 1, \ldots n.$$

For example,

$$p[t_1] = c_0 + c_1 t_1,$$

$$p[t_1, t_2] = c_0 + c_1 \frac{t_1 + t_2}{2} + c_2 t_1 t_2,$$

$$p[t_1, t_2, t_3] = c_0 + c_1 \frac{t_1 + t_2 + t_3}{3} + c_2 \frac{t_1 t_2 + t_1 t_3 + t_2 t_3}{3} + c_3 t_1 t_2 t_3,$$

and

$$p[t_1, t_2, t_3, t_4] = c_0 + c_1 \frac{t_1 + t_2 + t_3 + t_4}{4} + c_2 \frac{t_1 t_2 + t_1 t_3 + t_1 t_4 + t_2 t_3 + t_2 t_4 + t_3 t_4}{6}$$
$$+ c_3 \frac{t_1 t_2 t_3 + t_1 t_2 t_4 + t_1 t_3 t_4 + t_2 t_3 t_4}{4} + c_4 t_1 t_2 t_3 t_4.$$

The symmetric polynomial $b[t_1, \ldots t_n]$ for which $p[t, \ldots t] = p(t)$ is referred to as the *polar form* or *blossom* of $p(t)$.

Example Find the polar form of $p(t) = t^3 + 6t^2 + 3t + 1$.
Answer: $p[t_1, t_2, t_3] = 1 + 3\frac{t_1+t_2+t_3}{3} + 6\frac{t_1t_2+t_1t_3+t_2t_3}{3} + t_1t_2t_3$.

Theorem For every degree m polynomial $p(t)$ there exists a unique symmetric polynomial $p[t_1, \ldots, t_n]$ of degree $n \geq m$ such that $p[t, \ldots, t] = p(t)$. Furthermore, the coefficients b_i of the degree n Bernstein polynomial over the interval $[a, b]$ are

$$b_i = p[\underbrace{a, \ldots, a}_{n-i}, \underbrace{b, \ldots, b}_{i}]$$

Example Convert $p(t) = t^3 + 6t^2 + 3t + 1$ to a degree 3 Bernstein polynomial over the interval $[0, 1]$. We use the polar form of $p(t)$: $p[t_1, t_2, t_3] = 1 + 3\frac{t_1+t_2+t_3}{3} + 6\frac{t_1t_2+t_1t_3+t_2t_3}{3} + t_1t_2t_3$. Then,

$$b_0 = p[0, 0, 0] = 1, \quad b_1 = p[0, 0, 1] = 2, b_2 = p[0, 1, 1] = 5, b_3 = p[1, 1, 1] = 11.$$

6.12 Combining Bézier curves into a B-spline

Here we suggest how to convert a string of cubic Bézier curves into a single B-spline. The process initializes by assigning the first four B-spline control points to be the control points of the first Bézier curve, and the knot vector is initially $[0, 0, 0, 0, 1, 1, 1, 1]$.

Thereafter, each subsequent Bézier curve is analyzed to determine what order of continuity exists between it and the current B-spline, and it is appended to the B-spline as follows. Assume that at some step in this process, the B-spline has a knot vector $[k_{i-3}, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, k_{i+1}, k_{i+1}]$ with $k_{i-2} \leq k_{i-1} \leq k_i < k_{i+1}$, and the B-spline control points are labelled

$$\mathbf{P}_1, \ldots, \mathbf{P}_{n-3}, \quad \mathbf{P}_{n-2}, \quad \mathbf{P}_{n-1}, \quad \mathbf{P}_n.$$

The control points of the Béziercurve to be appended are

$$\mathbf{Q}_0 = \mathbf{P}_n, \quad \mathbf{Q}_1, \quad \mathbf{Q}_2, \quad \mathbf{Q}_3.$$

Then, depending on the continuity order between the B-spline and the Bézier curve, the B-spline after appending the Bézier becomes

Continuity	Knot Vector	Control Points
C^0	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_{n-2}, \mathbf{P}_{n-1}, \mathbf{P}_n, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3$
C^1	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_{n-2}, \mathbf{P}_{n-1}, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3$
C^2	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_{n-2}, \mathbf{P}_\alpha, \mathbf{Q}_2, \mathbf{Q}_3$
C^3	$[\ldots, k_{i-2}, k_{i-1}, k_i, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_\beta, \mathbf{P}_\gamma, \mathbf{Q}_3$

C^0 continuity occurs if control points $\mathbf{P}_{n-1}$, $\mathbf{P}_n$, and $\mathbf{Q}_1$ are not collinear. If they are collinear, then the value of knot e is chosen so as to satisfy

$$|[\mathbf{P}_n - \mathbf{P}_{n-1}](k_{i+1} - k_i) - [\mathbf{Q}_1 - \mathbf{P}_n](e - k_{i+1})| < TOL$$

This provides for C^1 (not merely G^1) continuity. TOL is a small number which is needed to account for floating point error. An appropriate value for TOL is the width of the reverse map of a pixel into world space.

C^2 continuity occurs if, in addition to C^1 continuity, the relationship

$$|(\mathbf{P}_{n-2}-\mathbf{Q}_2)(k_{i+1}-k_{i-1})(k_{i+1}-e)+(\mathbf{P}_{n-1}-\mathbf{P}_{n-2})(e-k_{i-1})(k_{i+1}-e)+(\mathbf{Q}_2-\mathbf{Q}_1)(k_i-e)(k_{i+1}-k_{i-1})|<TOL.$$

is satisfied. We can then compute

$$\mathbf{P}_\alpha=\frac{(k_{i+1}-e)\mathbf{P}_{n-2}+(e-k_{i-1})\mathbf{P}_{n-1}}{k_{i+1}-k_{i-1}}=\frac{(k_{i+1}-k_i)\mathbf{Q}_2+(e-k_{i+1})\mathbf{Q}_1}{e-k_i}.$$

C^3 continuity occurs if, further, the relationship

$$\left|\mathbf{P}_\alpha-\frac{(e-k_{i+1})\mathbf{P}_\beta+(k_{i+1}-k_{i-1})\mathbf{P}_\gamma}{e-k_{i-1}}\right|<TOL$$

is satisfied, where

$$\mathbf{P}_\beta=\frac{(e-k_{i-2})\mathbf{P}_{n-2}+(k_{i-1}-e)\mathbf{P}_{n-3}}{k_{i-1}-k_{i-2}}$$

and

$$\mathbf{P}_\gamma=\frac{(k_{i+1}-k_i)\mathbf{Q}_3+(k_i-e)\mathbf{Q}_2}{k_{i+1}-e}$$

6.13 Knot Intervals

B-spline curves are typically specified in terms of a set of control points, a knot vector, and a degree. Knot information can also be imposed on a B-spline curve using knot intervals, introduced in [SZSS98] as a way to assign knot information to subdivision surfaces. A knot interval is the difference between two adjacent knots in a knot vector, i.e., the parameter length of a B-spline curve segment. For even-degree B-spline curves, a knot interval is assigned to each control point, since each control point in an even-degree B-spline corresponds to a curve segment. For odd-degree B-spline curves, a knot interval is assigned to each control polygon edge, since in this case, each edge of the control polygon maps to a curve segment.

While knot intervals are basically just an alternative notation for representing knot vectors, knot intervals offer some nice advantages. For example, knot interval notation is more closely coupled to the control polygon than is knot vector notation. Thus, knot intervals have more geometric meaning than knot vectors, since the effect of altering a knot interval can be more easily predicted. Knot intervals are particularly well suited for periodic B-splines.

Knot intervals contain all of the information that a knot vector contains, with the exception of a knot origin. This is not a problem, since the appearance of a B-spline curve is invariant under linear transformation of the knot vector—that is, if you add any constant to each knot the curve's appearance does not change. B-splines originated in the field of approximation theory and were initially used to approximate functions. In that context, parameter values are important, and hence, knot values are significant. However, in curve and surface shape design, we are almost never concerned about absolute parameter values.

For odd-degree B-spline curves, the knot interval d_i is assigned to the control polygon edge $\mathbf{P}_i$—$\mathbf{P}_{i+1}$. For even-degree B-spline curves, knot interval d_i is assigned to control point $\mathbf{P}_i$. Each vertex (for even degree) or edge (for odd degree) has exactly one knot interval. If the B-spline is not periodic, $\frac{n-1}{2}$ "end-condition" knot intervals must be assigned past each of the two end control points. They can simply be written adjacent to "phantom" edges or vertices sketched adjacent to the end control points; the geometric positions of those phantom edges or vertices are immaterial.

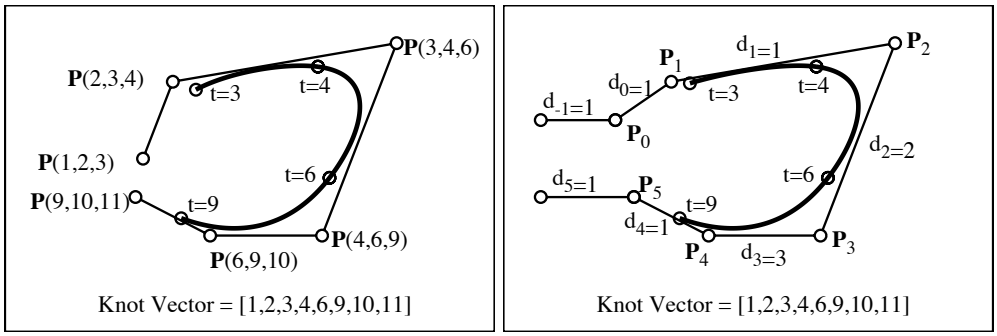


Figure 6.11: Sample cubic B-spline

Figure 6.11 shows a cubic B-spline curve. The control points in Figure 6.11.a are labeled with polar values, and Figure 6.11.b shows the control polygon edges labeled with knot intervals. End-condition knots require that we hang one knot interval off each end of the control polygon. Note the relationship between the knot vector and the knot intervals: Each knot interval is the difference between two consecutive knots in the knot vector.

For periodic B-splines, things are even simpler, since we don't need to deal with end conditions. Figure 6.12 shows two cubic periodic B-splines labelled with knot intervals. In this example, note

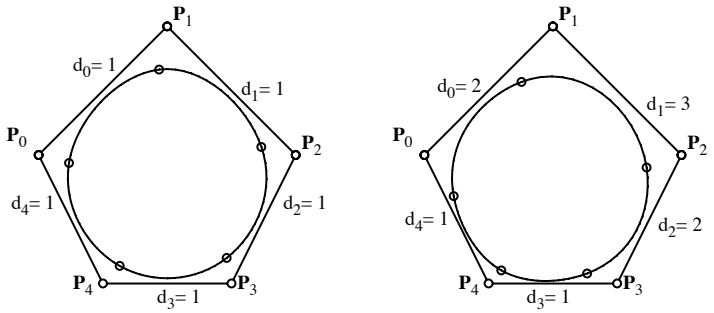


Figure 6.12: Periodic B-splines labelled with knot intervals

that as knot interval d_1 changes from 1 to 3, the length of the corresponding curve segment increases.

Figure 6.13 shows two periodic B-splines with a double knot (imposed by setting $d_0 = 0$) and a triple knot (set $d_0 = d_1 = 0$).

In order to determine formulae for operations such as knot insertion in terms of knot intervals, it is helpful to infer polar labels for the control points. Polar algebra [Ram89b] can then be used to create the desired formula. The arguments of the polar labels are sums of knot intervals. We are free to choose any knot origin. For the example in Figure 6.14, we choose the knot origin to coincide with control points $\mathbf{P}_0$. Then the polar values are as shown in Figure 6.14.b.

The following subsections show how to perform knot insertion and interval halving, and how to compute hodographs using knot intervals. These formulae can be verified using polar labels. The expressions for these operations written in terms of knot vectors can be found, for example, in [HL93].

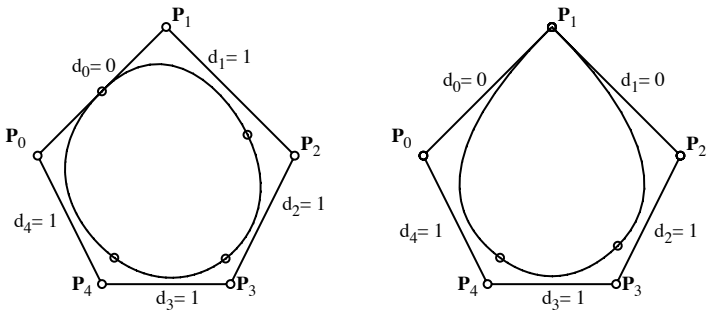


Figure 6.13: Periodic B-splines with double and triple knots.

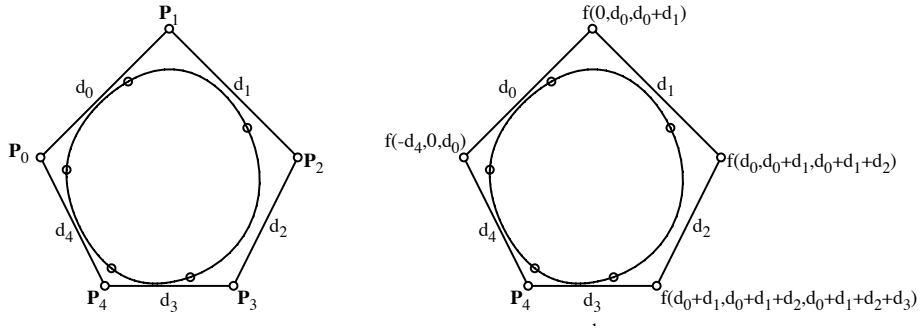


Figure 6.14: Inferring polar labels from knot intervals.

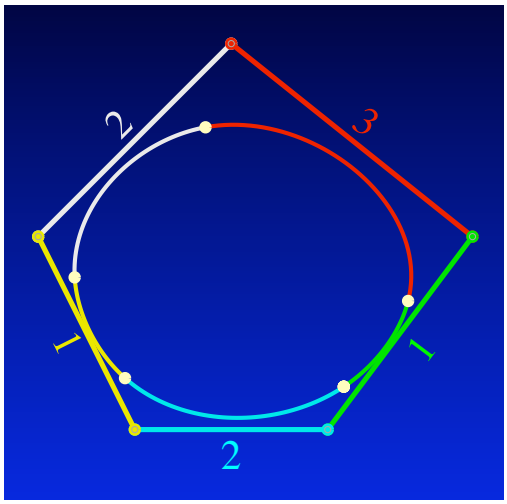
6.13.1 Knot Insertion

Recall from Section 6.7 that knot insertion is a fundamental operation whereby control points are added to a B-Spline control polygon without changing the curve. In Section 6.7, knot insertion is discussed in terms of knot vectors and polar form. We can perform knot insertion using knot intervals, as well. In fact, the construction algorithm is perhaps even easier to remember and to carry out than the polar form construction.

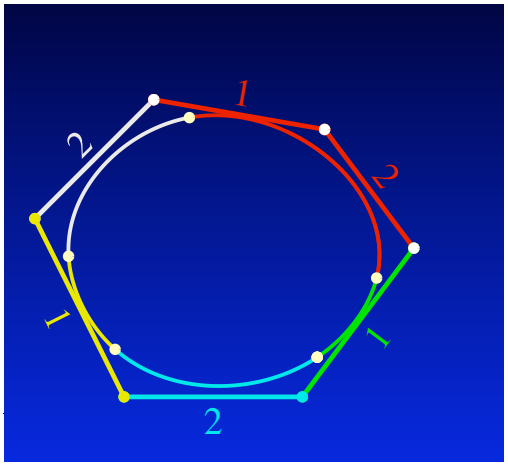
Using knot intervals, it is more natural to speak of interval splitting than of knot insertion. For example, Figure 6.15 shows a cubic periodic B-Spline in which the red knot interval is split into two knot intervals that sum to the original knot interval. The only question we need to address is, how can we compute the Cartesian coordinates of the white control points in Figure 6.15.b.

Figure 6.16 illustrates how to compute those control points. Split the white, red, and green edges of the control polygon into three segments each whose lengths are proportional to the neighboring knot intervals, as shown in Figure 6.16.a. Each of those three edges contains a segment labeled with a red "3." Split each of those segments into two pieces whose lengths are proportional to 1:2, as shown in Figure 6.16.b. This produces the three desired control points. Figure 6.17 shows the procedure for splitting the interval "3" into two equal intervals.

An important special case involves inserting a zero knot interval. In knot vector jargon, this is the same as inserting a double knot. In knot interval jargon, we can say that we are splitting a knot interval into two intervals, one of which is zero, and the other of which is the original knot interval. This special case is illustrated in Figure 6.18.a and Figure 6.18.b. If we repeat this operation three

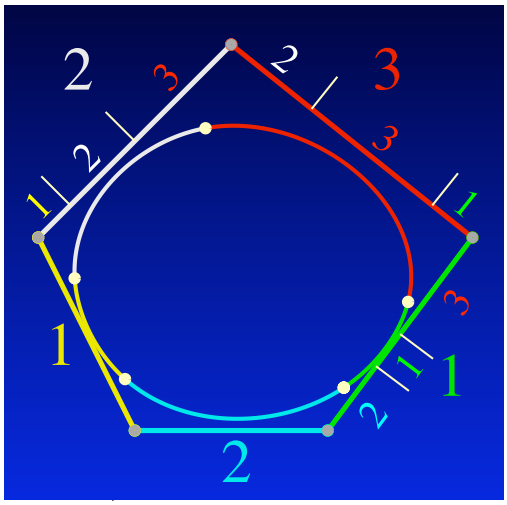


(a) Periodic Cubic B-Spline Labeled with Knot Intervals.

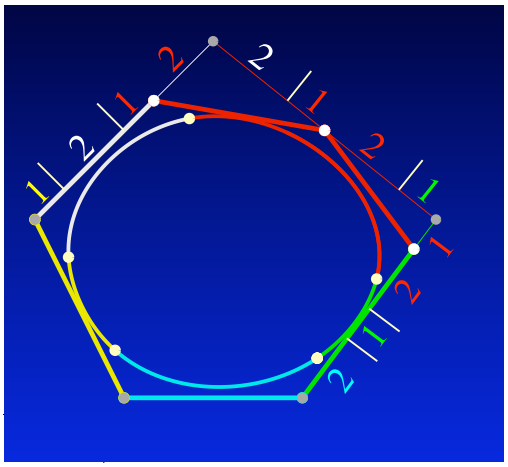


(b) Splitting Interval 3 into Intervals 1 and 2.

Figure 6.15: Knot Insertion using Knot Intervals.

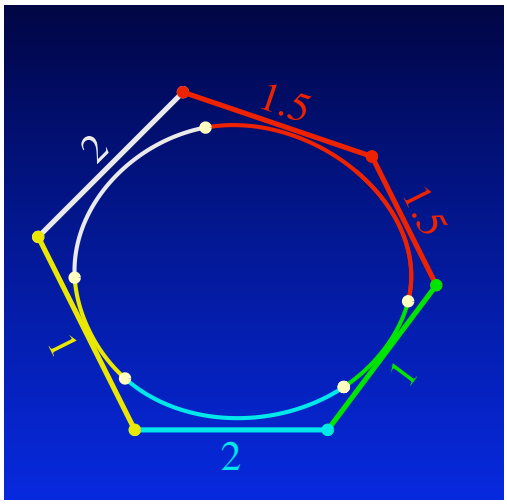


(a) Dividing Edges of the Control Polygon into Three Segments Whose Lengths are Proportional to the Neighboring Knot Intervals.

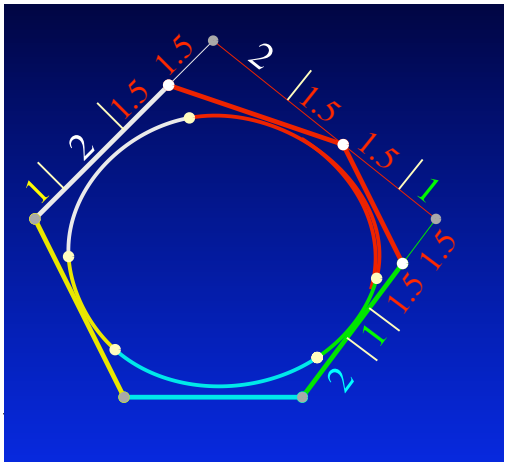


(b) Splitting Intervals Labeled with a 3 into Intervals 1 and 2.

Figure 6.16: "Interval Splitting" using Knot Intervals.



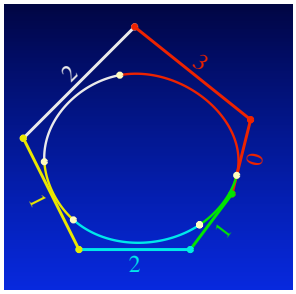
(a) Splitting the Interval Labeled with a 3 into Intervals 1.5 and 1.5.



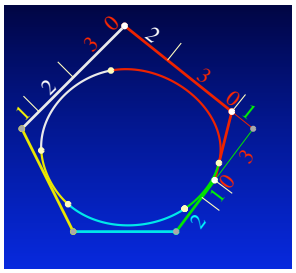
(b) Construction.

Figure 6.17: "Interval Splitting" using Knot Intervals.

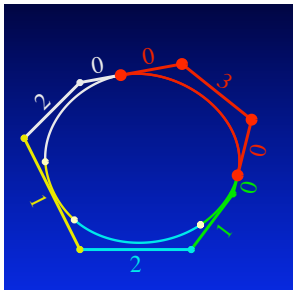
more times, arriving at the knot interval configuration shown in Figure 6.18.c, we uncover the Bézier control points for the red Bézier curve. The reason for this will be clear if you recall that a Bézier curve is a special case of B-Spline curve with knot vector $[a, a, a, b, b, b]$. Converting from knot vector to knot interval notation, we see that the two pairs of adjacent zero knot interval in Figure 6.18.c represent two sets of triple knots in the knot vector.



(a) Splitting the Interval Labeled with a 3 into Intervals 3 and 0.



(b) Construction.



(c) Bézier control points.

Figure 6.18: Introducing Zero Knot Intervals.

6.13.2 Interval Halving

Subdivision surfaces such as the Catmull-Clark scheme are based on the notion of inserting a knot half way between each existing pair of knots in a knot vector. These methods are typically restricted to uniform knot vectors. Knot intervals help to generalize this technique to non-uniform B-splines. Using knot intervals, we can think of this process as cutting in half each knot interval. For a quadratic

non-uniform B-spline, the interval halving procedure is a generalization of Chaikin's algorithm, but the placement of the new control points becomes a function of the knot interval values. If each knot interval is cut in halve, the resulting control polygon has twice as many control points, and their coordinates $\mathbf{Q}_k$ are:

$$\begin{aligned} \mathbf{Q}_{2i} &= \frac{(d_i + 2d_{i+1})\mathbf{P}_i + d_i\mathbf{P}_{i+1}}{2(d_i + d_{i+1})} \\ \mathbf{Q}_{2i+1} &= \frac{d_{i+1}\mathbf{P}_i + (2d_i + d_{i+1})\mathbf{P}_{i+1}}{2(d_i + d_{i+1})} \end{aligned} \tag{6.3}$$

as illustrated in Figure 6.19.

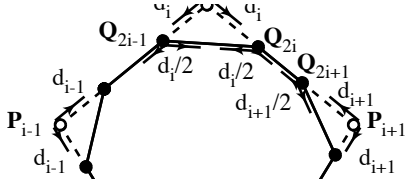


Figure 6.19: Interval halving for a non-uniform quadratic B-spline curve.

For non-uniform cubic periodic B-spline curves, interval halving produces a new control point corresponding to each edge, and a new control point corresponding to each original control point. The equations for the new control points $\mathbf{Q}_k$ generated by interval halving are:

$$\mathbf{Q}_{2i+1} = \frac{(d_i + 2d_{i+1})\mathbf{P}_i + (d_i + 2d_{i-1})\mathbf{P}_{i+1}}{2(d_{i-1} + d_i + d_{i+1})} \tag{6.4}$$

$$\mathbf{Q}_{2i} = \frac{d_i\mathbf{Q}_{2i-1} + (d_{i-1} + d_i)\mathbf{P}_i + d_{i-1}\mathbf{Q}_{2i+1}}{2(d_{i-1} + d_i)} \tag{6.5}$$

as shown in Figure 6.20.

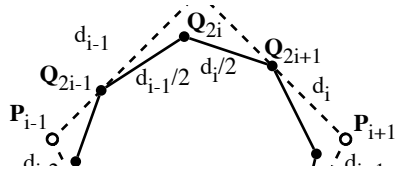


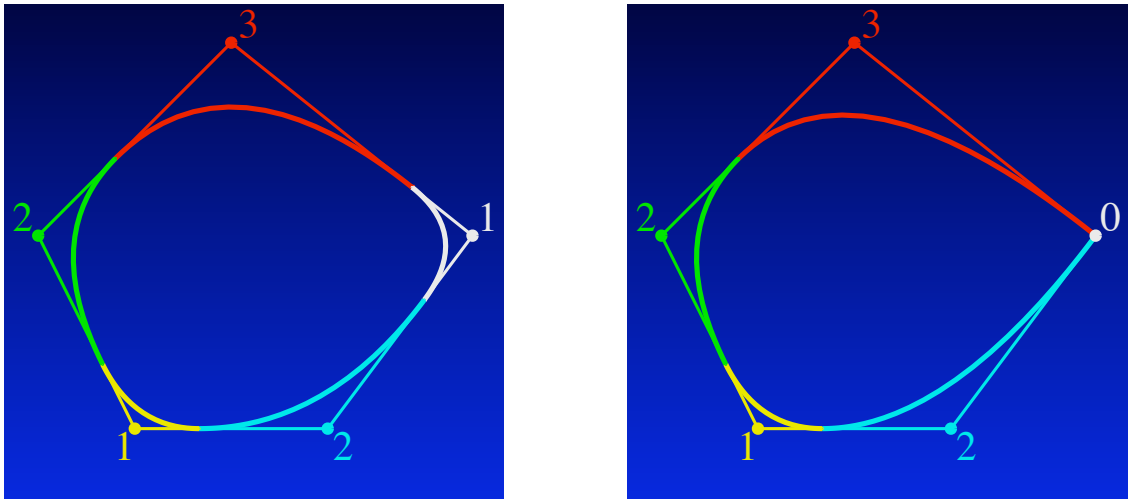
Figure 6.20: Interval halving for a non-uniform cubic B-spline curve.

Note that each new knot interval is half as large as its parent.

6.13.3 Degree-Two B-Splines using Knot Intervals

For any odd-degree B-Spline curve, the knot intervals are associated with the edges of the control polygon. For even-degree, the knot intervals are associated with the control points. Figure 6.21 shows two quadratic B-Splines with different knot intervals.

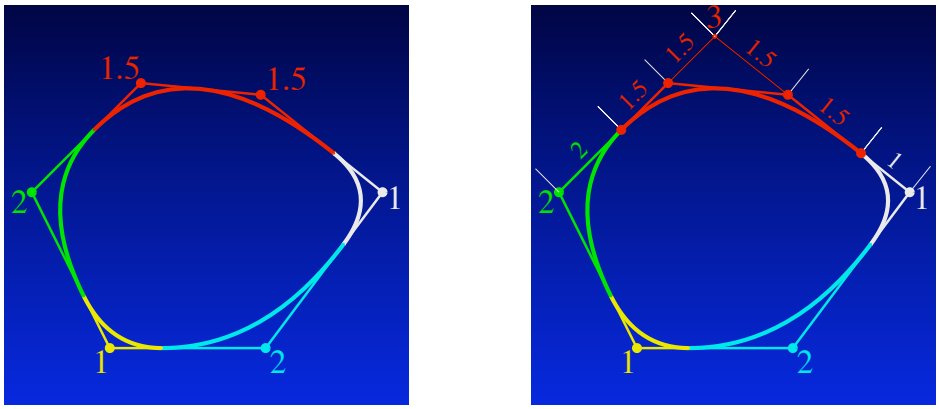
Figure 6.22 illustrates how to perform interval splitting for a quadratic B-Spline. Figure 6.23 illustrates how to split off a zero-knot interval on a quadratic B-Spline. To find the Bézier control points in a quadratic B-Spline, it suffices to insert a single zero knot interval on each side of a non-zero knot interval.



(a) Example 1.

(b) Example with Zero Knot Interval.

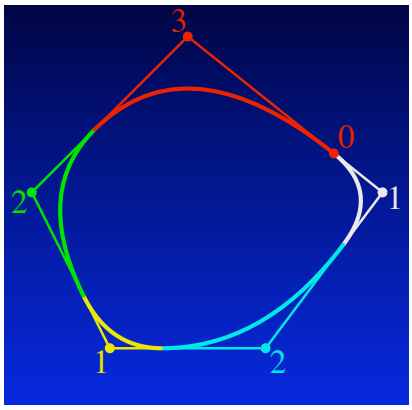
Figure 6.21: Quadratic B-Spline Curves.



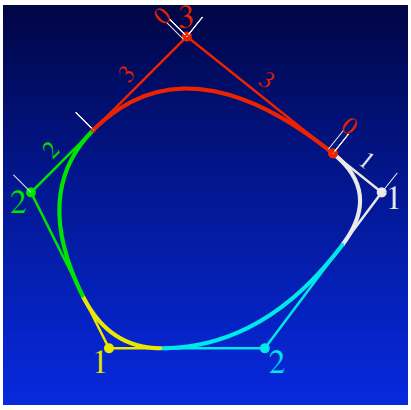
(a) Splitting the Knot Interval 3 into two equal knot intervals.

(b) Construction for the Interval Splitting.

Figure 6.22: Interval Splitting of a Quadratic B-Spline Curve.



(a) Splitting the Knot Interval 3 into two equal knot intervals.

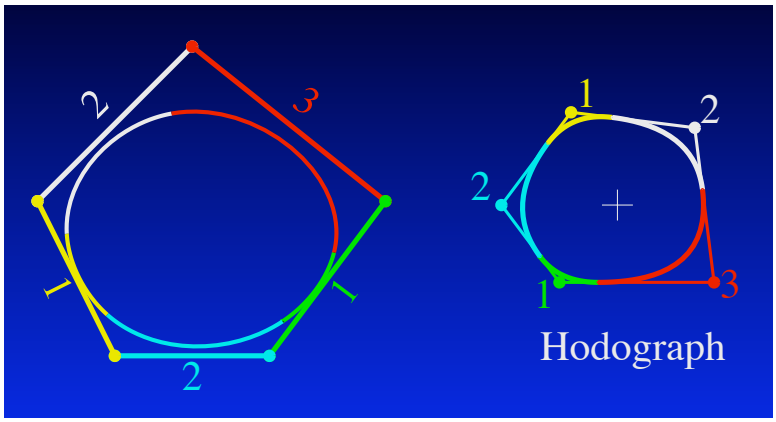


(b) Construction for the Interval Splitting.

Figure 6.23: Interval Splitting of a Quadratic B-Spline Curve.

6.13.4 Hodographs

Section 2.7 discusses the hodograph of a polynomial Bézier curve. The hodograph of a polynomial B-Spline curve can be constructed in a similar manner.



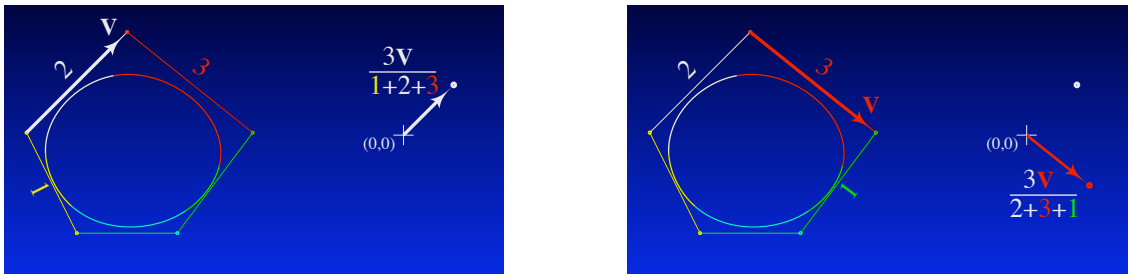
.

Figure 6.24: Hodograph of a Degree 3 Polynomial B-Spline Curve.

Figure 6.24 shows a cubic B-Spline curve and its hodograph, represented as a degree-two B-Spline curve. Note the relationship between the knot vectors in the original curve and in its hodograph. The control points for the hodograph are found in a manner similar to the control points for a Bézier hodograph.

The exact formula for the control points, as illustrated in Figure 6.25, is as follows. The hodograph of a degree n B-spline $\mathbf{P}(t)$ with knot intervals d_i and control points $\mathbf{P}_i$ is a B-spline of degree $n-1$ with the same knot intervals d_i and with control points $\mathbf{Q}_i$ where

$$\mathbf{Q}_i = c_i(\mathbf{P}_{i+1} - \mathbf{P}_i). \tag{6.6}$$



(a) (b)

Figure 6.25: Finding the Control Points of a B-Spline Hodograph.

The scale factor c_i is the inverse of the average value of n neighboring knot intervals. Specifically, if the curve is even-degree $n = 2m$, then

$$c_i = \frac{n}{d_{i-m+1} + \ldots + d_{i+m}}$$

and if the curve is odd degree $n = 2m + 1$

$$c_i = \frac{n}{d_{i-m} + \ldots + d_{i+m}}$$

6.13.5 Degree elevation

Ramshaw [Ram89b] presented an elegant insight into degree elevation using polar form. The symmetry property of polar labels demands that

$$f(a,b) = \frac{f(a) + f(b)}{2}; \quad f(a,b,c) = \frac{f(a,b) + f(a,c) + f(b,c)}{3}; \tag{6.7}$$

$$f(a,b,c,d) = \frac{f(a,b,c) + f(a,b,d) + f(a,c,d) + f(b,c,d)}{4}; \quad \text{etc.} \tag{6.8}$$

The procedure of degree elevation on a periodic B-spline that is labeled using knot intervals results in two effects. First, an additional control point is introduced for each curve segment. Second, if the sequence of knot intervals is initially $d_1, d_2, d_3, \ldots$, the sequence of knot intervals on the degree elevated control polygon will be $d_1, 0, d_2, 0, d_3, 0, \ldots$. The zeroes must be added because degree elevation raises the degree of each curve segment without raising the continuity between curve segments,

Degree elevation of a degree one B-spline is simple: merely insert a new control point on the midpoint of each edge of the control polygon. The knot intervals are as shown in Figures 6.26.a and b.

Degree elevation for a degree two B-spline is illustrated in Figures 6.26.c and d. The new control points are:

$$\mathbf{P}_{i,j} = \frac{(2d_i + 3d_j)\mathbf{P}_i + d_i\mathbf{P}_j}{3d_i + 3d_j}. \tag{6.9}$$

Figure 6.27 illustrates degree elevation from degree three to four. The equations for the new control points are:

$$\mathbf{P}_{i,i+1} = \frac{(d_i + 2d_{i+1})\mathbf{P}_i + (2d_{i-1} + d_i)\mathbf{P}_{i+1}}{2(d_{i-1} + d_i + d_{i+1})}$$

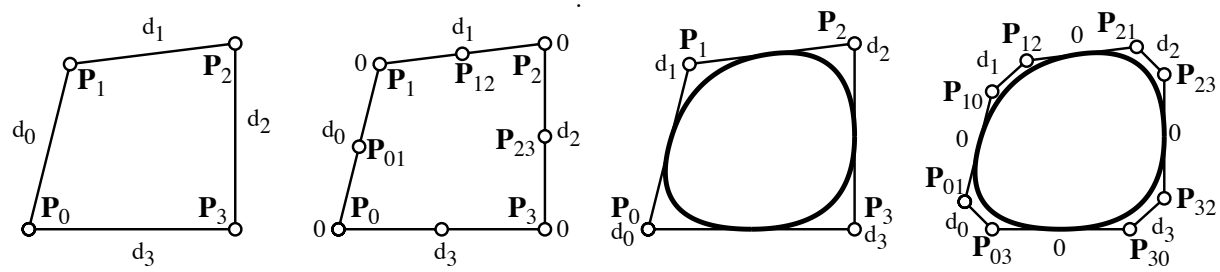


Figure 6.26: Degree elevating a degree one and degree two B-spline.

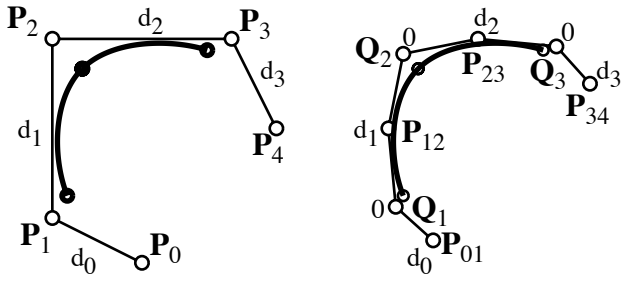


Figure 6.27: Degree elevating a degree three B-spline.

$$\mathbf{Q}_i = \frac{d_i}{4(d_{i-2}+d_{i-1}+d_i)}\mathbf{P}_{i-1} + \left(\frac{d_{i-2}+d_{i-1}}{4(d_{i-2}+d_{i-1}+d_i)} + \frac{d_i+d_{i+1}}{4(d_{i-1}+d_i+d_{i+1})} + \frac{1}{2}\right)\mathbf{P}_i + \frac{d_{i-1}}{4(d_{i-1}+d_i+d_{i+1})}\mathbf{P}_{i+1}$$

6.14 B-spline Basis Functions

A degree d B-spline curve with n control points can be expressed

$$\mathbf{P}(t) = \sum_{i=1}^{n} \mathbf{P_i} B_i^d(t).$$

The knot vector for this curve contains $n+d-1$ knots, denoted $[t_{1-d/2}, \ldots, t_{n+(d-1)/2}]$.

Each B-spline basis function $B_i^d(t)$ can be expressed as an explicit B-spline curve whose x coordinates are the Greville abscissae (Section 6.9) and whose y coordinates are all zero except $y_i = 1$. Therefore, $B_i^d(t) = 0$ for $t \le t_{i-(d+1)/2}$ and $t \ge t_{i+(d+1)/2}$. Each basis function is a piecewise polynomial that has $d+1$ non-zero segments. For example, the cubic B-spline in Figure 6.28, $n = 7$ and the knot vector is $[0,0,0,2,3,4,5,5,5]$. The basis function $B_3^3(t)$ is shown in Figure **??**.

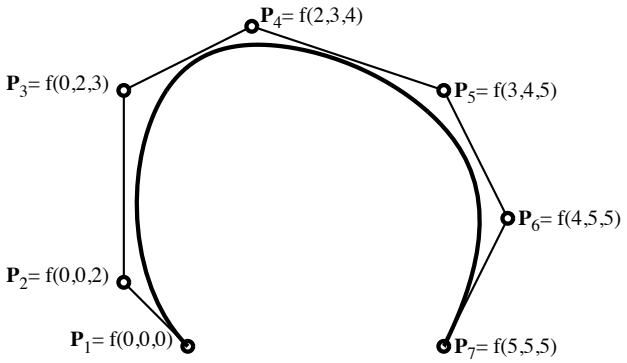


Figure 6.28: Cubic B-Spline Curve.

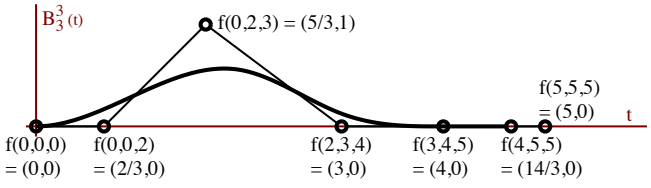


Figure 6.29: Basis function $B_3^3(t)$.

The basis function in Figure 6.29 is just like any other B-Spline curve. To evaluate $B_3^3(t)$, we compute polar value $f(t,t,t)$ using knot insertion. The y coordinate of $f(t,t,t)$ is $B_3^3(t)$, and the x-coordinate of $f(t,t,t)$ is t.

6.14.1 B-Spline Basis-Functions using Knot Intervals

Basis functions for B-Splines defined using knot intervals can likewise be represented as explicit B-Spline curves (Section 6.9). For the curve in Figure 6.30, the basis function for $\mathbf{P}_i$ is illustrated in Figure 6.31.

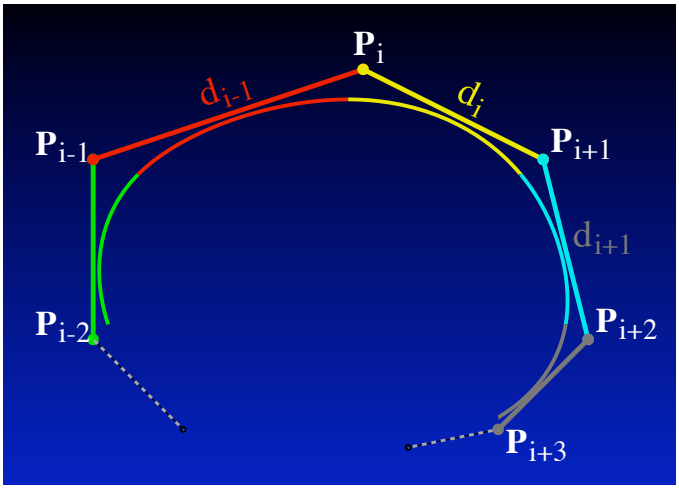


Figure 6.30: Sample Cubic B-Spline Curve.

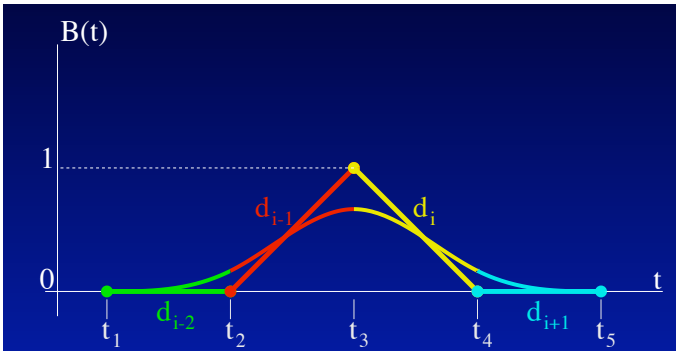


Figure 6.31: B-Spline Basis Function for Control Point $\mathbf{P}_i$ in Figure 6.30.

$B(t)$ is the y-coordinates the curve in Figure 6.31. Those y-coordinates are easily evaluated in the same manner as you would evaluate a point on any B-Spline curve. Note that the y-coordinates of the control points of this explicit B-Spline curve are all zero, except for the control point whose basis function we desire.

If we now evaluate a point on this explicit B-Spline curve using the knot-interval version of the de Boor algorithm, we arrive at the following equations for the B-Spline basis function in Figure 6.31. For $t \in [t_1, t_2]$:

$$B(t) = \frac{(t - t_1)^3}{(t_4 - t_1)(t_3 - t_1)_(t_2 - t_1)} \tag{6.10}$$

For $t \in [t_2, t_3]$:

$$B(t) = \frac{(t-t_2)^2(t_5-t)}{(t_5-t_2)(t_4-t_2)(t_3-t_2)} + \frac{(t-t_2)(t-t_1)(t_4-t)}{(t_4-t_1)(t_4-t_2)(t_3-t_2)} + \frac{(t_3-t)(t-t_1)^2}{(t_4-t_1)(t_3-t_1)(t_3-t_2)} \quad (6.11)$$

For $t \in [t_3, t_4]$:

$$B(t) = \frac{(t_4-t)^2(t-t_1)}{(t_4-t_1)(t_4-t_2)(t_4-t_3)} + \frac{(t_4-t)(t_5-t_)(t-t_2)}{(t_5-t_2)(t_4-t_2)(t_4-t_3)} + \frac{(t-t_3)(t_5-t)^2}{(t_5-t_2)(t_5-t_3)(t_4-t_3)} \quad (6.12)$$

For $t \in [t_4, t_5]$:

$$B(t) = \frac{(t_5-t)^3}{(t_5-t_2)(t_5-t_3)_(t_5-t_4)} \quad (6.13)$$

These four polynomials are C^2, with

$$B'(t_1) = B''(t_1) = 0, \quad B'(t_2) = \frac{3(t_2-t_1)}{(t_3-t_1)(t_4-t_1)}, \quad B''(t_2) = \frac{6}{(t_3-t_1)(t_4-t_1)},$$

$$B'(t_3) = \frac{3(t_1(t_2-t_3) - t_2t_3 + t_3t_4 + t_3t_5 - t_4t_5)}{(t_1-t_4)(t_2-t_4)(t_2-t_5)}, \quad B''(t_3) = \frac{6(t_1+t_2-t_4-t_5)}{(t_1-t_4)(t_4-t_2)(t_2-t_5)}$$

$$B'(t_4) = -\frac{3(t_5-t_4)}{(t_5-t_2)(t_5-t_3)}, \quad B''(t_4) = \frac{6}{(t_5-t_2)(t_5-t_3)}$$

6.14.2 Refinement of B-Spline Basis Functions

Denote by

$$N[s_0, s_1, \ldots, s_{n+1}](s)$$

a B-Spline basis function of degree n over a knot vector $\{s_0, s_1, \ldots, s_{n+1}\}$ and with support $[s_0, s_{n+1}]$. Upon inserting a knot k for which $s_i \le k \le s_{i+i}$, the basis function is split into two scaled basis functions:

$$N[s_0, s_1, \ldots, s_{n+1}](s) = c_i N[s_0, \ldots, s_i, k, s_{i+1}, \ldots, s_n](s) + d_i N[s_1, \ldots, s_i, k, s_{i+1}, \ldots, s_{n+1}](s)$$

where

$$c_i = \begin{cases} \frac{k-s_0}{s_{n+1}-s_0} & k < s_n \\ 1 & k \ge s_n \end{cases}$$

$$d_i = \begin{cases} \frac{s_{n+1}-k}{s_{n+1}-s_1} & k > s_1 \\ 1 & k \le s_1 \end{cases}$$

6.14.3 Recurrence Relation

B-spline basis functions can also be defined using the following recurrence relations.

A degree zero B-spline curve is defined over the interval $[t_i, t_{i+1}]$ using one control point, $\mathbf{P}_0$. Its basis function, which we will denote $B_i^0(t)$ is the step function

$$b_i^0(t) = \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise.} \end{cases}$$

The curve $B_i^0(t)\mathbf{P}_0$ consists of the discrete point $\mathbf{P}_0$.

Blending functions for higher degree B-splines are defined using the recurrence relation:

$$B_i^k(t) = \omega_i^k(t)B_i^{k-1}(t) + (1 - \omega_{i+1}^k(t))B_{i+1}^{k-1}(t) \tag{6.14}$$

where

$$\omega_i^k(t) = \begin{cases} \frac{t-t_i}{t_{i+k-1}-t_i} & \text{if } t_i \neq t_{i+k-1} \\ 0 & \text{otherwise.} \end{cases}$$

A degree one B-spline curve is defined over the interval $[t_i, t_{i+1}]$ using two control points, which we will denote as polar values $\mathbf{P}(t_i)$ and $\mathbf{P}(t_{i+1})$. The curve is simply the line segment joining the two control points:

$$\mathbf{P}(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i}\mathbf{P}(t_i) + \frac{t - t_i}{t_{i+1} - t_i}\mathbf{P}(t_{i+1}).$$

A single degree n B-spline curve segment defined over the interval $[t_i, t_{i+1}]$ with knot vector $\{\ldots, t_{i-1}, t_i, t_{i+1}, t_{i+2}, \ldots\}$ has $n+1$ control points written as polar values

$$\mathbf{P}(t_{i+1-n}, \ldots, t_i), \ldots, \mathbf{P}(t_{i+1}, \ldots, t_{i+n})$$

and blending functions $B_i^n(t)$ which are obtained from equation 6.14. The equation for the curve is:

$$\mathbf{P}(t) = \sum_{j=i}^{n+i} B_{j+1-n}^n(t)\mathbf{P}(t_{j+1-n}, \ldots, t_j) \tag{6.15}$$

In approximation theory, the basis functions themselves are referred to as B-splines.

Chapter 7

Planar Curve Intersection

Curve intersection involves finding the points at which two planar curves intersect. If the two curves are parametric, the solution also identifies the parameter values of the intersection points.

7.1 Bezout's Theorem

Bezout's theorem states that a planar algebraic curve of degree n and a second planar algebraic curve of degree m intersect in exactly mn points, if we properly count complex intersections, intersections at infinity, and possible multiple intersections. If they intersect in more than that number of points, they intersect in infinitely many points. This happens if the two curves are identical, for example.

The correct accounting of the multiplicity of intersections is essential to properly observe Bezout's theorem. If two curves intersect and they are not tangent at the point of intersection (and they do not self-intersect at that point), the intersection is said to have a multiplicity of one. Such an intersection is also called a transversal intersection, or a simple intersection. When two curves are tangent, they intersect twice at the point of tangency. They are also said to intersect with a multiplicity of two. If, in addition, the two curves have the same curvature, they are said to intersect with a multiplicity three. If two curves intersect with a multiplicity of n and there are no self-intersections at that point, the two curves are said to be G^{n-1} continuous at that point.

Bezout's theorem also tells us that two surfaces of degree m and n respectively intersect in an algebraic space curve of degree mn. Also, a space curve of degree m intersects a surface of degree n in mn points. In fact, this provides us with a useful definition for the degree of a surface or space curve. The degree of a surface is the number of times it is intersected by a general line, and the degree of a space curve is the number of times it intersects a general plane. The phrase "number of intersections" must be modified by the phrase "properly counted", which means that complex, infinite, and multiple intersections require special consideration.

7.1.1 Homogeneous coordinates

Proper counting of intersections at infinity is facilitated through the use of homogeneous coordinates. A planar algebraic curve of degree n

$$f(x,y) = \sum_{i+j \leq n} a_{ij} x^i y^j \;=\; 0$$

can be expressed in homogeneous form by introducing a homogenizing variable w as follows:

$$f(X,Y,W) = \sum_{i+j+k=n} a_{ij} X^i Y^j W^k \ = \ 0.$$

Note that, in the homogeneous equation, the degree of every term is n.

Any homogeneous triple (X,Y,W) corresponds to a point whose Cartesian coordinates (x,y) are $(\frac{X}{W}, \frac{Y}{W})$.

7.1.2 Circular Points at Infinity

A circle whose equation is $(x - x_c)^2 \ + \ (y \ - \ y_c)^2 \ - \ r^2 \ = \ 0$ has the homogeneous representation

$$(X - x_c W)^2 \ + \ (Y \ - \ y_c W)^2 \ - \ r^2 W^2 \ = \ 0. \tag{7.1}$$

A point whose homogeneous coordinate $W = 0$ corresponds to a point whose Cartesian coordinates are at infinity.

An example of the practical use of homogeneous coordinates is that it allows us to observe the interesting fact that every circle passes through the two points $(1, i, 0)$ and $(1, -i, 0)$. You can easily verify this by substituting these points into (7.1). The points $(1, i, 0)$ and $(1, -i, 0)$, known as the circular points at infinity, are both infinite and complex.

Circles are degree two curves, and thus Bezout s theorem requires two distinct circles to intersect in exactly four points. However, our practical experience with circles suggests that they intersect in at most two points. Of course, two distinct circles *can* only intersect in at most two **real** points. However, since all circles pass through the two circular points at infinity, we thus account for the two missing intersection points.

7.1.3 Homogeneous parameters

A unit circle at the origin can be expressed parametrically as

$$x \ = \ \frac{-t^2 \ + \ 1}{t^2 \ + \ 1} \qquad y \ = \ \frac{2t}{t^2 \ + \ 1}. \tag{7.2}$$

A circle can also be expressed in terms of homogeneous parameters (T,U) where $t \ = \ \frac{T}{U}$:

$$x \ = \ \frac{-T^2 \ + \ U^2}{T^2 \ + \ U^2} \qquad y \ = \ \frac{2TU}{T^2 \ + \ U^2}. \tag{7.3}$$

To plot the entire circle using equation (7.2), t would have to range from negative to positive infinity. An advantage to the homogeneous parameters is that the entire circle can be swept out with finite parameter values. This is done as follows:

- First Quadrant: $0 \leq T \leq 1$, $U = 1$
- Second Quadrant: $T = 1$, $1 \geq U \geq 0$
- Third Quadrant: $T = -1$, $0 \leq U \leq 1$
- Fourth Quadrant: $-1 \leq T \leq 0$, $U = 1$

7.1.4 The Fundamental Theorem of Algebra

The **fundamental theorem of algebra** can be thought of as a special case of Bezout's theorem. It states that a univariate polynomial of degree n has exactly n roots. That is, for $f(t) = a_0 + a_1 t + \cdots + a_n t^n$ there exist exactly n values of t for which $f(t) = 0$, if we count complex roots and possible multiple roots. If the a_i are all real, then any complex roots must occur in conjugate pairs. In other words, if the complex number $b + ci$ is a root of the real polynomial $f(t)$, then so also is $b - ci$.

7.2 The Intersection of Two Lines

A line is a degree one curve, so according to Bezout's theorem, two lines generally intersect in one point; if they intersect in more than one point, the two lines are identical. Several solutions to the problem of finding the point at which two lines intersect are presented in junior high school algebra courses. We present here an elegant solution involving homogeneous representation that is not found in junior high school algebra courses, and that properly accounts for intersections at infinity.

7.2.1 Homogeneous Points and Lines

We will denote by

$$\mathbf{P}(X, Y, W)$$

the point whose homogeneous coordinates are (X, Y, W) and whose Cartesian coordinates are $(x, y) = (X/W, Y/W)$. Likewise, we denote by

$$\mathbf{L}(a, b, c)$$

the line whose equation is $ax + by + c = aX + bY + cW = 0$.

The projection operator converts a point from homogeneous to Cartesian coordinates: $\Pi(\mathbf{P}(X, Y, W)) = (\frac{X}{W}, \frac{Y}{W})$.

The point $\mathbf{P}(X, Y, 0)$ lies at infinity, in the direction (X, Y). The line $\mathbf{L}(0, 0, 1)$ is called the line at infinite, and it includes all points at infinity.

Since points and lines can both be represented by triples of numbers, we now ask what role is played by the conventional operations of cross product

$$(a, b, c) \times (d, e, f) = (bf - ec, dc - af, ae - db)$$

and dot product

$$(a, b, c) \cdot (d, e, f) = ad + be + cf$$

are defined. The dot product determines incidence: point $\mathbf{P}(X, Y, W)$ lies on a line $\mathbf{L}(a, b, c)$ if and only if

$$\mathbf{P} \cdot \mathbf{L} = aX + bY + cW = 0.$$

The cross product has two applications: The line $\mathbf{L}$ containing two points $\mathbf{P}_1$ and $\mathbf{P}_2$ is given by $\mathbf{L} = \mathbf{P}_1 \times \mathbf{P}_2$ and the point $\mathbf{P}$ at which two lines $\mathbf{L}_1$ and $\mathbf{L}_2$ intersect is given by $\mathbf{P} = \mathbf{L}_1 \times \mathbf{L}_2$. This is an example of the principle of *duality* which, loosely speaking, means that general statements involving points and lines can be expressed in a reciprocal way. For example, "A unique line passes through two distinct points" has a dual expression, "A unique point lies at the intersection of two distinct lines".

Example The points $\mathbf{P}(2, 3, 1)$ and $\mathbf{P}(3, 1, 1)$ define a line $(2, 3, 1) \times (3, 1, 1) = \mathbf{L}(2, 1, -7)$. The points $\mathbf{P}(2, 3, 1)$ and $\mathbf{P}(1, 4, 1)$ define a line $(2, 3, 1) \times (1, 4, 1) = \mathbf{L}(-1, -1, 5)$. The lines $\mathbf{L}(2, 1, -7)$

and $\mathbf{L}(-1,-1,5)$ intersect in a point $(2,1,-7)\times(-1,-1,5) = \mathbf{P}(-2,-3,-1)$. Note that $\Pi(2,3,1) = \Pi(-2,-3,-1) = (2,3)$.

We noted that Bezout's theorem requires that we properly account for complex, infinite, and multiple intersections, and that homogeneous equations allow us to deal with points at infinity. We now illustrate. Students in junior high school algebra courses are taught that there is no solution for two equations representing the intersection of parallel lines, such as

$$3x + 4y + 2 = 0, \qquad 3x + 4y + 3 = 0.$$

However, using homogeneous representation, we see that the intersection is given by

$$\mathbf{L}(3,4,2) \times \mathbf{L}(3,4,3) = \mathbf{P}(-4,3,0)$$

which is the point at infinity in the direction $(-4,3)$.

7.3 Intersection of a Parametric Curve and an Implicit Curve

The points at which a parametric curve $(x(t), y(t))$ intersects an implicit curve $f(x,y) = 0$ can be found as follows. Let $g(t) = f(x(t), y(t))$. Then the roots of $g(t) = 0$ are the parameter values at which the parametric curve intersects the implicit curve. For example, let

$$x(t) = 2t - 1, \qquad y(t) = 8t^2 - 9t + 1$$

and

$$f(x,y) = x^2 + y^2 - 1$$

Then

$$g(t) = f(x(t), y(t)) = 64t^4 - 144t^3 + 101t^2 - 22t + 1$$

The roots of $g(t) = 0$ are $t = 0.06118$, $t = 0.28147$, $t = 0.90735$, and $t = 1.0$. The Cartesian coordinates of the intersection points can then be found by evaluating the parametric curve at those values of t: $(-0.8776, 0.47932)$, $(-0.43706, -0.89943)$, $(0.814696, -0.5799)$, $(1, 0)$.

If the parametric curve is rational,

$$x = \frac{a(t)}{c(t)}, \quad y = \frac{b(t)}{c(t)}$$

it is more simple to work in homogeneous form:

$$X = a(t), \quad Y = b(t), \quad W = c(t).$$

For example, to intersect the curves

$$x = \frac{t^2 + t}{t^2 + 1}, \quad y = \frac{2t}{t^2 + 1}$$

and

$$f(x,y) = x^2 + 2x + y + 1 = 0$$

we homogenize the implicit curve

$$f(X,Y,W) = X^2 + 2XW + YW + W^2 = 0$$

and the parametric curve

$$X(t) = t^2 + t, \quad Y(t) = 2t, \quad W(t) = t^2 + 1$$

and make the substitution

$$g(t) = f(X(t), Y(t), W(t)) = 4t^4 + 6t^3 + 5t^2 + 4t + 1$$

In this case, $g(t)$ has two real roots at $t = 1$ and $t = 0.3576$ and two complex roots.

This method is very efficient because it reduces the intersection problem to a polynomial root finding problem. If both curves are parametric, it is possible to convert one of them to implicit form using *implicitization*. The implicitization curve intersection algorithm is briefly sketched in section 17.8. A curve intersection algorithm based on implicitization is very fast, although it has some limitations. For example, if the degree of the two curves is large, the method is slow and experiences significant floating point error propagation. For numerical stability, if the computations are to be performed in floating points, one should implicitize in the Bernstein basis as discussed in section 17.6. But even this does not adequately reduce floating point error if two two curves are, say, degree ten.

7.3.1 Order of Contact

Given a degree m parametric curve

$$\mathbf{P}(t) = (X(t), Y(t), W(t))$$

and a degree n implicit curve

$$f(X, Y, W) = 0$$

the polynomial $g(t) = f(X(t), Y(t), W(t))$ is degree mn and, according to the fundamental theorem of algebra, it will have mn roots. This is a substantiation of Bezout's theorem.

A point on the curve $f(X, Y, W) = 0$ is a *singular* point if $f_X = f_Y = f_W = 0$. A point that is not singular is called a simple point.

If $g(\tau) = 0$, then $\mathbf{P}(\tau)$ is a point of intersection and $g(t)$ can be factored into $g(t) = (1 - \tau)^k \tilde{g}(t)$ where k is the *order of contact* between the two curves. If $\mathbf{P}(\tau)$ is a simple point on $f(X, Y, W) = 0$, then the two curves are G^{k-1} continuous at $\mathbf{P}(\tau)$.

Example The intersection between the line $X(t) = -t + 1$, $Y(t) = t$, $W(t) = 1$ and the curve $X^2 + Y^2 - W^2 = 0$ yields $g(t) = 2t^2 - 2t$, so there are two intersections, at $t = 0$ and $t = 1$. The Cartesian coordinates of the intersection points are $(1, 0)$ and $(0, 1)$.

Example The intersection between the line $X(t) = 1 + t^2$, $Y(t) = t$, $W(t) = 1$ and the curve $X^2 + Y^2 - W^2 = 0$ yields $g(t) = t^2$, so there is an intersection at $t = 0$ with order of contact of 2. These two curves are tangent at $(1, 0)$.

Example The intersection between the line $X(t) = 1 - t^2$, $Y(t) = t + t^2$, $W(t) = 1$ and the curve $X^2 + Y^2 - W^2 = 0$ yields $g(t) = 4t^3 + 3t^4$, so there is an intersection at $t = 0$ with order of contact of 3, plus an intersection at $t = -\frac{4}{3}$ with order of contact 1. These two curves are curvature continuous at $(1, 0)$.

The order-of-contact concept gives a powerful way to analyze curvature: The osculating circle is the unique circle that intersects $\mathbf{P}(t)$ with an order of contact of three.

Example $f(X, Y, W) = X^2 + Y^2 - 2\rho YW = 0$ is a circle of radius ρ with a center at $(0, \rho)$. What is the curvature of the curve $X(t) = x_1 t + x_2 t^2$, $Y(t) = y_2 t^2$, $W(t) = w_0 + w_1 t + w_2 t^2$? For these two curves,

$$g(t) = (x_1^2 - 2\rho w_0 y_2)t^2 + (2x_1 x_2 - 2\rho w_1 y_2)t^3 + (x_2^2 - 2\rho w_2 y_2 + y_2^2)t^4$$

For any value of ρ these two curves are tangent continuous. They are curvature continuous if $x_1^2 - 2\rho w_0 y_2 = 0$, which happens if

$$\rho = \frac{x_1^2}{2w_0 y_2}$$

Therefore, the curvature of $\mathbf{P}(t)$ at $t = 0$ is

$$\kappa = \frac{2w_0 y_2}{x_1^2}.$$

7.4 Computing the Intersection of Two Bézier Curves

Several algorithms address the problem of computing the points at which two Bézier curves intersect. Predominant approaches are the Bézier subdivision algorithm [LR80], the interval subdivision method adapted by Koparkar and Mudur [KM83], implicitization [SP86a], and Bézier clipping [SN90].

7.4.1 Timing Comparisons

A few sample timing comparisons for these four methods are presented in [SN90]. Comparative algorithm timings can of course change somewhat as the implementations are fine tuned, if tests are run on different computers, or even if different compilers are used. These timing tests were run on a Macintosh II using double precision arithmetic, computing the answers to eight decimal digits of accuracy.

The columns in Table 7.1 indicate the relative execution time for the algorithms *clip* = Bézier clipping algorithm, *Impl.* = implicitization, *Int* = Koparkar's interval algorithm and *Sub* = the conventional Bézier subdivision algorithm. In general, the implicitization intersection algorithm is

Example	Degree	Clip	Impl.	Int	Sub.
1	3	2.5	1	10	15
2	3	1.8	1	5	6
3	5	1	1.7	3	5
4	10	1	na	2	4

Table 7.1: Relative computation times

only reliable for curves of degree up to five using double precision arithmetic. For higher degrees, it is possible for the polynomial condition to degrade so that no significant digits are obtained in the answers. For curves of degree less than five, the implicitization algorithm is typically 1-3 times faster than the Bézier clip algorithm, which in turn is typically 2-10 times faster than the other two algorithms. For curves of degree higher than four, the Bézier clipping algorithm generally wins.

A brief discussion of these curve intersection methods follows.

7.5 Bézier subdivision

The Bézier subdivision curve intersection algorithm relies on the convex hull property and the de Casteljau algorithm. Though we overview it in terms of Bézier curves, it will work for any curve

which obeys the convex hull property. Figure 7.1 shows the convex hull of a single Bézier curve, and the convex hulls after subdividing into two and four pieces.

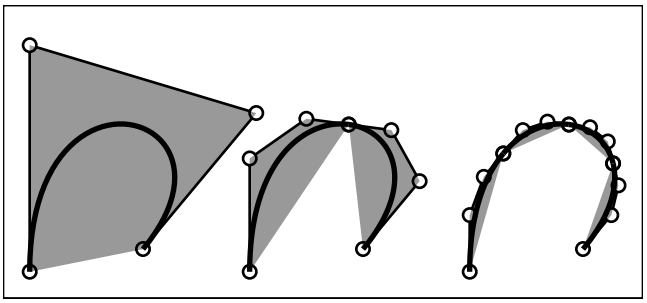


Figure 7.1: Convex Hulls

The intersection algorithm proceeds by comparing the convex hulls of the two curves. If they do not overlap, the curves do not intersect. If they do overlap, the curves are subdivided and the two halves of one curve are checked for overlap against the two halves of the other curve. As this procedure continues, each iteration rejects regions of curves which do not contain intersection points. Figure 7.2 shows three iterations.

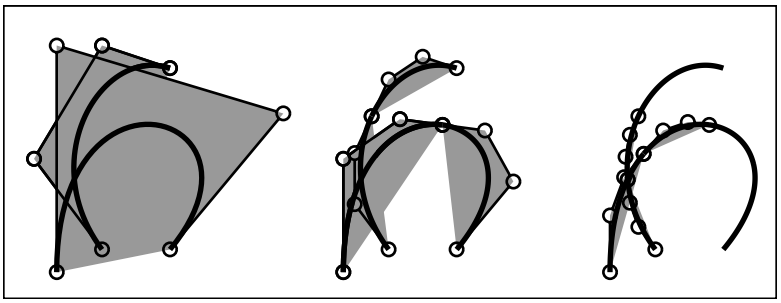


Figure 7.2: Three iterations of Bézier subdivision

Once a pair of curves has been subdivided enough that they can each be approximated by a line segment to within a tolerance ϵ (as given in equation 10.4), the intersection of the two approximating line segments is found.

Since convex hulls are rather expensive to compute and to determine overlap, in practice one normally just uses the $x - y$ bounding boxes.

7.6 Interval subdivision

The interval subdivision algorithm method is similar in spirit to the Bézier subdivision method. In this case, each curve is preprocessed to determine its vertical and horizontal tangents, and divided into "intervals" which have such tangents only at endpoints, if at all. Note that within any such interval, a rectangle whose diagonal is defined by any two points on the curve completely bounds the curve between those two endpoints. The power of this method lies in the fact that subdivision can now be performed by merely evaluating the curve (using Horner's method instead of the more

expensive de Casteljau algorithm) and that the bounding box is trivial to determine. Figure 7.3 illustrates this bounding box strategy.

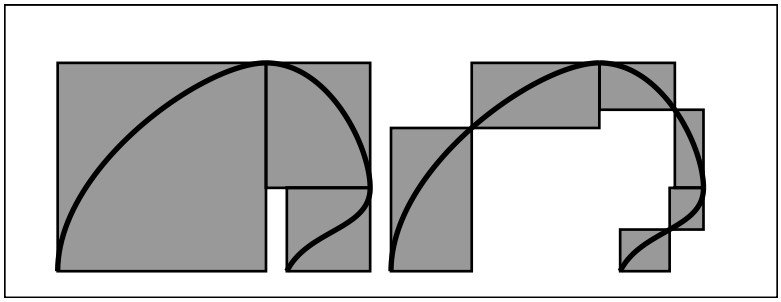


Figure 7.3: Interval preprocess and subdivision

7.7 Bézier Clipping method

The method of Bézier clipping has many applications, such as ray tracing trimmed rational surface patches [NSK90], algebraic curve intersection [Sed89], and tangent intersection computation [SN90]. It can be viewed as kind of an interval Newton's method, because it has quadratic convergence, but robustly finds all intersections. Since it is such a powerful tool, and since it is based on some ideas that haven't been discussed previously in these notes, the majority of this chapter is devoted to this method.

7.7.1 Fat Lines

Define a *fat line* as the region between two parallel lines. Our curve intersection algorithm begins by computing a fat line which bounds one of the two Bézier curves. Similar bounds have been suggested in references [Bal81, SWZ89].

Denote by $\bar{L}$ the line $\mathbf{P}_0 - \mathbf{P}_n$. We choose a fat line parallel to $\bar{L}$ as shown in Figure 7.4. If $\bar{L}$ is

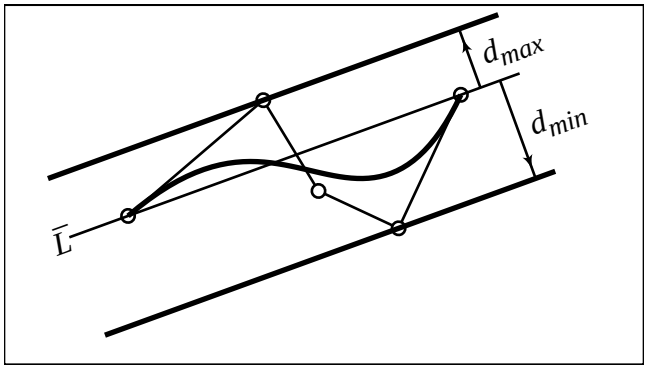


Figure 7.4: Fat line bounding a quartic curve

defined in its normalized implicit equation

$$ax + by + c = 0 \quad (a^2 + b^2 = 1) \tag{7.4}$$

then, the distance $d(x, y)$ from any point (x, y) to $\bar{L}$ is

$$d(x, y) = ax + by + c \tag{7.5}$$

Denote by $d_i = d(x_i, y_i)$ the signed distance from control point $\mathbf{P}_i = (x_i, y_i)$ to $\bar{L}$. By the convex hull property, a fat line bounding a given rational Bézier curve with non-negative weights can be defined as the fat line parallel to $\bar{L}$ which most tightly encloses the Bézier control points:

$$\{(x, y) | d_{min} \leq d(x, y) \leq d_{max}\} \tag{7.6}$$

where

$$d_{min} = \min\{d_0, \ldots, d_n\}, \quad d_{max} = \max\{d_0, \ldots, d_n\}. \tag{7.7}$$

7.7.2 Bézier Clipping

Figure 7.5 shows two polynomial cubic Bézier curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$, and a fat line $\mathbf{L}$ which bounds $\mathbf{Q}(u)$. In this section, we discuss how to identify intervals of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L}$, and hence for which $\mathbf{P}(t)$ does not intersect $\mathbf{Q}(u)$.

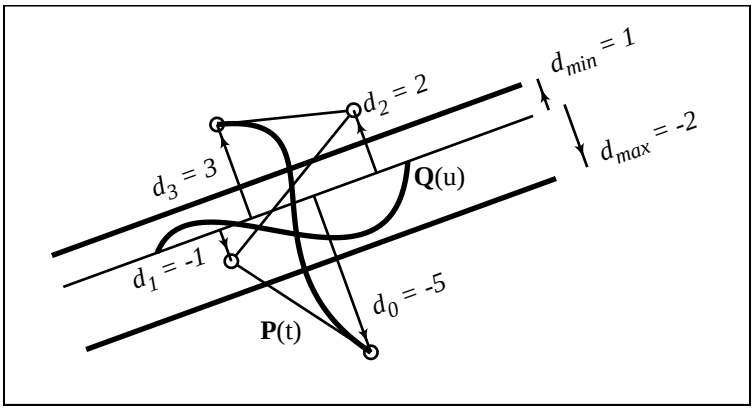


Figure 7.5: Bézier curve/fat line intersection

$\mathbf{P}$ is defined by its parametric equation

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t) \tag{7.8}$$

where $\mathbf{P}_i = (x_i, y_i)$ are the Bézier control points, and $B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$ denote the Bernstein basis functions. If the line $\bar{L}$ through $\mathbf{Q}_0 - \mathbf{Q}_n$ is defined by

$$ax + by + c = 0 \quad (a^2 + b^2 = 1), \tag{7.9}$$

then the distance $d(t)$ from any point $\mathbf{P}(t)$ to $\bar{L}$ can be found by substituting equation 7.8 into equation 7.9:

$$d(t) = \sum_{i=0}^{n} d_i B_i^n(t), \quad d_i = ax_i + by_i + c. \tag{7.10}$$

Note that $d(t) = 0$ for all values of t at which $\mathbf{P}$ intersects $\bar{L}$. Also, d_i is the distance from $\mathbf{P}_i$ to $\bar{L}$ (as shown in Figure 7.5).

The function $d(t)$ is a polynomial in Bernstein form, and can be represented as an explicit Bézier curve (Section 2.14) as follows:

$$\mathbf{D}(t) = (t, d(t)) = \sum_{i=0}^{n} \mathbf{D}_i B_i^n(t). \tag{7.11}$$

Figure 7.6 shows the curve $\mathbf{D}(t)$ which corresponds to Figure 7.5.

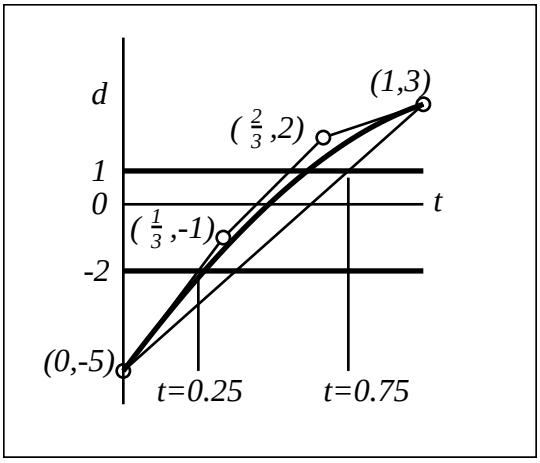


Figure 7.6: Explicit Bézier curve

Values of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L}$ correspond to values of t for which $\mathbf{D}(t)$ lies above $d = d_{max}$ or below $d = d_{min}$. We can identify parameter ranges of t for which $\mathbf{P}(t)$ is guaranteed to lie outside of $\mathbf{L}$ by identifying ranges of t for which the *convex hull* of $\mathbf{D}(t)$ lies above $d = d_{max}$ or below $d = d_{min}$. In this example, we are assured that $\mathbf{P}(t)$ lies outside of $\mathbf{L}$ for parameter values $t < 0.25$ and for $t > 0.75$.

Bézier clipping is completed by subdividing $\mathbf{P}$ twice using the de Casteljau algorithm, such that portions of $\mathbf{P}$ over parameter values $t < 0.25$ and $t > 0.75$ are removed.

Figure 7.6 shows how to clip against a fat line using a single explicit Bézier curve. This approach only works for polynomial Bézier curves. For rational Bézier curves, the explicit Bézier curves generated for each of the two lines are not simple translations of each other, so we must clip against each of the two lines separately. This is illustrated in Sections 7.7.7 and 7.7.8

7.7.3 Iterating

We have just discussed the notion of Bézier clipping in the context of curve intersection: regions of one curve which are guaranteed to not intersect a second curve can be identified and subdivided

away. Our Bézier clipping curve intersection algorithm proceeds by iteratively applying the Bézier clipping procedure.

Figure 7.7 shows curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$ from Figure 7.5 after the first Bézier clipping step in which regions $t < 0.25$ and $t > 0.75$ have been clipped away from $\mathbf{P}(t)$. The clipped portions of $\mathbf{P}(t)$ are shown in fine pen width, and a fat line is shown which bounds $\mathbf{P}(t)$, $0.25 \leq t \leq 0.75$. The next step in the curve intersection algorithm is to perform a Bézier clip of $\mathbf{Q}(u)$, clipping away regions of $\mathbf{Q}(u)$ which are guaranteed to lie outside the fat line bounding $\mathbf{P}(t)$. Proceeding as before, we

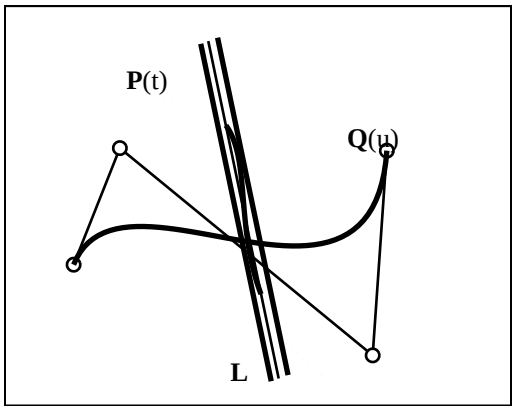


Figure 7.7: After first Bézier clip

define an explicit Bézier curve which expresses the distance from $\bar{L}$ in Figure 7.7 to the curve $\mathbf{Q}(u)$, from which we conclude that it is safe to clip off regions of $\mathbf{Q}(u)$ for which $u < .42$ and $u > .63$.

Next, $\mathbf{P}(t)$ is again clipped against $\mathbf{Q}(u)$, and so forth. After three Bézier clips on each curve, the intersection is computed to within six digits of accuracy (see Table 7.2).

Step	$\mathbf{t_{min}}$	$\mathbf{t_{max}}$	$\mathbf{u_{min}}$	$\mathbf{u_{max}}$
0	0	1	0	1
1	0.25	0.75	0.4188	0.6303
2	0.3747	0.4105	0.5121	0.5143
3	0.382079	0.382079	0.512967	0.512967

Table 7.2: Parameter ranges for $\mathbf{P}(t)$ and $\mathbf{Q}(u)$.

7.7.4 Clipping to other fat lines

The fat line defined in section 7.7.1 provides a nearly optimal Bézier clip in most cases, especially after a few iterations. However, it is clear that *any* pair of parallel lines which bound the curve can serve as a fatline. In some cases, a fat line perpendicular to line $\mathbf{P}_0 - \mathbf{P}_n$ provides a larger Bézier clip than the fat line parallel to line $\mathbf{P}_0 - \mathbf{P}_n$. We suggest that in general it works best to examine both fat lines to determine which one provides the largest clip. Experience has shown this extra overhead to reap a slightly lower average execution time.

7.7.5 Multiple Intersections

Figure 7.8 shows a case where two intersection points exist. In such cases, iterated Bézier clipping

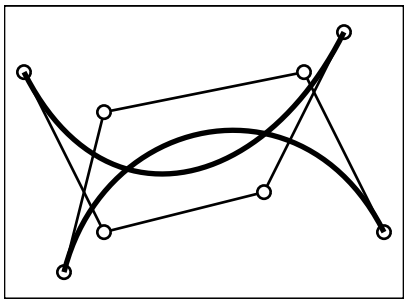


Figure 7.8: Two intersections

cannot converge to a single intersection point. The remedy is to split one of the curves in half and to compute the intersections of each half with the other curve, as suggested in Figure 7.9. A stack

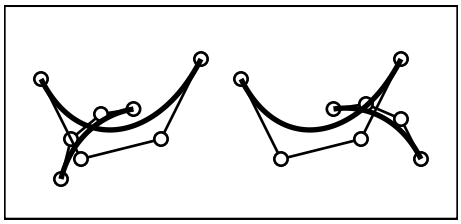


Figure 7.9: Two intersections, after a split

data structure is used to store pairs of curve segments, as in the conventional divide-and-conquer intersection algorithm [LR80].

Experimentation suggests the following heuristic. If a Bézier clip fails to reduce the parameter range of either curve by at least 20%, subdivide the "longest" curve (largest remaining parameter interval) and intersect the shorter curve respectively with the two halves of the longer curve. This heuristic, applied recursively if needed, allows computation of arbitrary numbers of intersections.

7.7.6 Rational Curves

If $\mathbf{P}$ is a *rational* Bézier curve

$$\mathbf{P}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{P}_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)} \tag{7.12}$$

with control point coordinates $\mathbf{P}_i = (x_i, y_i)$ and corresponding non-negative weights w_i, the Bézier clip computation is modified as follows. Substituting equation 7.12 into equation 7.9 and clearing the denominator yields:

$$d(t) = \sum_{i=0}^{n} d_i B_i^n(t), \quad d_i = w_i(ax_i + by_i + c).$$

The equation $d(t) = 0$ expresses the intersection of $\mathbf{P}(t)$ with a line $ax + by + c = 0$. However, unlike the non-rational case, the intersection of $\mathbf{P}(t)$ with a fat line *cannot* be represented as $\{(x, y) = \mathbf{P}(t) | d_{min} \leq d(t) \leq d_{max}\}$. Instead, $\mathbf{P}$ must be clipped independently against each of the two lines bounding the fat line. Thus, we identify ranges of t for which

$$\sum_{i=0}^{n} w_i(ax_i + by_i + c - d_{max})B_i^n(t) > 0$$

or for which

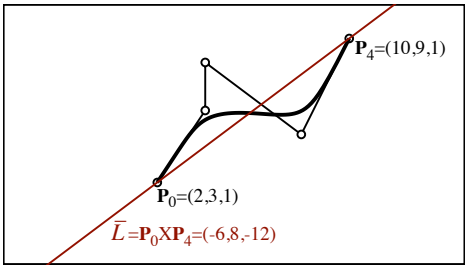
$$\sum_{i=0}^{n} w_i(ax_i + by_i + c + d_{min})B_i^n(t) < 0.$$

These ranges are identified using the Bézier clipping technique as previously outlined.

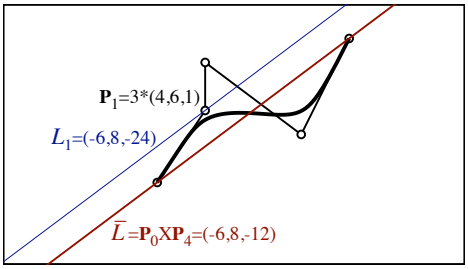
7.7.7 Example of Finding a Fat Line

We now run through an example of how to compute a fat line and perform a Bézier clip on a pair of rational Bézier curves. We use the notation for points and lines as triple of numbers, presented in Section 7.2. This leads to an elegant solution.

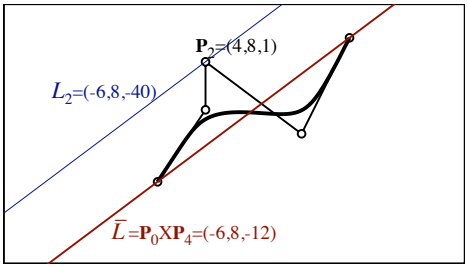
Figure 7.10.a shows a rational Bézier curve of degree $n = 4$. The control points are written in homogeneous form: $\mathbf{P}_i = w_i * (x_i, y_i, 1)$. For example, the notation $\mathbf{P}_1 = 3 * (4, 6, 1)$ means that the Cartesian coordinates are $(4, 6)$ and the weight is 3.



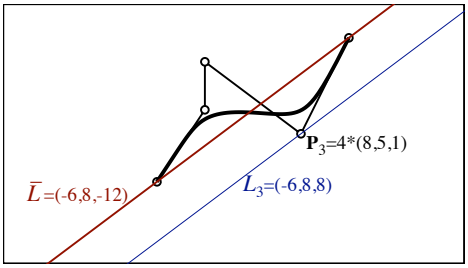
(a) Rational Bézier Curve. Line $\bar{L} = \mathbf{P}_0 \times \mathbf{P}_4$.



(b) Line L_1.



(c) Line L_1.



(d) Line L_1.

Figure 7.10: Example of how to find fat lines.

To compute a fat line that bounds this curve, begin by computing the base line $\bar{L} = \mathbf{P}_0 \times \mathbf{P}_n = (-6, 8, -12)$. Then, compute the lines L_i, $i = 1, \ldots, n-1$ that are parallel to $\bar{L}$ and that pass through $\mathbf{P}_i$. In general, if $\bar{L} = (a, b, c)$, then $L_i = (a, b, c_i)$. Thus, we must solve for c_i such that

$(a, b, c_i) \cdot \mathbf{P}_i = 0$. Thus,

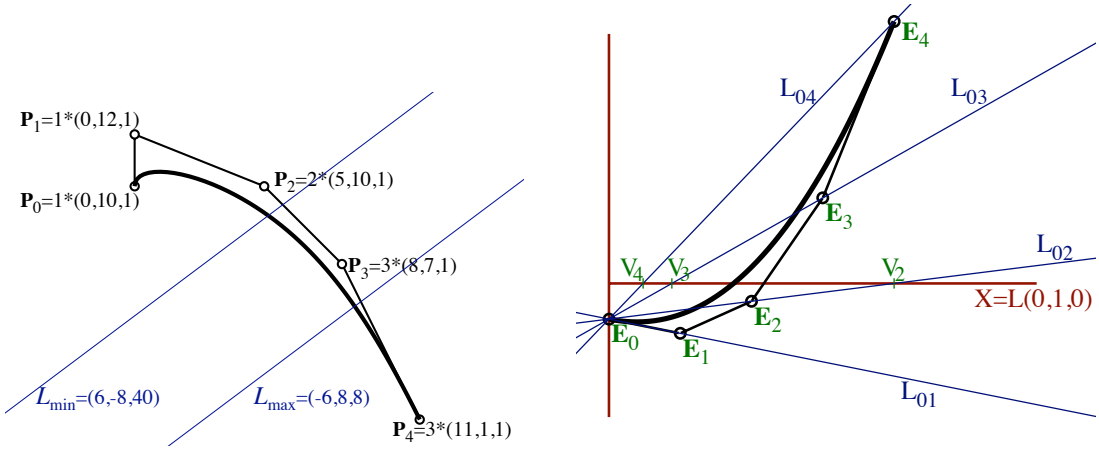
$$c_i = -ax_i - by_i.$$

Call the line with the smallest value of c_i L_{min} and the line with the largest value of c_i L_{max}. We want the curve to lie in the positive half space of each bounding line. It can be shown that this will happen if we scale L_{min} by -1.

Thus, in our example, we have

$$L_{min} = -L_2 = (6, -8, 40), \qquad L_{max} = L_3 = (-6, 8, 8). \tag{7.13}$$

7.7.8 Example of Clipping to a Fat Line



(a) Curve and fat line.

(b) Explicit Bézier Curve for Clipping against L_{min}.

Figure 7.11: Clipping to a fat line.

Figure 7.11.a shows a rational Bézier curve to be clipped against the fat line (7.13). Figure 7.11.b shows the resulting explicit Bézier curve where

$$E_i = (\frac{i}{n}, L_{min} \cdot \mathbf{P}_i, 1).$$

In this example,

$$E_0 = (0, -40, 1), \quad E_1 = (\frac{1}{4}, -56, 1), \quad E_2 = (\frac{1}{2}, -20, 1), \quad E_3 = (\frac{3}{4}, 96, 1), \quad E_4 = (1, 294, 1)$$

Note that we only want to clip away the portions of $\mathbf{P}(t)$ that lie in the negative half space of L_{min}. That half space corresponds to the values of t for which the explicit curve is < 0. Therefore, we can clip away the "left" portion of the curve if and only if $\mathbf{E}_0$ lies below the x-axis and we can clip away the "right" portion of the curve if and only if $\mathbf{E}_n$ lies below the x-axis. So in this example, we can compute a clip value at the "left" end of the curve, but not the "right" end.

To compute the clip value, we first compute the lines

$$L_{0,i} = E_0 \times E_i$$

and the points

$$L_{0,i} \times \mathbf{X} = V_i = (a_i, 0, c_i)$$

where $\mathbf{X} = (0, 1, 0)$ is the line corresponding to the x-axis ($y = 0$). In our example,

$$L_{01} = (16, \frac{1}{4}, 10), \quad L_{02} = (-20, \frac{1}{2}, 20), \quad L_{03} = (-136, \frac{3}{4}, 30), \quad L_{04} = (-334, 1, 40)$$

and

$$V_1 = (-10, 0, 16), \quad V_2 = (-20, 0, -20), \quad V_3 = (-30, 0, -136), \quad V_4 = (-40, 0, -334).$$

Our "clip" value will be the x-coordinate of the left-most V_i that is to the right of the origin. To determine this, we must project the homogeneous points V_i to their corresponding Cartesian points v_i which yields:

$$v_1 = (-\frac{5}{8}, 0), \quad v_2 = (1, 0), \quad v_3 = (\frac{30}{136}, 0) \approx (.2206, 0), \quad v_4 = (\frac{40}{334}, 0) \approx (.1198, 0).$$

So the desired t value at which to clip against L_{min} is .1198, and we can eliminate the domain $t \in [0, 0.1198)$.

We now clip against line L_{max}. The explicit Bézier curve is shown in Figure 7.12, for which

$$E_i = (\frac{i}{n}, L_{max} \cdot \mathbf{P}_i, 1).$$

In this example,

$$E_0 = (0, 88, 1), \quad E_1 = (\frac{1}{4}, 104, 1), \quad E_2 = (\frac{1}{2}, 116, 1), \quad E_3 = (\frac{3}{4}, 48, 1), \quad E_4 = (1, -150, 1)$$

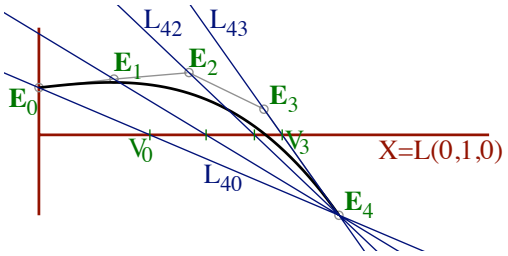


Figure 7.12: Clipping to L_{max}.

Since E_0 is above the x-axis and E_4 is below the x-axis, we can clip the right side of the curve but not the left. We have

$$L_{40} = (-238, -1, 88), \quad L_{41} = (-254, -\frac{3}{4}, 141.5), \quad L_{42} = (-266, -\frac{1}{2}, 191), \quad L_{43} = (-198, -\frac{1}{4}, 160.5)$$

and

$$V_0 = (-88, 0, -238), \quad V_1 = (-141.5, 0, -254), \quad V_2 = (-191, 0, -266), \quad V_3 = (-160.5, 0, -198).$$

Projecting the homogeneous points V_i to their corresponding Cartesian points v_i yields

$$v_0 \approx (.3697, 0), \quad v_1 \approx (.5571, 0), \quad v_2 \approx (.7180, 0), \quad v_3 \approx (.8106, 0).$$

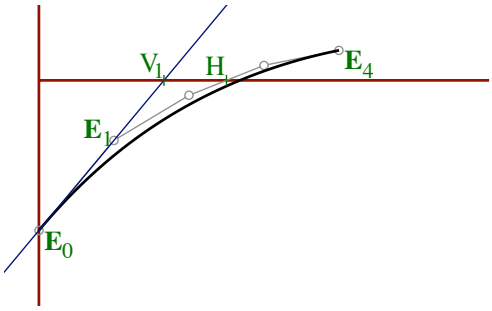
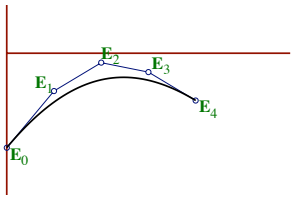


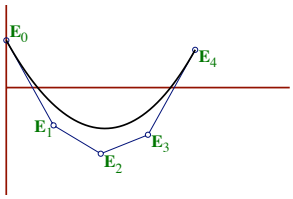
Figure 7.13: Clipping example.

When clipping away the right end of the curve, we need to identify the right-most V_i that is to the left of $(1, 0)$, which in this example is V_3. Thus, we can clip away $t \in (.8106, 1]$.

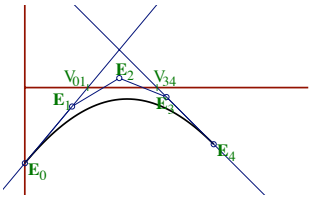
Note that we are not computing the exact intersection between the convex hull and the x-axis, as illustrated in Figure 7.13, where the algorithm described in this section would compute a clip value corresponding to V_1, whereas the convex hull crosses the x-axis at H. In our experiments, the method described here runs faster than the method where we compute the exact intersection with the convex hull because in most cases the two values are the same and the method described here is more simple.



(a) Clip away everything.



(b) Clip away nothing.



(c) Clip to the left of V_{01} and to the right of V_{34}.

Figure 7.14: Additional examples.

A few additional examples are illustrated in Figure 7.14. In Figure 7.14.a, all control points lie below the x-axis. In this case, there is no intersection. In Figure 7.14.b, E_0 and E_n lie above the x-axis. In this case, no clipping is performed. In Figure 7.14.c, clipping values are computed for both the left and right sides of the curve.

Chapter 8

Offset Curves

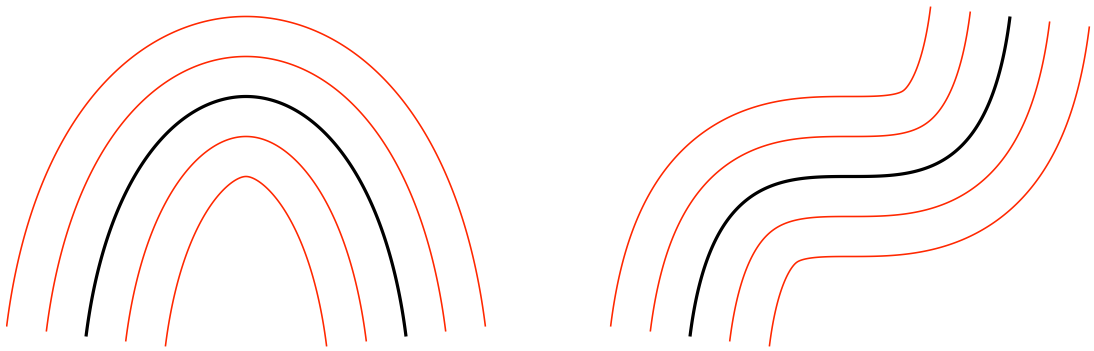
An offset curve is the set of all points that lie a perpendicular distance ρ from a given curve in R^2. The scalar ρ is called the *offset radius*. If the parametric equation of the given curve is

$$\mathbf{P}(t) = (x(t), y(t)) \tag{8.1}$$

then the offset curve with offset radius ρ is given by

$$\Omega(\rho, \mathbf{P}(t)) = \mathbf{P}(t) + \rho \frac{(y'(t), -x'(t))}{\sqrt{x'^2(t) + y'^2(t)}} \tag{8.2}$$

Note that in this definition, if ρ is positive, the offset is on our right as we walk along the base curve in the direction of increasing parameter value.



(a) Offsets of a convex curve.

(b) Offset of a curve that has an inflection point.

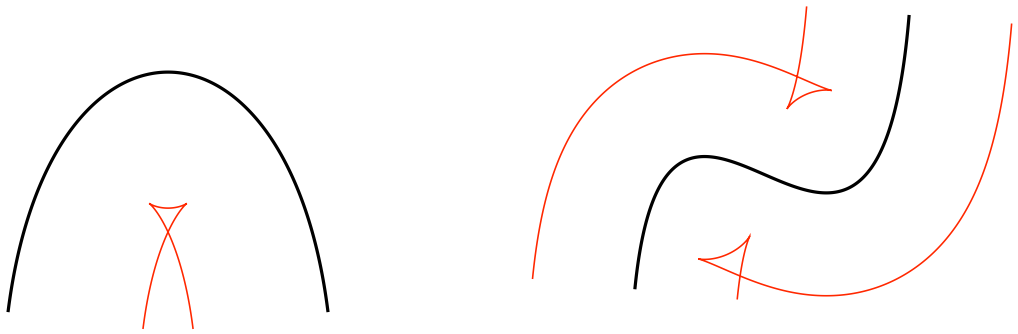
Figure 8.1: Offset Curves.

In Figure 8.1, the red curves are offsets of the black curves.

Offset curves play an important roll in computer aided design and manufacturing (CAD/CAM). If a numerically controlled machine is used to cut out a shape, the cutting tool has a finite radius.

Therefore, the path that the tool traverses is an offset curve and the offset radius is the radius of the cutting tool.

It is important that the tool radius is less than the minimum radius of curvature of the curve, otherwise the tool will perform unintended gouging. Figure 8.2 illustrates an offset curve whose radius exceeds the radius of curvature along part of the base curve. Every point at which the radius of curvature in the base curve matches the offset radius creates a cusp in the offset curve. At the cusp, the offset curve changes direction. Between cusps, the first derivative vectors of the base curve and offset curve point in opposite directions.



(a) Offsets of a convex curve.

(b) Offset of a curve that has an inflection point.

Figure 8.2: Offset Curves in which the Offset Radius Exceeds the Radius of Curvature for a Portion of the Base Curve.

If $\mathbf{P}$ is an offset of $\mathbf{Q}$, the reverse is generally true, as long as the offset radius is everywhere less than the radius of curvature of both curve segments:

$$\Omega(-\rho, \Omega(\rho, \mathbf{P}(t))) = \mathbf{P}(t). \tag{8.3}$$

Ini general, offset curves cannot be represented in Bézier form because (8.2) contains a square root of a polynomial. The obvious exceptions are circles and straight lines. A non-obvious exception is that the offset of any parabola can be represented as a degree eight rational Bézier curve.

In addition, a family of curves has been identified by Rida Farouki which can be represented in rational Bézier form. There curves are called Pythagorean Hodograph curves, and they defined by the property that the sum $x'^2(t) + y'^2(t)$ is a perfect square of a polynomial. Farouki has written many papers on this subject that detail some elegant geometric properties that can be imposed on the control polygon of a Bézier curve to assure that it satisfies the Pythagorean Hodograph requirement.

Chapter 9

Polynomial Root Finding in Bernstein Form

The Bernstein polynomial basis enjoys widespread use in the fields of CAGD and computer graphics. The use of the Bernstein basis functions to express Bézier curves and surfaces allows many basic algorithms concerned with the processing of such forms to be reduced to the problem of computing the real roots of polynomials in Bernstein form.

A typical example is the problem of computing the intersection points of two Bézier curves using the implicitization algorithm. Given two curves of degrees m and n, respectively, the problem of identifying their intersection points can be reformulated in terms of finding the real roots on the unit interval of a degree mn polynomial in Bernstein form [SP86a]. Other examples from CAGD include finding a point on each loop of the intersection curve of two surfaces and the "trimming" of offset and bisector curves. Examples from computer graphics include ray tracing and algebraic surface rendering.

In the early years of computer graphics and CAGD, the tendency was to convert from the Bernstein to the power basis for root finding since the power basis is more "familiar" and library routines are commonly available to solve for polynomial roots in the power basis. However, it has subsequently been demonstrated [FR87] that the Bernstein basis is inherently better conditioned than the power basis for finding real roots on the unit interval (and this is all that is of interest when dealing with Bézier curves anyway). Thus, root finding algorithms for polynomials in Bernstein form are not merely a convenience, they in fact offer significantly better accuracy than their power basis counterparts.

Furthermore, by expressing Bernstein form polynomials as explicit Bézier curves, one can take advantage of geometric insights based on the control polygon in developing root–finding algorithms. This can allow us to robustly find all real roots in the unit interval, typically much faster than can a library polynomial root finder which finds all real *and* complex roots. Thus, a root finder that works in the Bernstein basis could provide enhanced speed and accuracy when applied to problems whose solution traditionally relies on the computation of real polynomial roots in the power basis.

A detailed study of root finding algorithms for polynomials in Bernstein form is found in [Spe94]. In this chapter we review two basic approaches. The fundamental idea it to express a polynomial as an explicit Bézier curve (Section 2.14).

9.1 Convex Hull Marching

The *Bernstein convex hull approximating step algorithm* finds each real root of $[0,1]$ in ascending order by computing a sequence of subdivided explicit Bézier curves which are guaranteed to not skip over a real root. Each step is determined by exploiting the convex hull property which assures that all roots lie within the interval where the convex hull of the control polygon intersects the t-axis. A root is realized when its respective sequence of approximating steps converge to a limit.

This algorithm is quite similar to Newton's method, but it has the advantage that it always approaches the root from the left, and hence it cannot diverge like Newton's method.

This algorithm was first overviewed in [SP86a]. Since then, it has been adopted to various applications and to higher dimensions under a technique referred to as Bézier clipping [SWZ89, Sed89, SN90, NSK90].

The algorithm is illustrated in Figure 9.1. Begin by determining the left-most intersection between the convex-hull of the explicit Bézier curve $f_{[0,1]}(t)$. and the t axis. No roots can exist in the interval $[0, t_1]$ in Figure 9.1.a. Next, perform the de Casteljau algorithm (Figure 9.1.b) yielding $f_{[t_1,1]}(t)$ and find the value t_2 where the new convex hull intersects the t axis. These two operations are repeated until $t_{k+1} = t_k$ to within floating point tolerance, and t_k is taken to be a root. In this case, $k = 6$. To within floating point tolerance, $f_0 = 0$ in the explicit Bézier curve $f_{[t_6,1]}(t)$.

One pitfall to be aware of is that, due to floating point roundoff, it is possible (although rare) for the algorithm to sneak past a root. To avoid this problem, you should check to make sure that, at each iteration, $f_{[t_i,1]}(t_i)$ and $f_{[t_{i+1},1]}(t_{i+1})$ have the same sign. If they do not, t_i and t_{i+1} will differ only by floating point error. You should take $f_{[t_i,1]}(t_i)$ to have a root at t_i and proceed with the deflation step.

When a root is found, it is "deflated" out and the algorithm proceeds as before to find the next root. Deflation is the process of finding a degree $n-1$ polynomial $f^1_{[t_k,1]}(t)$which has the same roots as $f_{[t_k,1]}(t)$ except for the root at t_k. If we denote the deflated polynomial by $f^1_{[t_k,1]}(t)$, then

$$f_{[t_k,1]}(t) = (t - t_k) f^1_{[t_k,1]}(t).$$

The coefficients of $f^1_{[t_k,1]}(t)$ are

$$y^1_i = \frac{n}{i+1} y_{i+1}, \quad i = 0, \ldots, n-1.$$

This can be seen from the following derivation:

$$
\begin{aligned}
f_{[0,1]}(t) &= \sum_{i=1}^{n} y_i B_i^n(t) \\
&= \sum_{i=1}^{n} y_i \binom{n}{i} (1-t)^{n-i} t^i \\
&= t \sum_{i=0}^{n-1} y_{i+1} \binom{n}{i+1} (1-t)^{n-1-i} t^i \\
&= t \sum_{i=0}^{n-1} \frac{\binom{n}{i+1}}{\binom{n-1}{i}} y_{i+1} B_i^{n-1} \\
&= t \sum_{i=0}^{n-1} \frac{n}{i+1} y_{i+1} B_i^{n-1}
\end{aligned}
\tag{9.1}
$$

This algorithm performs reliably. It is numerically stable. The main drawback is the expense of doing a de Casteljau subdivision at each step.

9.2 Bernstein Combined Subdivide & Derivative Algorithm

In general, the most efficient root finding algorithm for polynomials of degree larger than seven is one which isolates roots and then refines them using the modified regula falsi method. A root is isolated if there is exactly one sign change in the Bernstein coefficients (this is due to the variation diminishing property).

The root isolation heuristic is illustrated in Figures 9.2 and 9.3. The basic idea is to perform a binary search to isolate the left-most root. If the root is not isloated after a few de Casteljau subdivisions (first at t=.5, then t=.25, then t=.125), the algorithm computes the left-most root of the derivative polynomial and uses that value to isolate the left-most root of the polynomial.

Figure 9.2.a shows an initial degree five polynomial with simple roots at 0, 0.2, and 1, along with a double root at 0.6. Figure 9.2.b shows the polynomial after deflating the roots at 0 and 1. In Figure 9.2.c, the polynomial has been split in two pieces at τ_1 and the segment $\mathbf{P}^{(2)}_{[0,\tau_1]}(t)$ is determined to contain exactly one root since its control polygon crosses the t axis exactly once. In Figure 9.2.d, $\mathbf{P}^{(3)}_{[\tau_1,1]}(t)$ does not pass the isolation test because its control polygon crosses the t axis twice.

In Figure 9.3.e, $\mathbf{P}^{(3)}_{[\tau_1,1]}(t)$ is split and the right half is seen to have no roots, while the left half is indeterminate. Figure 9.3.f performs another subdivision, with the same results, as does Figure 9.3.g. At this point, the algorithm changes its heuristic for isolating the root from binary search to finding the left–most root of the derivative of $\mathbf{P}^{(6)}(t)$. That derivative, shown in Figure 9.3.h, has only one root in the interval. The root is refined and used to isolate the left–most root of $\mathbf{P}^{(6)}(t)$. In this case, the double root at 0.6 is identified by checking that the numerical value of $\mathbf{P}^{(6)}(0.6)$ is less than the computed running error bound.

This algorithm is generally faster than the one in Section 9.1 because the modified regula falsi refinement algorithm can be preformed using the $O(n)$ Horner's evaluation algorithm instead of the $O(n^2)$ de Casteljau algorithm.

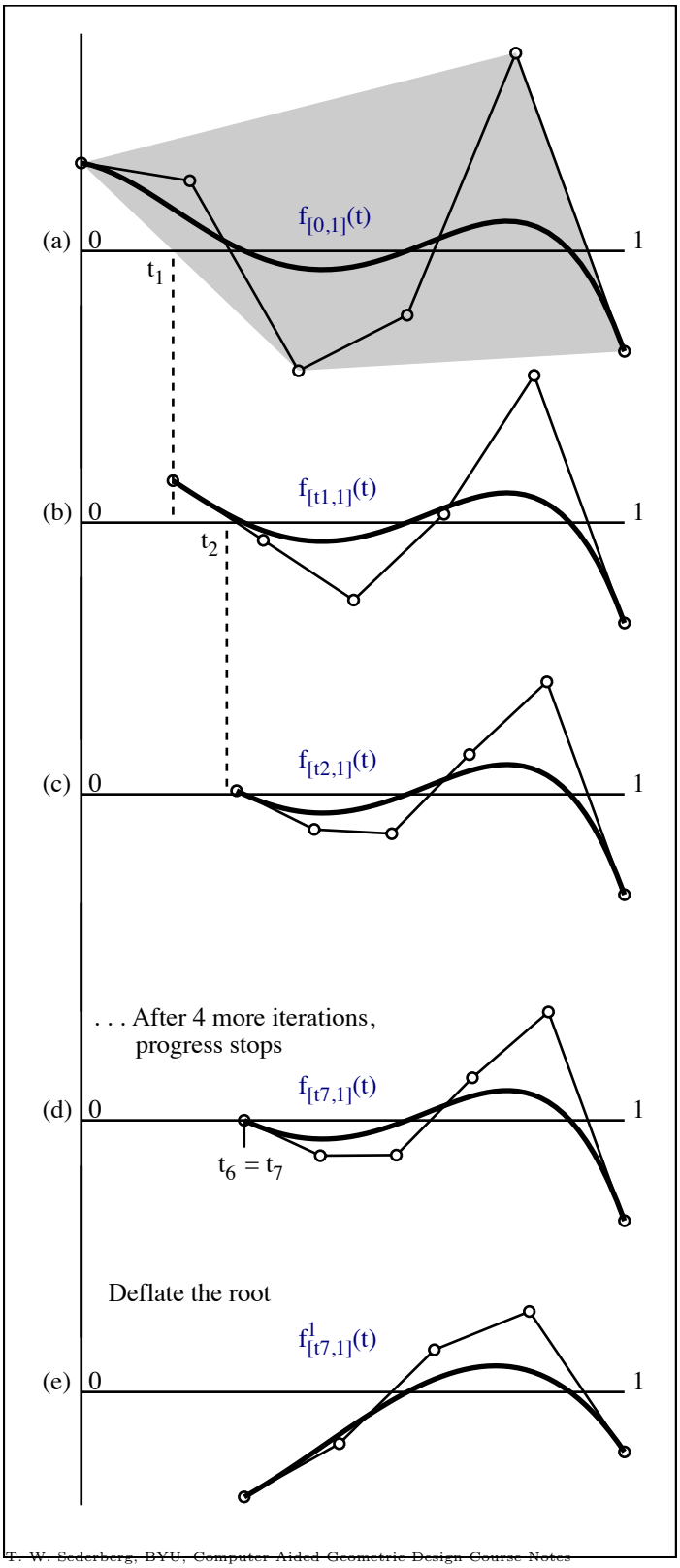


Figure 9.1: Bernstein root finding

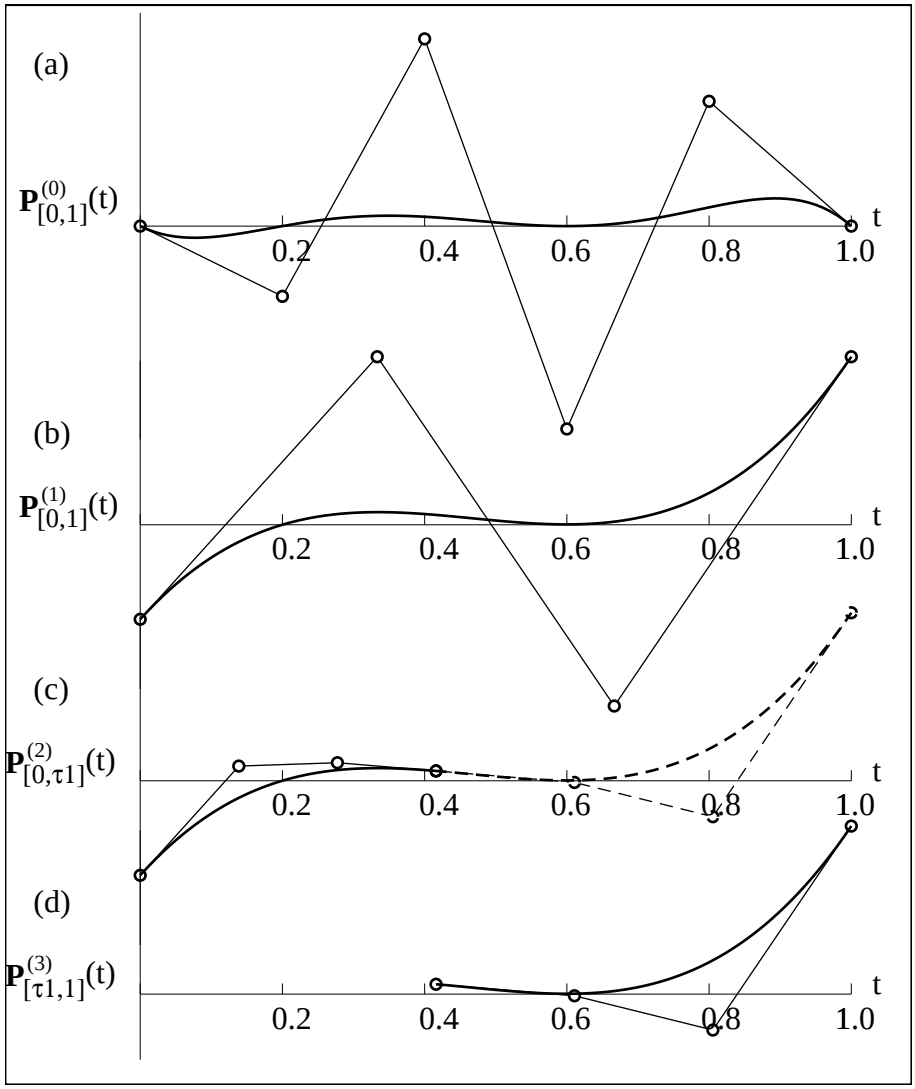


Figure 9.2: Root isolation heuristic (a-d).

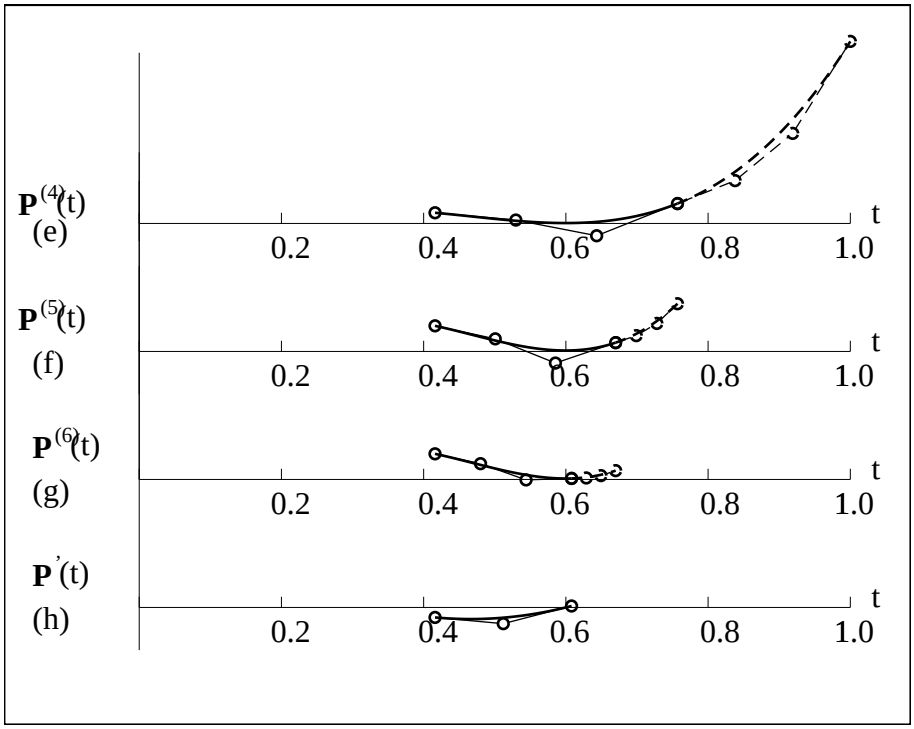


Figure 9.3: Root isolation heuristic (e-h).

9.3 Multiplication of Polynomials in Bernstein Form

Given

$$f(t) = \sum_{i=0}^{m} f_i B_i^m(t), \quad \text{and} \quad g(t) = \sum_{i=0}^{n} g_i B_i^n(t)$$

their product $h(t) = f(t)g(t)$ is a polynomial in Bernstein form of degree $m+n$ whose coefficients are

$$h_k = \frac{\sum_{i+j=k} \binom{m}{i} f_i \binom{n}{j} g_j}{\binom{m+n}{k}}$$

The notation $\sum_{i+j=k}$ means to use all valid values of i and j that add up to k. Denote

$$\langle f_0, f_1, \ldots, f_m \rangle^m(t) = \sum_{i=0}^{m} f_i B_i^m(t).$$

Then, for example,

$$\langle f_0, f_1, f_2 \rangle^2(t) \times \langle g_0, g_1, g_2, g_3 \rangle^3(t) =$$
$$\langle f_0 g_0, \frac{3f_0 g_1 + 2f_1 g_0}{5}, \frac{3f_0 g_2 + 6f_1 g_1 + f_2 g_0}{10}, \frac{f_0 g_3 + 6f_1 g_2 + 3f_2 g_1}{10}, \frac{2f_1 g_3 + 3f_2 g_2}{5}, f_2 g_3 \rangle$$

The binomial coefficients $\binom{n}{i}$ can be computed using the recurrence relation

$$\binom{n}{i} = \frac{n-i+1}{i} \binom{n}{i-1}.$$

Thus, $\binom{n}{0} = 1$, $\binom{n}{1} = n$, $\binom{n}{2} = \frac{n-1}{2}\binom{n}{1}$, etc.

9.4 Intersection between a Line and a Rational Bézier Curve

The points of intersection between a rational Bézier curve

$$\mathbf{P}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{P}_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}$$

and a line

$$ax + by + cw = 0$$

can be computed by substituting the parametric equation of the curve into the implicit equation of the line, producing a polynomial in Bernstein form

$$f(t) = \sum_{i=0}^{n} f_i B_i^n(t), \qquad f_i = a x_i w_i + b y_i w_i + c w_i.$$

The roots of $f(t)$ give the parameter values of the points at which the curve intersects the line.

Chapter 10

Polynomial Interpolation

10.1 Undetermined Coefficients

The method of undetermined coefficients provides a solution to the problem of finding a parametric curve which passes through a set of points. For example, suppose we wish to find a cubic parametric curve which passes through the points (0,0), (2,2), (0,3), and (2,4). We must first specify at what parameter value the curve will interpolate each point. Lets say we want (0,0) to have a parameter value of 0, (2,2) is to have a parameter value of 1/4, (0,3) should correspond to parameter $t = 3/4$, and (2,4) should have a parameter value of 1.

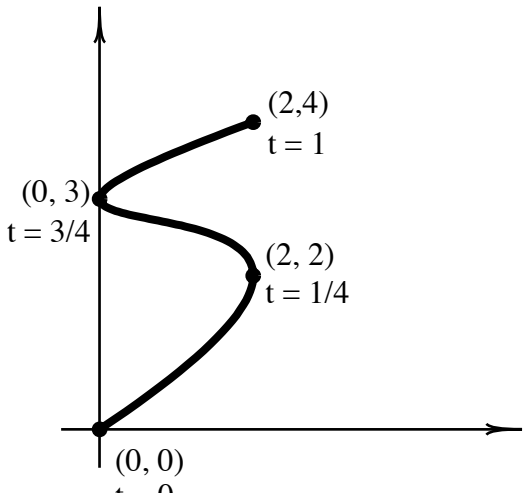


Figure 10.1: Interpolating Four Points

We can now pick any form we wish for the parametric equations. Let's first see how it is done for standard power basis polynomials, and then we will solve the same problem using Bernstein polynomials. For power basis polynomials, the parametric equations are of the form

$$x = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$y \;=\; b_0 \;+\; b_1 t \;+\; b_2 t^2 \;+\; b_3 t^3$$

To solve for the a_i, we set up four linear equations:

$$0 \;=\; a_0 \;+\; a_1 0 \;+\; a_2 0^2 \;+\; a_3 0^3$$

$$2 \;=\; a_0 \;+\; a_1(\frac{1}{4}) \;+\; a_2(\frac{1}{4})^2 \;+\; a_3(\frac{1}{4})^3$$

$$0 \;=\; a_0 \;+\; a_1(\frac{3}{4}) \;+\; a_2(\frac{3}{4})^2 \;+\; a_3(\frac{3}{4})^3$$

$$2 \;=\; a_0 \;+\; a_1 1 \;+\; a_2 1^2 \;+\; a_3 1^3$$

In matrix form, we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{4} & (\frac{1}{4})^2 & (\frac{1}{4})^3 \\ 1 & \frac{3}{4} & (\frac{3}{4})^2 & (\frac{3}{4})^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 2 \\ 0 \\ 2 \end{Bmatrix}.$$

from which

$$x \;=\; 18t \;-\; 48t^2 \;+\; 32t^3$$

Likewise for y, we solve the set of equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{4} & (\frac{1}{4})^2 & (\frac{1}{4})^3 \\ 1 & \frac{3}{4} & (\frac{3}{4})^2 & (\frac{3}{4})^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} \;=\; \begin{Bmatrix} 0 \\ 2 \\ 3 \\ 4 \end{Bmatrix}.$$

and we solve for

$$y \;=\; 12t \;-\; \frac{56}{3}t^2 \;+\; \frac{32}{3}t^3.$$

Let's next look at how we could solve directly for the Bézier control points of a Bézier curve which interpolates those same points at those same parameter values. The only difference between this problem and the one we just solved is the form of the polynomial. In this case, we want to solve for the coefficients of a Bernstein polynomial:

$$x \;=\; a_0(1-t)^3 \;+\; 3a_1 t(1-t)^2 \;+\; 3a_2 t^2(1-t) \;+\; a_3 t^3.$$

We evaluate this expression at $x = 0$, $t = 0$, again at $x = 2$, $t = \frac{1}{4}$, again at $x = 0$, $t = \frac{3}{4}$ and again at $x = 2$, $t = 1$ to produce a set of equations:

$$0 \;=\; a_0(1-0)^3 \;+\; 3a_1 0(1-0)^2 \;+\; 3a_2 0^2(1-0) \;+\; a_3 0^3.$$

$$2 \;=\; a_0(1-\frac{1}{4})^3 \;+\; 3a_1\frac{1}{4}(1-\frac{1}{4})^2 \;+\; 3a_2\frac{1}{4}^2(1-\frac{1}{4}) \;+\; a_3\frac{1}{4}^3.$$

$$0 \;=\; a_0(1-\frac{3}{4})^3 \;+\; 3a_1\frac{3}{4}(1-\frac{3}{4})^2 \;+\; 3a_2\frac{3}{4}^2(1-\frac{3}{4}) \;+\; a_3\frac{3}{4}^3.$$

$$2 \;=\; a_0(1-1)^3 \;+\; 3a_1 1(1-1)^2 \;+\; 3a_2 1^2(1-1) \;+\; a_3 1^3.$$

In matrix form, we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{27}{64} & \frac{27}{64} & \frac{9}{64} & \frac{1}{64} \\ \frac{1}{64} & \frac{9}{64} & \frac{27}{64} & \frac{27}{64} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 2 \\ 0 \\ 2 \end{Bmatrix}.$$

The x coordinates a_i of the Bézier control points work out to be $a_0 = 0$, $a_1 = 6$, $a_2 = -4$, and $a_3 = 2$. We can perform a similar computation to compute the y coordinates of the Bézier control points. They work out to be $0,\ 4,\ \frac{16}{9}$, and 4.

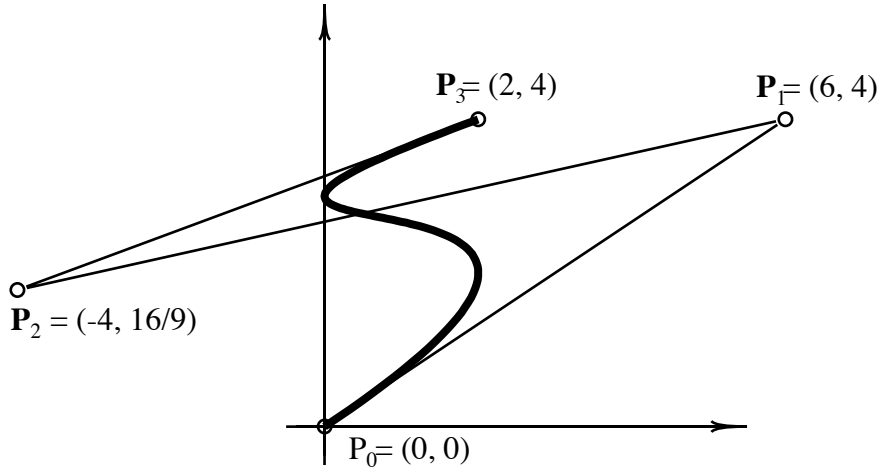


Figure 10.2: Interpolating Four Points

10.2 Lagrange Interpolation

There exists a clever set of basis polynomials which enable us to interpolate a set of points without having to solve a set of linear equations. These are known as Lagrange polynomials and are denoted $L_i^n(t)$. The purpose of Lagrange polynomials is to enable us, with virtually no computation, to find a degree n parametric curve which interpolates $n+1$ points $\mathbf{P}_0,\ \mathbf{P}_1,\ \ldots\ ,\mathbf{P}_n$ at parameter values $t_0,\ t_1,\ \ldots,\ t_n$. The curve is defined by

$$\mathbf{P}(t) = \mathbf{P}_0 L_0^n(t) + \mathbf{P}_1 L_1^n(t) + \ldots + \mathbf{P}_n L_n^n(t).$$

Note the following about the $L_i^n(t)$: $L_i^n(t_j) = 1$ whenever $i = j$ and $L_i^n(t_j) = 0$ whenever $i \neq j$. This must be so in order for the curve to interpolate point $\mathbf{P}_i$ at parameter value t_i. You can easily verify that the following choice for $L_i^n(t)$ satisfies those conditions:

$$L_i^n(t) = \frac{(t - t_0)(t - t_1) \ldots (t - t_{i-1})(t - t_{i+1}) \ldots (t - t_n)}{(t_i - t_0)(t_i - t_1) \ldots (t_i - t_{i-1})(t_i - t_{i+1}) \ldots (t_i - t_n)}$$

or

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

In the example from the previous section, we have

$$L_0^3(t) \;=\; \frac{(t-\frac{1}{4})(t-\frac{3}{4})(t-1)}{(0-\frac{1}{4})(0-\frac{3}{4})(0-1)} \;=\; -\frac{16}{3}(t-\frac{1}{4})(t-\frac{3}{4})(t-1)$$

$$L_1^3(t) \;=\; \frac{(t-0)(t-\frac{3}{4})(t-1)}{(\frac{1}{4}-0)(\frac{1}{4}-\frac{3}{4})(\frac{1}{4}-1)} \;=\; \frac{32}{3}t(t-\frac{3}{4})(t-1)$$

$$L_2^3(t) \;=\; \frac{(t-0)(t-\frac{1}{4})(t-1)}{(\frac{3}{4}-0)(\frac{3}{4}-\frac{1}{4})(\frac{3}{4}-1)} \;=\; -\frac{32}{3}t(t-\frac{1}{4})(t-1)$$

$$L_3^3(t) \;=\; \frac{(t-0)(t-\frac{1}{4})(t-\frac{3}{4})}{(1-0)(1-\frac{1}{4})(1-\frac{3}{4})} \;=\; \frac{16}{3}t(t-\frac{1}{4})(t-\frac{3}{4})$$

The interpolating curve is thus

$$\mathbf{P}(t) \;=\; \left\{ \begin{array}{c} x(t) \\ y(t) \end{array} \right\} \;=$$

$$\left\{ \begin{array}{c} 0 \\ 0 \end{array} \right\} L_0^3(t) \;+\; \left\{ \begin{array}{c} 2 \\ 2 \end{array} \right\} L_1^3(t) \;+\; \left\{ \begin{array}{c} 0 \\ 3 \end{array} \right\} L_2^3(t) \;+\; \left\{ \begin{array}{c} 2 \\ 4 \end{array} \right\} L_3^3(t)$$

If this expression is expanded out, it is seen to be identical to the equation we obtained using the method of undetermined coefficients.

It can be shown that the Lagrange blending functions are coordinate system independent, but do not obey the convex-hull property.

10.3 Newton Polynomials

Newton polynomials provide an additional method for finding a degree-n polynomial $p(t)$ that interpolates $n+1$ points:

$$p(t_i) = y_i, \qquad i = 0, \ldots, n.$$

Newton polynomials are defined

$$p_0(t) = a_0$$

$$p_1(t) = a_0 + a_1(t - t_0)$$

$$p_2(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)(t - t_1) \quad \ldots$$

$$p_n(t) = p_{n-1}(t) + a_n(t - t_0)(t - t_1)\cdots(t - t_{n-1}).$$

The coefficients a_i can be computed using *divided differences*, which can be found using a table that is reminiscent of forward differences. The notation for a divided difference is $f_{i,\ldots,k}$, and the coefficients a_i turn out to be:

$$a_0 = f_0, \quad a_1 = f_{0,1}, \quad a_2 = f_{0,1,2}, \ldots, a_n = f_{0,1,\ldots,n}.$$

The divided differences are computed using a recurrence relation:

$$f_i = y_i$$

$$f_{i,i+1} = \frac{f_{i+1} - f_i}{t_{i+1} - t_i}$$

$$f_{i,\ldots,k} = \frac{f_{i+1,\ldots,k} - f_{i,\ldots,k-1}}{t_k - t_i}$$

These divided differences are easily calculated by means of a divided difference table:

$$\begin{array}{c|ccccccc} t_0 & y_0 = f_0 & & & & & \\ t_1 & y_1 = f_1 & f_{0,1} & & & & \\ t_2 & y_2 = f_2 & f_{1,2} & f_{0,1,2} & & & \\ t_3 & y_3 = f_3 & f_{2,3} & f_{1,2,3} & f_{0,1,2,3} & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ t_n & y_n = f_n & f_{n-1,n} & \cdots & \cdots & \cdots & f_{0,1,\ldots n} \end{array}$$

Example
A polynomial parametric curve $\mathbf{P}(t)$ of degree three satisfies the following conditions:

$$t_0 = 0; \quad \mathbf{P}_0 = (0,0); \quad t_1 = 1, \quad \mathbf{P}_1 = (3,3); \quad t_2 = 3, \quad \mathbf{P}_2 = (5,3); \quad t_3 = 4, \quad \mathbf{P}_3 = (7,0).$$

Express $\mathbf{P}(t)$ using Newton polynomials and compute $\mathbf{P}(2)$.

Setting up the divided difference table:

$$\begin{array}{c|cccc} 0 & (0,0) & & & \\ 1 & (3,3) & (3,3) & & \\ 3 & (5,3) & (1,0) & (-\frac{2}{3},-1) & \\ 4 & (7,0) & (2,-3) & (\frac{1}{3},-1) & (\frac{1}{4},0) \end{array}$$

from which

$$\mathbf{P}(t) = (0,0) + (3,3)(t-0) + (-\frac{2}{3},-1)(t-0)(t-1) + (\frac{1}{4},0)(t-0)(t-1)(t-3)$$

from which

$$\mathbf{P}(2) = (\frac{25}{6}, 4).$$

The divided difference table is similar to forward differencing. However, with forward differencing, it is only possible to evaluate a polynomial at evenly spaced parameter values, whereas with a divided difference table, you can evaluate the polynomial at any parameter value. We can also evaluate a polynomial directly from the divided difference table (in a manner similar to forward differencing), but it is more expensive than forward differencing.

Example A polynomial parametric curve $\mathbf{P}(t)$ of degree three satisfies the following conditions:

$$t_0 = 0; \quad \mathbf{P}_0 = (0,0); \quad t_1 = 1, \quad \mathbf{P}_1 = (3,3,); \quad t_2 = 3, \quad \mathbf{P}_2 = (5,3); \quad t_3 = 4, \quad \mathbf{P}_3 = (7,0).$$

Compute $\mathbf{P}(2)$ directly from the divided difference table.

Solution
Since this is a degree-three polynomial, the third column in the divided difference table is constant. We add another row to the divided difference table, place 2 (the desired parameter value) in column 1, copy the value of the third column, then solve for c_0, then c_1, then c_2.

$$\begin{array}{c|cccc} 0 & (0,0) & & & \\ 1 & (3,3) & (3,3) & & \\ 3 & (5,3) & (1,0) & (-\frac{2}{3},-1) & \\ 4 & (7,0) & (2,-3) & (\frac{1}{3},-1) & (\frac{1}{4},0) \\ 2 & c_2 & c_1 & c_0 & (\frac{1}{4},0) \end{array}$$

To find c_0, we set up the divided difference equation

$$\frac{(\frac{1}{3},-1)-c_0}{1-2}=(\frac{1}{4},0)$$

from which $c_0=(\frac{7}{12},-1)$. To find c_1, we set up the divided difference equation

$$\frac{(2,-3)-c_1}{3-2}=c_0$$

from which $c_1=(\frac{17}{12},-2)$. To find c_2, we set up the divided difference equation

$$\frac{(7,0)-c_2}{4-2}=c_1$$

from which $c_2=(\frac{25}{6},4)$.

10.4 Neville's Scheme

Neville's scheme addresses the problem: Given a set of $n+1$ points and a parameter value assigned to each point

$$p(t_i)=p_i, \qquad i=0,\ldots,n.$$

find the point $p(t)$ for any other point on the unique degree n polynomial curve that interpolates those points. Note that while Lagrange and Newton polynomials give us an actual curve equation, Neville's scheme provides us with a *geometric construction* for finding any point on the curve, without formally using an equation for the curve.

Neville's scheme is based on Aitken's Lemma. Denote by $p_{i,\ldots k}(t)$ the degree $k-1$ polynomial curve that interpolates points $p_i,\ldots p_k$. Aitken's Lemma states:

$$p_{i,\ldots,k}(t)=\frac{(t_k-t)p_{i,\ldots,k-1}(t)-(t_i-t)p_{i+1,\ldots,k}(t)}{t_k-t_i}.$$

Neville's scheme applies Aitken's Lemma as follows. Suppose we wish to find the point $p(\tau)$. Begin by labeling the interpolated points as $p_0,\ldots p_n$. Then compute the n points

$$p_{i,i+1}(\tau)=\frac{(t_{i+1}-t)p_i-(t_i-t)p_{i+1}}{t_{i+1}-t_i}, \quad i=0,\ldots,n-1.$$

Next compute the $n-1$ points

$$p_{i,i+1,i+2}(\tau)=\frac{(t_{i+2}-t)p_{i,i+1}(\tau)-(t_i-t)p_{i+1,i+2}(\tau)}{t_{i+2}-t_i}, \quad i=0,\ldots,n-2.$$

Then likewise compute the $n-2$ points $p_{i,i+1,i+2,i+3}(\tau)$ and so forth until $p_{i,\ldots,n}(\tau) = p(\tau)$ is computed. A key concept here is that $p_{i,\ldots k}(t)$ **is the curve that interpolates** $\mathbf{P}_i \ldots \mathbf{P}_k$**, and therefore** $p_{i,\ldots k}(\tau)$ **is the point on that curve corresponding to** $t = \tau$**.**

Example A polynomial parametric curve $\mathbf{P}(t)$ of degree three satisfies the following conditions:

$$t_0 = 0; \quad \mathbf{P}_0 = (0,0); \quad t_1 = 1, \quad \mathbf{P}_1 = (3,3,); \quad t_2 = 3, \quad \mathbf{P}_2 = (5,3); \quad t_3 = 4, \quad \mathbf{P}_3 = (7,0).$$

Find $\mathbf{P}(2)$.

We first compute

$$\mathbf{P}_{01}(2) = \frac{(t_1-2)\mathbf{P}_0 + (2-t_0)\mathbf{P}_1}{1-0} = \frac{(1-2)\mathbf{P}_0 + (2-0)\mathbf{P}_1}{1-0} = (6,6);$$

$$\mathbf{P}_{12}(2) = \frac{(t_2-2)\mathbf{P}_1 + (2-t_1)\mathbf{P}_2}{t_2-t_1} = \frac{(3-2)\mathbf{P}_1 + (2-1)\mathbf{P}_2}{2} = (4,3);$$

$$\mathbf{P}_{23}(2) = \frac{(t_3-2)\mathbf{P}_2 + (2-t_2)\mathbf{P}_3}{t_3-t_2} = \frac{(4-2)\mathbf{P}_2 + (2-3)\mathbf{P}_3}{4-3} = (3,6).$$

then

$$\mathbf{P}_{012}(2) = \frac{(t_2-2)\mathbf{P}_{01}(2) + (2-t_0)\mathbf{P}_{12}(2)}{t_2-t_0} = \frac{(3-2)\mathbf{P}_{01}(2) + (2-0)\mathbf{P}_{12}(2)}{3-0} = (\frac{14}{3},4)$$

$$\mathbf{P}_{123}(2) = \frac{(t_3-2)\mathbf{P}_{12}(2) + (2-t_1)\mathbf{P}_{23}(2)}{t_3-t_1} = \frac{(4-2)\mathbf{P}_{12}(2) + (2-1)\mathbf{P}_{23}(2)}{4-1} = (\frac{11}{3},4)$$

and finally

$$\mathbf{P}_{0123}(2) = \frac{(t_3-2)\mathbf{P}_{012}(2) + (2-t_0)\mathbf{P}_{123}(2)}{t_3-t_0} = \frac{(4-2)\mathbf{P}_{012}(2) + (2-0)\mathbf{P}_{123}(2)}{4-0} = (\frac{25}{6},4)$$

10.5 Comparison

We have outlined four different methods for finding a polynomial of degree n that interpolates $n+1$ points. The method of undetermined coefficients requires the solution of $n+1$ equations in $n+1$ unknowns. Lagrange interpolation allows you to write down the equation of the polynomial, but each evaluation requires $O(n^2)$ operations. Newton polynomials evaluation can be performed in $O(n)$ but there is some preprocessing involved in the divided difference table. Evaluation using Neville's scheme is $O(n^2)$. Thus, if you need to evaluate a lot of points, Newton polynomials are the preferred method. They are also numerically stable, assuming the points you are evaluating are within the min and max t values of the sample points.

10.6 Error Bounds

We can use degree n Lagrange polynomials to approximate any function by interpolating $n+1$ points $f(x_0),\ f(x_1),\ \ldots\ f(x_n)$ on the function. In the following equation, $f(x)$ is the function we wish to approximate and p(x) is the Lagrange approximation.

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n) \tag{10.1}$$

The value $f^{(n+1)}(\xi)$ is the $(n+1)^{th}$ derivative of $f(x)$ evaluated at some value ξ where $\min(x, x_0, \ldots, x_n) \leq \xi \leq \max(x, x_0, \ldots, x_n)$. If we can determine a bound on this derivative, then equation 10.1 can be used to provide an error bound on our approximation.

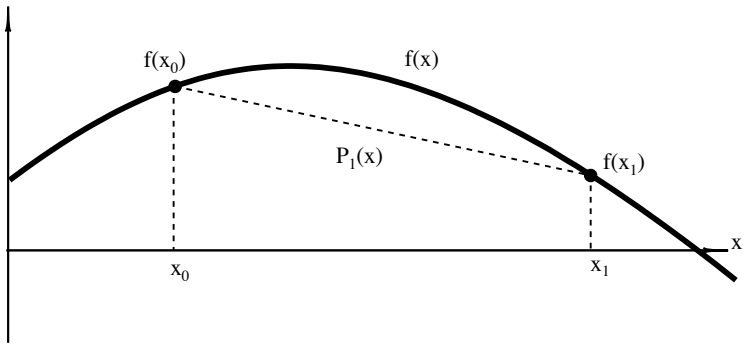


Figure 10.3: Error Bounds

A particularly useful case for equation 10.1 is $n = 1$, or linear approximation. In this case, we can compute the maximum distance that a polynomial deviates from a straight line. If the straight line is

$$p_1(x) = \frac{f(x_0)(x_1 - x) + f(x_1)(x - x_0)}{x_1 - x_0}$$

then

$$|f(x) - p_1(x)| \leq \frac{\max_{(\min(x,x_0,x_1) \leq x \leq \max(x,x_0,x_1))}(f''(x))}{2}(x - x_0)(x - x_1).$$

Let $\delta = x_1 - x_0$ and $L = \max_{(\min(x,x_0,x_1) \leq x \leq \max(x,x_0,x_1)}(f''(x))$. Since the expression $(x-x_0)(x-x_1)$ has a maximum value at $x = \frac{x_0+x_1}{2}$ of $\frac{(x_1-x_0)^2}{4} = \frac{\delta^2}{4}$,

$$|f(x) - p_1(x)| \leq L\frac{\delta^2}{8}.$$

We can assure that the approximation error will be less than a specified tolerance ϵ by using m line segments whose endpoints are evenly spaced in x, where

$$m \geq \sqrt{\frac{L}{8\epsilon}}(x_1 - x_0).$$

A useful application of this idea is to determine how many line segments are needed for plotting a Bézier curve so that the maximum distance between the curve and the set of line segments is less than ϵ. Figure 10.4 shows a cubic Bézier curve approximated with various numbers of line segments. In this case, we simply get a bound on the error in x coordinates and y coordinates independently, and apply the Pythagorean theorem. Bounds (L_x, L_y) on the second derivatives of Bézier curves are easily obtained by computing the second hodograph:

$$L_x = n(n-1) \max_{0 \leq i \leq n-2} |x_{i+2} - 2x_{i+1} + x_i| \tag{10.2}$$

and

$$L_y = n(n-1) \max_{0 \leq i \leq n-2} |y_{i+2} - 2y_{i+1} + y_i|, \tag{10.3}$$

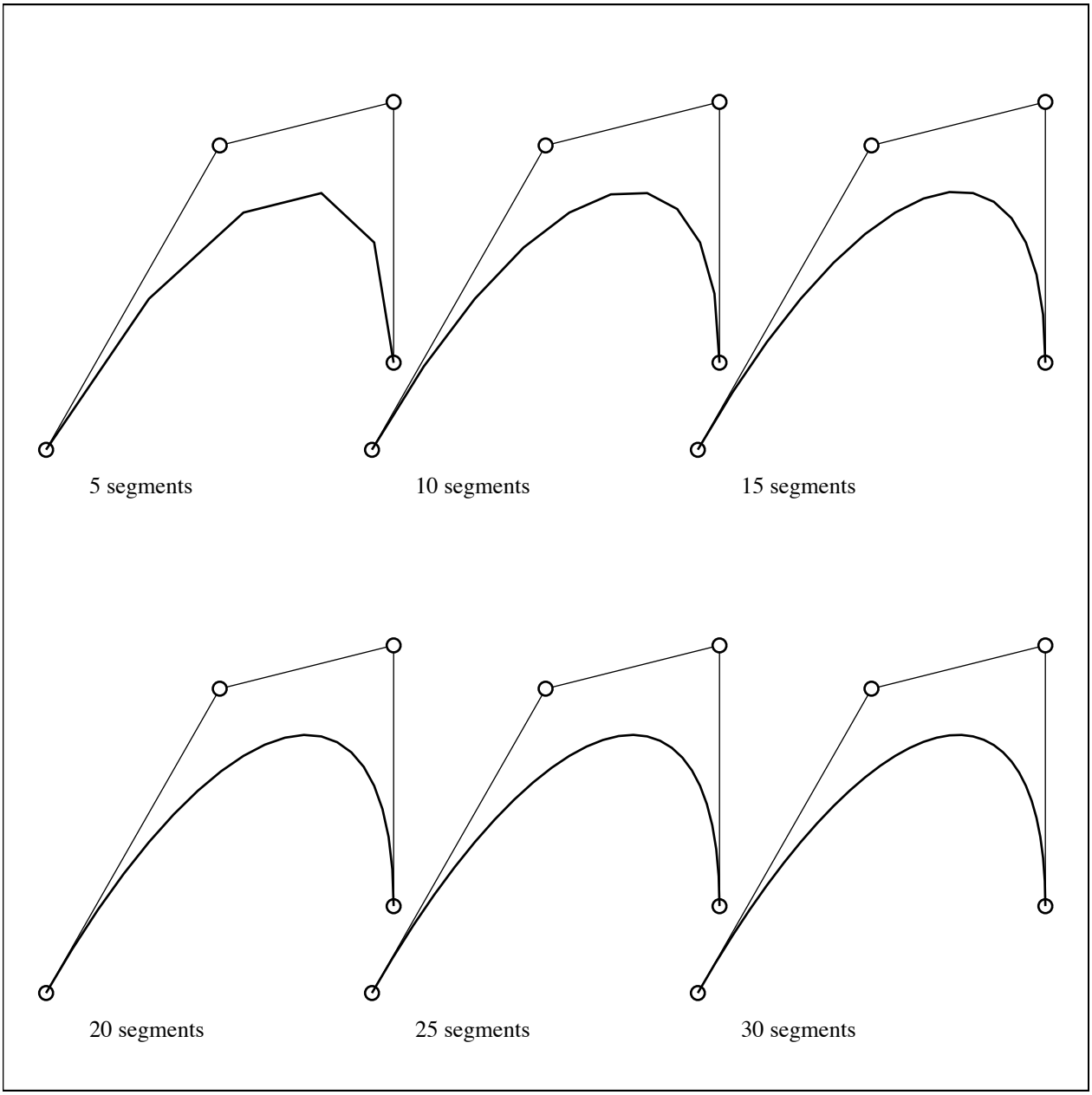


Figure 10.4: Piecewise linear approximation of a Bézier curve

Now, if we use m line segments for approximating the curve where

$$m \geq \sqrt{\frac{\sqrt{L_x^2 + L_y^2}}{8\epsilon}},$$

the maximum error will be less than ϵ.

We can likewise determine how many times r a Bézier curve must be recursively bisected until each curve segment is assured to deviate from a straight line segment by no more than ϵ [GZ84]:

$$r = \log_2 \sqrt{\frac{\sqrt{L_x^2 + L_y^2}}{8\epsilon}} \tag{10.4}$$

This form of the approximation error equation is useful for applications such as computing the intersection between two curves.

10.6.1 Chebyshev Polynomials

If we wish to use a Lagrange polynomial of degree greater than one to perform approximation, we see from equation 10.1 that a wise choice of the interpolation points x_i can improve the approximation error. From equation 10.1,

$$|f(x) - p(x)| \leq \max_{a \leq x \leq b} \frac{f^{(n+1)}(x)}{(n+1)!} \max_{a \leq x \leq b} |(x - x_0)(x - x_1) \cdots (x - x_n)| \tag{10.5}$$

where $x, x_0, \ldots, x_n \in [a, b]$. If we want to decrease the approximation error, we have no control over $\max_{a \leq x \leq b} \frac{f^{(n+1)}(x)}{(n+1)!}$. However, there is an optimal choice for the x_i which will minimize $\max_{a \leq x \leq b} |(x - x_0)(x - x_1) \cdots (x - x_n)|$.

One might make an initial guess that the best choice for the x_i to minimize $\max_{a \leq x \leq b} |(x - x_0)(x - x_1) \cdots (x - x_n)|$ would be simply to space them evenly between x_0 and x_n. This actually gives pretty good results. For the interval $[a, b]$ with $x_i = a + (b - a_i)/n$, it is observed that at least for $4 \leq n \leq 20$, $\max_{a \leq x \leq b} |(x - x_0)(x - x_1) \cdots (x - x_n)| \approx \left(\frac{b-a}{3}\right)^n$.

The *best* choice for the x_i is the Chebyshev points:

$$x_i = \frac{a + b}{2} + \frac{b - a}{2} \cos \frac{2i + 1}{2n + 2} \pi, \quad i = 0, \ldots, n - 1. \tag{10.6}$$

In this case, $\max_{a \leq x \leq b} |(x - x_0)(x - x_1) \cdots (x - x_n)| \equiv 2 \left(\frac{b-a}{4}\right)^n$.

Figure 10.5 shows the Chebyshev and uniform spacing for degree nine polynomials.

10.7 Interpolating Points and Normals

It is possible to use the method of undetermined coefficients to specify tangent vectors as well as interpolating points. For example, consider the cubic parametric curve

$$\mathbf{P}(t) \;=\; \mathbf{a}_0 \;+\; \mathbf{a}_1 t \;+\; \mathbf{a}_2 t^2 \;+\; \mathbf{a}_3 t^3.$$

The derivative of the curve is

$$\mathbf{P}'(t) \;=\; \mathbf{a}_1 \;+\; 2\mathbf{a}_2 t \;+\; 3\mathbf{a}_3 t^2$$

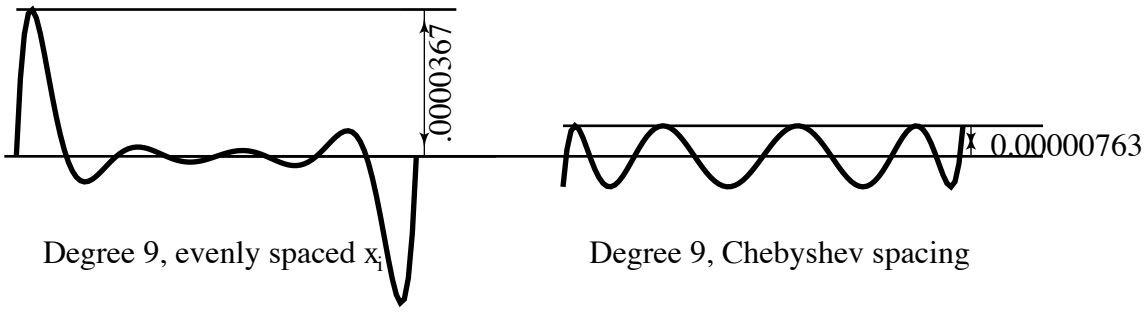


Figure 10.5: Two cases of $(x - x_0)(x - x_1) \cdots (x - x_9)$ for $0 \leq x \leq 1$.

It should be obvious that we can now specify, for example, $\mathbf{P}(t_1)$, $\mathbf{P}(t_2)$, $\mathbf{P}'(t_3)$ and $\mathbf{P}'(t_4)$. Alternately, we can specify three interpolating points and one slope, or three slopes and one interpolating point, etc.

An important case of specifying points and tangent vectors is the Hermite blending functions. In this case, the curve is determined by specifying $\mathbf{P}(0)$, $\mathbf{P}(1)$, $\mathbf{P}'(0)$, and $\mathbf{P}'(1)$ - that is, the two end points and the two end tangent vectors. Solving this case using the method of undetermined coefficients, we obtain the cubic Hermite curve:

$$\mathbf{P}(t) = \mathbf{P}(0)(2t^3 - 3t^2 + 1) + \mathbf{P}(1)(-2t^3 + 3t^2) + \mathbf{P}'(0)(t^3 - 2t^2 + t) + \mathbf{P}'(1)(t^3 - t^2)$$

Chapter 11

Approximation

Xin Li, from the University of Science and Technology of China, contributed significantly to this chapter.

11.1 Introduction

Approximation theory plays an important role in computer aided geometry design and computer graphics. A common general problem in approximation theory can be stated as follows. Let $\mathbf{G}$ denote a set of given data. Let $\mathbf{A}$ denote a set (usually infinite) of candidate mathematical objects (such as functions, parametric surfaces, T-Splines, etc.) with which to approximate $\mathbf{G}$. Let $e(A_i, \mathbf{G})$ be an error measure that indicates how well $A_i \in \mathbf{A}$ approximates $\mathbf{G}$. We wish to find a "best" approximation $A_i \in \mathbf{A}$ for which $e(A_i, \mathbf{G}) \leq e(A_j, \mathbf{G}) \forall A_j \in \mathbf{A}$. Since finding a best approximation can involve expensive optimization algorithms, we might be content to find any approximation for which $e(A_i, \mathbf{G})$ is less than a threshold error. A few simple examples will illustrate.

Example 1. Let $\mathbf{G} = \{2, 3, 3, 5, 7\}$ and let $\mathbf{A}$ be the set of all real numbers. If we choose the error measure to be

$$e(A_i, \mathbf{G}) = \sum_{G \in \mathbf{G}} ((A_i - G)^2$$

the best approximation is $A_i = 4$ for which $e(4, \mathbf{G}) = 16$. This error measure is referred to as the L^2 error measure, and the best approximation is the mean of the values in $\mathbf{G}$.

Example 2. Again, let $\mathbf{G} = \{2, 3, 3, 5, 7\}$ and let $\mathbf{A}$ be the set of all real numbers. Let the error measure be

$$e(A_i, \mathbf{G}) = \max_{G \in \mathbf{G}} |A_i - G|.$$

The best approximation is now $A_i = 4.5$, for which $e(4, \mathbf{G}) = 2.5$. This error measure is referred to as the L^∞ error measure, and the best approximation is the average of the largest and smallest values in $\mathbf{G}$. Thus, different error functions can produce different best approximations.

Example 3. Let $\mathbf{G}$ be a Bézier curve of degree n (call it $\mathbf{P}(t)$) and let $\mathbf{A}$ be the set of all Bézier curves of degree $n - 1$, and $A(t)$ a Bézier curve in $\mathbf{A}$. Let the error measure be

$$\max ||\mathbf{P}(t) - A(t)||, t \in [0, 1].$$

This is the "degree reduction" problem.

Example 4. Let **G** be a B-Spline curve (call it $\mathbf{P}(t)$) of degree n, knot vector k, and domain $t \in [t_0, t_1]$. Let **A** be the set of all B-Spline curves of degree n and with a knot vector $\tilde{k}$ which is obtained by removing one or more knots from k. Let $A(t)$ be a curve in **A**. Let the error measure be

$$\max ||\mathbf{P}(t) - A(t)||, t \in [0, 1].$$

This is the "knot removal" problem.

Example 5. **G** is a large set of data points, measured on the surface of a physical object. **A** is a set of NURBS, T-Splines, or Subdivision Surfaces. This is the "reverse engineering," for which it is customary to use the L^2 error measure.

Example 6. In drawing a curve or surface, it is customary to approximate the curve with a polygon, or the surface with a polyhedron. These are examples of approximating a continuous object with a simpler continuous object. The question of how many line segment or triangles are needed to make the curve or surface appear smooth belongs to approximation theory.

11.2 L^2 Error

The L^2 error measure is widely used in approximation problems because in many cases it is particularly easy to identify A_i that minimizes the L^2 error measure. Approximation using the L^2 error measure is also called "least squares" approximation. In the case where **G** is a set of real numbers, the best L^2 approximation is simply the mean of those numbers. Specifically, let

$$f(x) = \sum_{i=1}^{n} (x - x_i)^2.$$

Then we have the following lemma.

Lemma 11.2.1 *Suppose* $\hat{x} = \frac{\sum_{i=1}^{n} x_i}{n}$, *then*

$$f(x) \geq f(\hat{x}).$$

Proof 1

$$\begin{aligned} f(x) - f(\hat{x}) &= \sum_{i=1}^{n} [(x - x_i)^2 - (\hat{x} - x_i)^2] \\ &= \sum_{i=1}^{n} [(x - \hat{x})(x + \hat{x} - 2x_i)] \\ &= (x - \hat{x})(nx - \sum_{i=1}^{n} x_i) \\ &= n(x - \hat{x})^2 \geq 0 \end{aligned}$$

So the lemma holds.

11.3 Approximating a Set of Discrete Points with a B-Spline Curve

Let **G** be a set of data points $P_i = (x_i, y_i, z_i)$, $i = 1, \ldots, M$, with associated parameter values, t_i. Let **A** be the set of all B-Spline curves with a given knot vector and degree. We seek $A_i \in \mathbf{A}$ that

minimizes the discrete L^2 error where $A_i \in \mathbf{A}$ is given by $B(t) = \sum_{j=1}^{n} T_j B_j(t)$. This amounts to identifying the B-Spline control points $T_1, T_2, \ldots, T_n$ that minimize the function

$$Lsq(T_1, T_2, \ldots, T_n) = \sum_{i=1}^{M} (P_i - \sum_{j=1}^{n} T_j B_j(t_i))^2. \tag{11.1}$$

This problem reduces to the solution of the following linear functions:

$$\frac{\partial Lsq(T_1, T_2, \ldots, T_n)}{\partial T_j} = 0. \tag{11.2}$$

We can solve for the control points T_i from the linear equation

$$M_{fit} * T = B \tag{11.3}$$

where M_{fit} is a $n \times n$ matrix with element $a_{ij} = \sum_{k=1}^{M} B_i(s_k) B_j(s_k)$. $T = [T_i]$ is the vector of control points for which we are solving and B is a vector whose elements are $b_i = \sum_{k=1}^{M} P_k B_i(t_k)$.

Algorithm 1 Setting Up the fitting matrix M_{fit}

Require: sample points P_i, associated parameter values (t_i), knot vector and degree for the B-Spline curve.
Ensure: $M_{fit} = [a_{ij}]$
  for $i = 1$ to N **do**
    for $j = i$ to N **do**
      $a_{ij} \Leftarrow a_{ji} \Leftarrow 0$
      $\Omega_{ij} = Support(B_i) \cap Support(B_j)$
      if $\Omega_{ij} \neq \varnothing$ **then**
        for $k = 1$ to M **do**
          if $(s_k) \in \Omega_{ij}$ **then**
            $d \Leftarrow B_i(s_k) * B_j(s_k)$
            $a_{ij} += d$
            $a_{ji} += d$
          end if
        end for
      end if
    end for
  end for

The prudent choice of s_i for each sample point P_i, which determines the parametrization of the curve, is crucial in obtaining a good fit. The choice of knot vector or knot intervals can also impact the quality of the fit. The next two subsections elaborate on these topics.

11.3.1 Parametrization

The simplest parametrization assigns evenly spaced parameter values to the data points:

$$t_j = \frac{j-1}{M-1}. \tag{11.4}$$

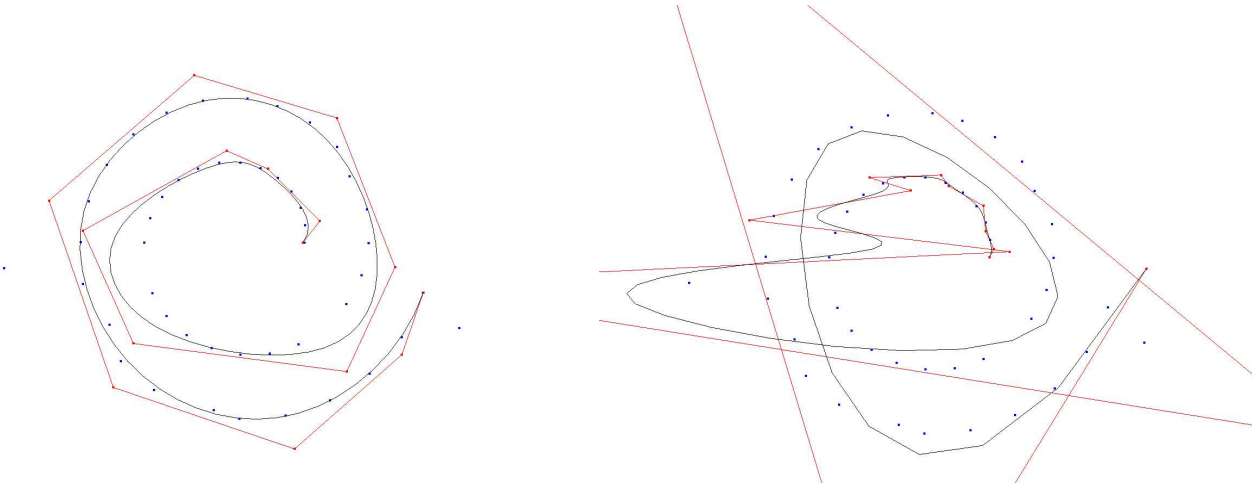


Figure 11.1: Uniform vs. bad parametrization.

This *uniform* parametrization works well if the data points are evenly spaced, as illustrated in Figurefig:1.

The second is arc length parametrization, which is defined as

$$s_j = \frac{\sum_{i=2}^{j} \|P_i - P_{i-1}\|}{\sum_{i=2}^{M} \|P_i - P_{i-1}\|}. \tag{11.5}$$

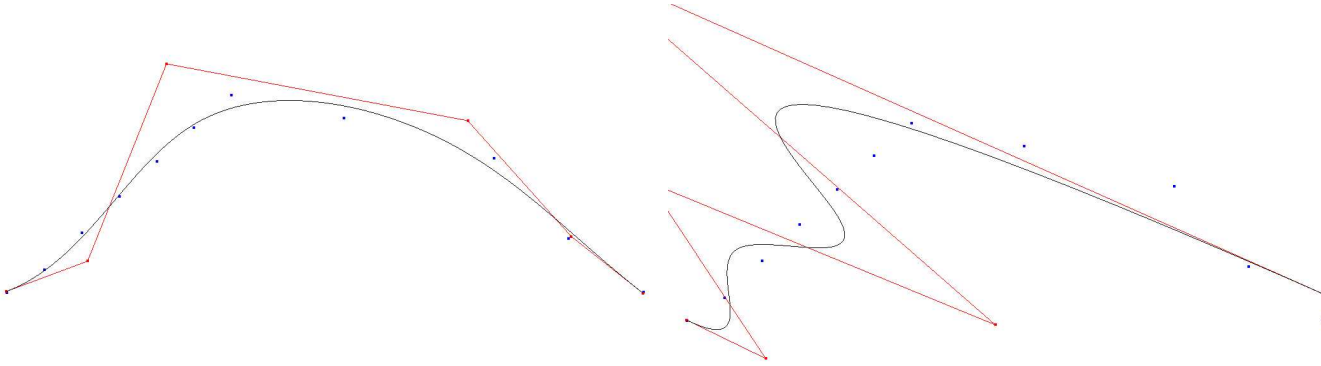


Figure 11.2: Arc length vs. bad parametrization.

The uniform parametrization only suits for regular sample points. The arc length parametrization is much more popular. Here we will give some examples to show the result of least square fitting with different parametrization. In the following examples, the red line is the control polygon and the blue points are the sample points. Figure 11.2,11.1 are two examples of using different parametrization.

11.3.2 Knot vector

Knot vectors are also very important for the curve or surface fitting. As we known, the degree three B-spline function which associated with knot vector $s = [s_0, s_1, s_2, s_3, s_4]$ is zero out of the interval $[s_0, s_4]$, which is called the support of the basis function. The knots and the parameters are the keys for the structure of the matrix M_{fit}. So if the knots around some parameter are very sparse but very dense around some others, then the curve won't be very good, to see 11.3 as an example. The second picture is fitting with a very bas knots which are very dense among the first six knots.

In general, the knots are selected as the subset of the parametrization. Given a integer m, then then knots are s_{j*m} respectively. Bad knots also will lead to very bad spline curve. Figure 11.3 gives some examples. The first picture is fitting with uniform knots.

11.4 Fairing

In order to get smooth curve or surface to fit the given samples points very well, we often add some energy to the equation 11.3. For curve, the item often used is$\int f_{ss}^2$. So the problem is changed to

$$\min \sum_{i=1}^{M} (P_i - \sum_{j=1}^{n} T_j B_j(s_i))^2 + c \int (\sum_{j=1}^{n} T_j B_j(s))^2) \tag{11.6}$$

Here the constant c is defined by user. But this constant is very important for the result curve and it is very difficult to choose the constant. Similarly, the problem also equates the solution of the following linear functions:

$$(M_{fit} + cM_{fair}) * T = B; \tag{11.7}$$

Here the algorithm for computing the fairing matrix is listed as follows:

Algorithm 2 Compute the fairing matrix M_{fair}

Require: all the blending functions B_i
Ensure: $M_{fair} = [b_{ij}]$
  for $i = 1$ to N **do**
    for $j = i$ to$N$ **do**
      $a_{ij} \Leftarrow a_{ji} \Leftarrow d \Leftarrow 0$
      $\Omega_{ij} = Support(B_{iss}) \cap Support(B_{jss}) = [a, b]$
      if $\Omega_{ij} \neq \varnothing$ **then**
        $k \Leftarrow 10, l \Leftarrow 10$
        for $i1 = 1$ to $2k - 1$, $i1+ = 2$ **do**
          $d+ = B_{iss}(a + \frac{i1(b-a)}{2k}) B_{jss}(a + \frac{i1(b-a)}{2k})$
        end for
      end if
      $d\times = (b - a)/(k)$
      $a_{ij}+ = d$
      $a_{ji}+ = d$
    end for
  end for

Figure 11.4 illustrated two pictures which are least square fitting and after fairing. We can see that after fairing, the curve becomes much better.

But as we just said, it is very difficult to choose the constant. A larger constant will leads the curve to be shrank, to see Figure 11.5 as an examples.

We must mention here that it is very difficult to smooth a curve with bad parameters. Figure 11.6 gives the example. With a bigger constant, the curve will be shrank as the picture in Figure 11.5.

11.4.1 Interpolation

The interpolation means that we specify some points that the result curve must pass through. Suppose the interpolation points are $Q_i, i = 1, \ldots, I$, and the associated parameter are u_i respectively. Then the problem turns to

$$\min \sum_{i=1}^{M} (P_i - \sum_{j=1}^{n} T_j B_j(s_i))^2 + c \int (\sum_{j=1}^{n} T_j B_j(s))^2)$$

$$\text{subject to: } \sum_{j=1}^{n} T_j B_j(u_i) = Q_i, i = 1, \ldots, I.$$

Using **Lagrange Multiplier method**, we also can change the problem to

$$\min \sum_{i=1}^{M} (P_i - \sum_{j=1}^{n} T_j B_j(s_i))^2 + c \int (\sum_{j=1}^{n} T_j B_j(s))^2) + \sum_{k=1}^{I} \lambda_k (\sum_{j=1}^{n} T_j B_j(u_k) - Q_k)$$

The derivations of the function for all the T_j and λ_k are all zero. all the linear functions can be written as a matrix form:

$$\begin{pmatrix} M_{fit} + cM_{fair} & | & A \\ - - - - - - - - & + & - - \\ A^T & | & 0 \end{pmatrix} [T_1, \ldots, T_n, \lambda_1, \ldots, \lambda_I]^T = [B|C]^T$$

Here A is a matrix with n rows and I columns which element of $i-th$ rows and $j-th$ columns is

$$a_{ij} = B_i(u_j), 1 \leq i \leq n, 1 \leq j \leq I.$$

And the element of C is

$$R_j = Q_j, 0 < i \leq I.$$

Solve the linear functions, we can compute all the control points for the curve. But the result curve is also associated with the constant. The curve is much less sensitive than that of without interpolation. Figure 11.7 shows an example of interpolation with different constants. In the next section, we will give the method for eliminating the constant.

11.4.2 Constrained fairing

In this section, we will discuss a new idea for constructing a fairing curve or surface.

Denote

$$L^2_\varepsilon = \{f | f = \sum_{i=1}^{n} T_i B_i(s), Lsq(T_1, T_2, \ldots, T_n) < \varepsilon\}. \tag{11.8}$$

If ε is less than the least square error, then the set is null. The idea for fairing is that select the fairest curve from L^2_ε according to a given error ε. The problem is a convex problem. But it is very difficult to solve. So we want to reduce the condition to turn the problem to a linear condition.

Define a bound box BX_i for each sample point P_i, and we restrict real point belongs to the bound box. That is to say, we want to find the fairest curve from the space:

$$L = \{f | f(s) = \sum_{i=1}^{n} T_i B_i(s), f(u_i \in BX_i)\}. \tag{11.9}$$

Here we will define two kinds of bounding box, the first is very popular, which parallel to the coordinates.

$$BX_i = \{(x, y) | sx_i \le x \le lx_i, sy_i \le y \le ly_i\} \tag{11.10}$$

Then the problem can be divided into two or three independent problems such like that:

$$\begin{gathered} \min X^T H X \\ \text{subject to: } AX \le B \end{gathered} \tag{11.11}$$

Here

$$\begin{gathered} X = [x_1, x_2, \ldots, x_M]; \\ H = [h_{ij}]_{n\times n}, h_{ij} = \int B_i(s) B_j(s); \\ A = [a_{ij}]_{2M\times n}, a_{ij} = a_{(i+M)j} = B_j(u_i); \\ B = [b_i]_{2M}, b_i = lx_i, b_{i+M} = -sx_i. \end{gathered}$$

Here X is a vector of all the control points. H is called the Hessian matrix for the object function and A is the constrain matrix.

The results are listed in Figure 11.8. All the parameters are same as those of in Figure 11.7. If we set very big bound boxes, then we compute a line.

The second boxes are associated with the neighbors of the sample points. We estimate a tangent vector t_i and a normal vector n_i for each point, and the bound box for the point is

$$BX_i = \{P_i + s * t_i + t * n_i | -0.5 \le s \le 0.5, -0.5 \le t \le 0.5, \} \tag{11.12}$$

Then the problem become to a similar problem as 11.11.

$$\begin{gathered} X = [x_1, x_2, \ldots, x_M, y_1, y_2, \ldots, y_M]; \\ H = [h_{ij}]_{2n\times 2n}, h_{ij} = h_{(i+n)(j+n)} \int B_i(s) B_j(s); \\ A = [a_{ij}]_{4M\times 2n}, (a_{ij}, a_{i(j+n)}) = -(a_{(i+M)j}, a_{(i+M)(j+n)}) = B_j(u_i) t_i, \\ (a_{(i+2M)j}, a_{(i+2M)(j+n)}) = -(a_{(i+3M)j}, a_{(i+3M)(j+n)}) = B_j(u_i) n_i; \\ B = [b_i]_{4M}, b_i = P_i \cdot t_i + 0.5 t_i \cdot t_i, b_{i+M} = -P_i \cdot t_i + 0.5 t_i \cdot t_i, \\ b_{i+2M} = P_i \cdot n_i + 0.5 n_i \cdot n_i, b_{i+3M} = -P_i \cdot n_i + 0.5 n_i \cdot n_i, \end{gathered}$$

Solve the quadratic problem, we can get the solution. We can see that it is much better than both least square and fairing.

Here the tangent t_i and normal n_i for each point P_i are estimated with a very simple method. Let $P_0 = P_1$ and $P_{M+1} = P_M$, then the direction of t_i is the same as $P_{i+1} - P_{i-1}$. n_i is the perpendicularity direction of t_i.

11.4.3 Images

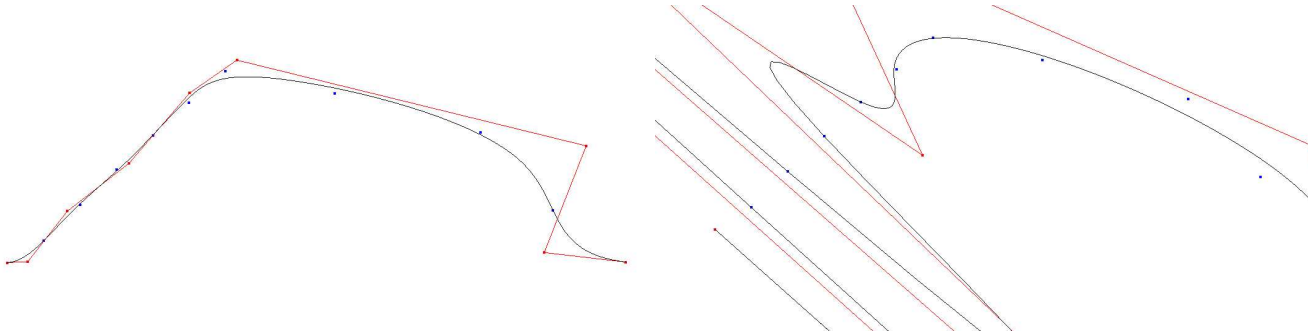


Figure 11.3: Uniform vs. bad knots.

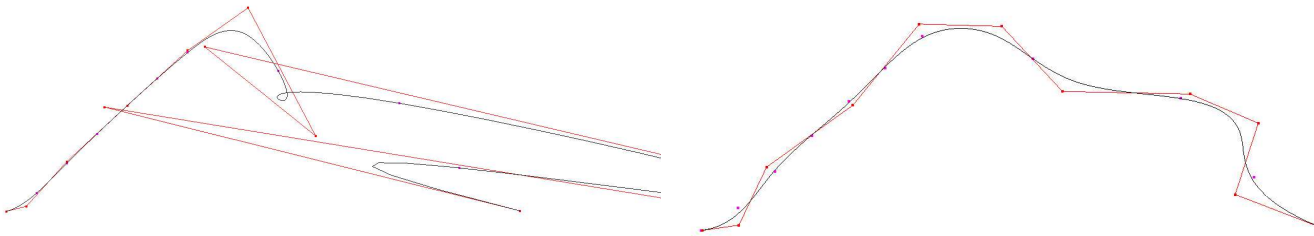


Figure 11.4: The fairing effect.

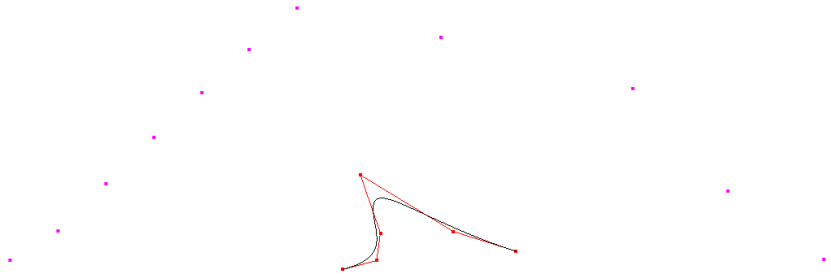


Figure 11.5: The shrank curve.

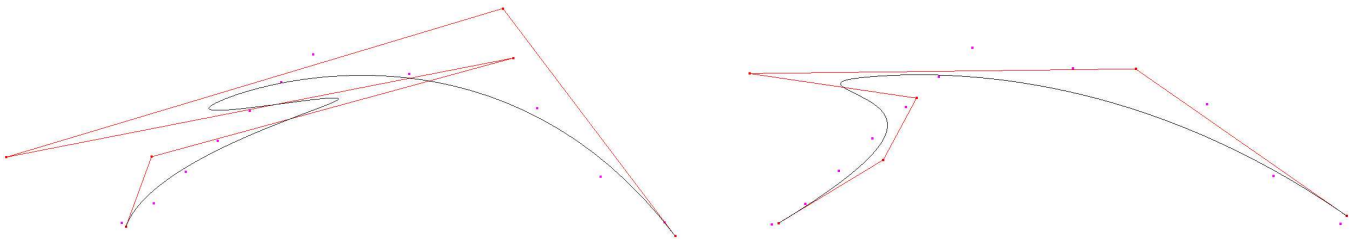


Figure 11.6: The fairing effect of bad parameter.

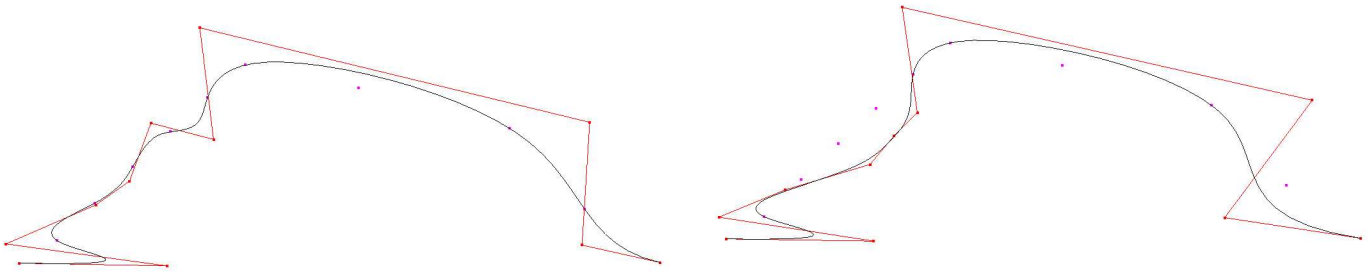


Figure 11.7: Fairing and interpolation with different constant.

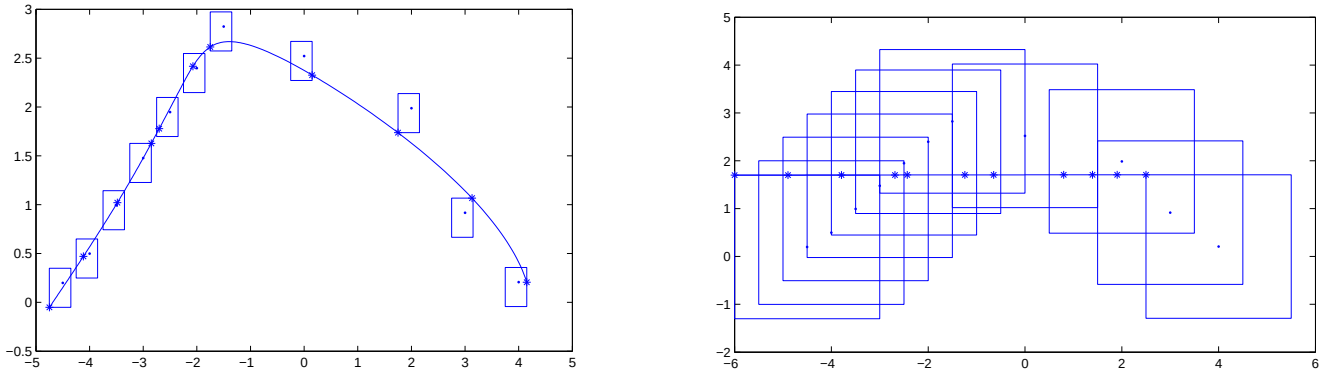
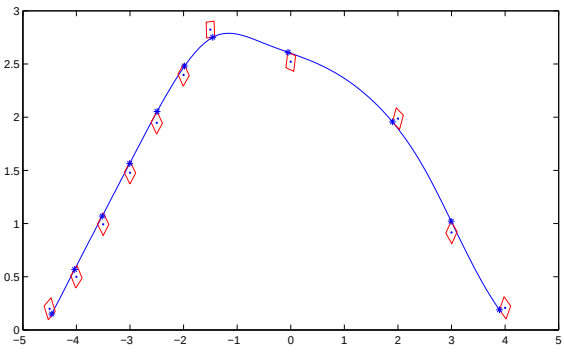


Figure 11.8: Constrained fairing.



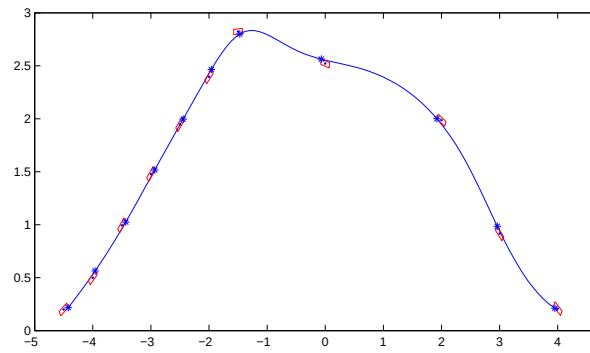


Figure 11.9: Constrained fairing.

Chapter 12

Interval Bézier Curves

In recent years there has been considerable interest in approximating the curves and surfaces that arise in computer–aided design applications by other curves and surfaces that are of lower degree, of simpler functional form, or require less data for their specification (see, for example, [Hos87], [Hos88], [Lac88], [LM87], [Pat89], [SK91], [SWZ89], [WW88]). The motivation for this activity arises from the practical need to communicate product data between diverse CAD/CAM systems that impose fundamentally incompatible constraints on their canonical representation schemes, e.g., restricting themselves to polynomial (rather than rational) forms, or limiting the polynomial degrees that they accommodate.

While most of the approximation schemes in the references cited above attempt at minimum to guarantee that an approximation will satisfy a prescribed tolerance, once this has been achieved none of them proposes to carry detailed information on the approximation error forward to subsequent applications in other systems. Such information can be of crucial importance, for example, in tolerance analyses of the components of a mechanism. This chapter describes a form — the interval Bézier curve — that is capable of carrying such information, and we show how a complete description of approximation errors may be readily embodied in such forms in a straightforward and natural manner.

12.1 Interval arithmetic and interval polynomials

By a scalar interval $[\,a, b\,]$ we mean a closed set of real values of the form

$$[\,a, b\,] = \{\, t \mid a \le t \le b \,\}\,. \tag{12.1}$$

Given two such intervals $[\,a, b\,]$ and $[\,c, d\,]$, the result of a binary arithmetic operation $\star \in \{\,+\,,\,-\,,\,\cdot\,,\,/\,\}$ on them is defined to be the set of all values obtained by applying $\star$ to operands drawn from each interval:

$$[\,a, b\,] \star [\,c, d\,] = \{\, x \star y \mid x \in [\,a, b\,] \text{ and } y \in [\,c, d\,] \,\}\,. \tag{12.2}$$

Specifically, it is not difficult to verify that

$$\begin{aligned}
[\,a, b\,] + [\,c, d\,] &= [\,a + c, b + d\,]\,, \\
[\,a, b\,] - [\,c, d\,] &= [\,a - d, b - c\,]\,, \\
[\,a, b\,] \cdot [\,c, d\,] &= [\,\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)\,]\,, \\
[\,a, b\,] \,/\, [\,c, d\,] &= [\,a, b\,] \cdot [\,1/d, 1/c\,]\,,
\end{aligned} \tag{12.3}$$

where division is usually defined only for denominator intervals that do not contain 0. Occasionally, we shall wish to treat a single real value as a *degenerate* interval,

$$a = [\, a, a \,] \tag{12.4}$$

so that we can apply the rules (12.3) to mixed operands (e.g., $a + [\, b, c \,] = [\, a + b, a + c \,]$). We shall also make use of a convenient short–hand notation for intervals, denoting them by a single symbol enclosed in square parentheses, e.g., $[\, u \,] = [\, a, b \,]$ and $[\, v \,] = [\, c, d \,]$. (However, the reader is warned against allowing this notation to impart an undue sense of familiarity — note, for example, that $[\, u \,] - [\, u \,] \neq 0$ in general!)

It can be verified from (12.3) that interval addition and multiplication are commutative and associative, but that multiplication does not, in general, distribute over addition. A notable exception is the multiplication of a sum of intervals by a scalar value, for which

$$\alpha\,([\, u \,] + [\, v \,]) = \alpha [\, u \,] + \alpha [\, v \,] \tag{12.5}$$

holds for arbitrary real values α and intervals $[\, u \,]$, $[\, v \,]$. An example of the converse case — the sum of scalar multiples of a given interval — is given in equation (12.24) below. For a more rigorous and detailed discussion of these matters, see [Moo66], [Moo79].

Interval arithmetic offers an essentially infallible (although often pessimistic) means of monitoring error propagation in numerical algorithms that employ floating point arithmetic. Many familiar algorithms can be re–formulated to accept interval operands (e.g., [Han78b], [Han78a]). The use of interval techniques in the context of geometric modeling calculations has been discussed in [MK84].

An *interval polynomial* is a polynomial whose coefficients are intervals. We shall denote such polynomials in the form $[\, p \,](t)$ to distinguish them from ordinary (single–valued) polynomials. In general we express an interval polynomial of degree n in the form

$$[\, p \,](t) = \sum_{k=0}^{n} [\, a_k, b_k \,]\, B_k^n(t)\,, \tag{12.6}$$

in terms of the Bernstein polynomial basis

$$B_k^n(t) = \binom{n}{k} (1-t)^{n-k} t^k \quad \text{for } k = 0, \ldots, n \tag{12.7}$$

on $[\, 0, 1 \,]$. Using (12.3), the usual rules of polynomial arithmetic can be carried over with minor modifications to the case of interval polynomials (see [Rok75]; polynomial arithmetic in the Bernstein basis is described in [FR88]). Since the basis functions (12.7) are evidently all non–negative for $t \in [\, 0, 1 \,]$, we can also express (12.6) in the form

$$[\, p \,](t) = [\, p_{\min}(t), p_{\max}(t) \,] \quad \text{for all } t \in [\, 0, 1 \,]\,, \tag{12.8}$$

where

$$p_{\min}(t) = \sum_{k=0}^{n} a_k\, B_k^n(t) \quad \text{and} \quad p_{\max}(t) = \sum_{k=0}^{n} b_k\, B_k^n(t)\,. \tag{12.9}$$

A useful concept in dealing with interval polynomials defined over a finite domain as approxima-

tion tools is the *width* of such a polynomial,

$$\begin{aligned} \text{width}(\,[\,p\,](t)\,) &= \int_0^1 p_{\max}(t) - p_{\min}(t)\,dt \\ &= \sum_{k=0}^{n} (b_k - a_k) \int_0^1 B_k^n(t)\,dt \\ &= \frac{1}{n+1} \sum_{k=0}^{n} b_k - a_k\,. \end{aligned} \tag{12.10}$$

The last step follows from the fact that the definite integral of each of the basis functions (12.7) over $[\,0,1\,]$ is just $1/(n+1)$ [FR88].

The width (12.10) measures the total area enclosed between the two polynomials (12.9) that define the upper and lower bounds on $[\,p\,](t)$ over $t \in [\,0,1\,]$. The Bernstein–Bézier formulation (12.6) has the convenient feature that the width of an interval polynomial $[\,p\,](t)$ is simply the *average* of the widths $b_k - a_k$ of its coefficients.

A desirable characteristic of any scheme that calls for approximation by interval polynomials is to provide a means whereby successive approximations of uniformly decreasing width are generated. The approximated function can thus be confined within as small an area as desired.

12.2 Interval Bézier curves

We will define a vector–valued interval $[\,\mathbf{p}\,]$ in the most general terms as any compact set of points (x, y) in two dimensions. The addition of such sets is given by the *Minkowski sum*,

$$[\,\mathbf{p}_1\,] + [\,\mathbf{p}_2\,] = \{\,(x_1 + x_2, y_1 + y_2) \mid (x_1, y_1) \in [\,\mathbf{p}_1\,] \text{ and } (x_2, y_2) \in [\,\mathbf{p}_2\,]\,\}\,. \tag{12.11}$$

Such point–set operations arise, for example, in image analysis [Ser82]. However, since they are in general quite difficult to perform algorithmically (even when $[\,\mathbf{p}_1\,]$ and $[\,\mathbf{p}_2\,]$ are restricted to polygonal boundaries), it will be prudent for the present to restrict our attention to vector–valued intervals that are just the *tensor products* of scalar intervals,

$$[\,\mathbf{p}\,] = [\,a,b\,] \times [\,c,d\,] = \{\,(x,y) \mid x \in [\,a,b\,] \text{ and } y \in [\,c,d\,]\,\}\,. \tag{12.12}$$

We occasionally use the notation $([\,a,b\,],[\,c,d\,])$ instead of $[\,a,b\,] \times [\,c,d\,]$ for $[\,\mathbf{p}\,]$. Such vector–valued intervals are clearly just rectangular regions in the plane, and their addition is a trivial matter:

$$[\,\mathbf{p}_1\,] + [\,\mathbf{p}_2\,] = [\,a_1 + a_2, b_1 + b_2\,] \times [\,c_1 + c_2, d_1 + d_2\,]\,, \tag{12.13}$$

where $[\,\mathbf{p}_1\,] = [\,a_1, b_1\,] \times [\,c_1, d_1\,]$ and $[\,\mathbf{p}_2\,] = [\,a_2, b_2\,] \times [\,c_2, d_2\,]$. The extension of these ideas to vector–valued intervals in spaces of higher dimension is straightforward.

We note that there are other potentially useful shapes for interval control points (e.g., circular disks); the discussion that follows adapts readily to interval control points of more general shape, although specific algorithms may be more difficult to implement than in the tensor–product case.

Let us recall now that a Bézier curve on the parameter interval $[\,0,1\,]$ is defined by

$$\mathbf{P}(t) = \sum_{k=0}^{n} \mathbf{P}_k\, B_k^n(t)\,, \tag{12.14}$$

where the Bernstein basis function $B_k^n(t)$ are as defined above in (12.7). The vector coefficients $\mathbf{P}_k = (x_k, y_k)$ in (12.14) are called the *control points* of the curve.

An interval Bézier curve (i.e., a Bézier curve with *vector–interval control points*) is written in the form

$$[\,\mathbf{P}\,](t) = \sum_{k=0}^{n} [\,\mathbf{P}_k\,]\, B_k^n(t)\,, \tag{12.15}$$

where we assume that the $[\,\mathbf{P}_k\,]$ are rectangular (possibly degenerate) intervals of the form (12.12). Figure 12.1 illustrates a sample cubic interval Bézier curve.

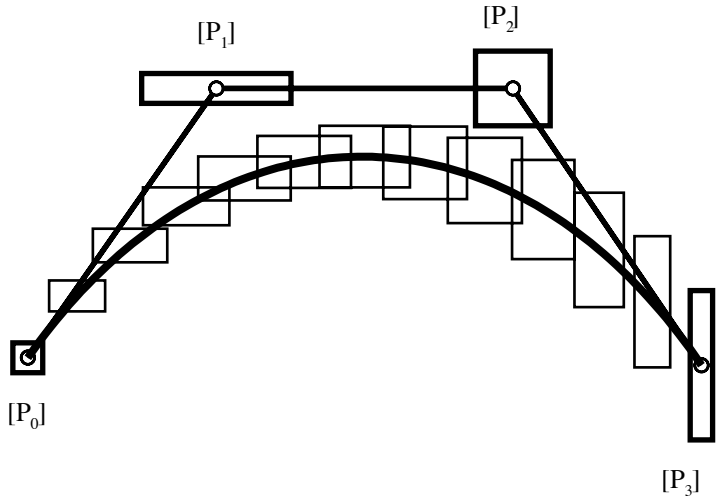


Figure 12.1: A cubic interval Bézier curve.

For each $t \in [\,0,1\,]$, the value $[\,\mathbf{P}\,](t)$ of the interval Bézier curve (12.15) is a vector interval that has the following significance: For any Bézier curve $\mathbf{P}(t)$ whose control points satisfy $\mathbf{P}_k \in [\,\mathbf{P}_k\,]$ for $k = 0, \ldots, n$, we have $\mathbf{P}(t) \in [\,\mathbf{P}\,](t)$. Likewise, the entire interval Bézier curve (12.15) defines a region in the plane that contains all Bézier curves whose control points satisfy $\mathbf{P}_k \in [\,\mathbf{P}_k\,]$ for $k = 0, \ldots, n$.

We shall not concern ourselves here with defining curves of the form (12.15) *ab initio*, but rather with the use of that form in approximating more complicated curves and surfaces by polynomial interpolation in such a manner that *the interval control points* $[\,\mathbf{P}_k\,]$ *reflect the approximation error*. Thus, the plane region defined by an interval approximant $[\,\mathbf{P}\,](t)$ to some other curve $\mathbf{r}(t)$ will be guaranteed to contain the latter.

While an initial approximant of this form may be unacceptably crude — defining an area that is too large for practical purposes — a process of refinement by subdivision will usually remedy this problem. A precise representation of the error incurred by approximations can be crucial in applications such as tolerance analysis, where a nominal approximant is of little value without information on its deviation from the exact form. The Bernstein–Bézier form offers an intuitive framework for expressing approximants, along with their error terms, as interval–valued polynomials.

12.2.1 Affine maps

A key operation in the de Casteljau subdivision and degree elevation algorithms for Bézier curves (see [Far90]) is computing the affine map

$$\mathbf{M}(p_0, p_1, t) = (1-t)\, p_0 + t\, p_1 \tag{12.16}$$

of two points p_0 and p_1 (we consider for now the case where p_0 and p_1 are simply scalar values). This map can be visualized as shown in Figure 12.2, where the values of p_0 and p_1 are taken as the vertical coordinates, and the values of t as horizontal coordinates. At $t = 0$, $y = p_0$ while at $t = 1$, $y = p_1$. The affine map is then represented graphically by drawing a straight line through p_0 and p_1. This line can be regarded as the affine map function for the two points: at any value of t, the affine map is simply the vertical coordinate of the line.

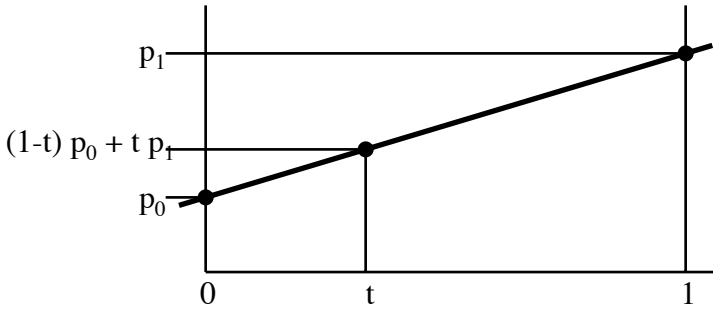


Figure 12.2: The affine map of two scalar points.

Consider next the affine map of two scalar intervals:

$$\begin{aligned} [\,\mathbf{M}\,]([\,a,b\,],[\,c,d\,],t\,) &= (1-t)\,[\,a,b\,] + t\,[\,c,d\,] \\ &= \{\,(1-t)\,u + t\,v \mid u \in [\,a,b\,] \text{ and } v \in [\,c,d\,]\,\}\,. \end{aligned} \tag{12.17}$$

The affine map (12.17) can be visualized as shown in Figure 12.3 — for a given value of t,

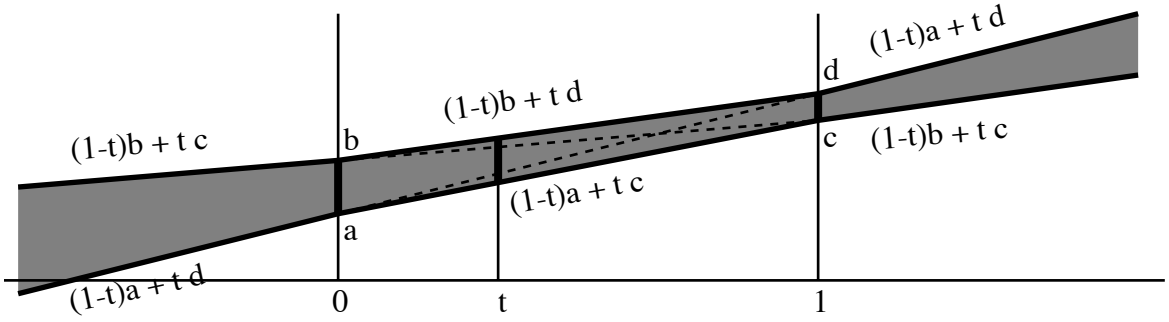


Figure 12.3: The affine map of two scalar intervals.

$[\,\mathbf{M}\,]([\,a,b\,],[\,c,d\,],t\,)$ is generated by drawing a vertical line at that t value and identifying all points on that line which are collinear with any point in $[\,a,b\,]$ at $t=0$ and any point in $[\,c,d\,]$ at $t=1$. Qualitatively, we see that the width of the affine map (12.17) lies between the widths of $[\,a,b\,]$ and $[\,c,d\,]$ when $t\in[\,0,1\,]$, whereas the width of the affine map grows linearly — without bound — for t outside the unit interval.

The affine map of two vector-valued intervals $[\mathbf{P}_0]$ and $[\mathbf{P}_1]$ as defined in equation 12.12 is simply

$$[\,\mathbf{M}\,]([\,\mathbf{P}_0],[\,\mathbf{P}_1],t\,)=\{\,(1-t)\,\mathbf{u}+t\,\mathbf{v}\mid\mathbf{u}\in[\,\mathbf{P}_0\,]\text{ and }\mathbf{v}\in[\,\mathbf{P}_1\,]\,\}\tag{12.18}$$

as shown in figure 12.4. Observe that the midpoint, width, and height of the affine map rectangle

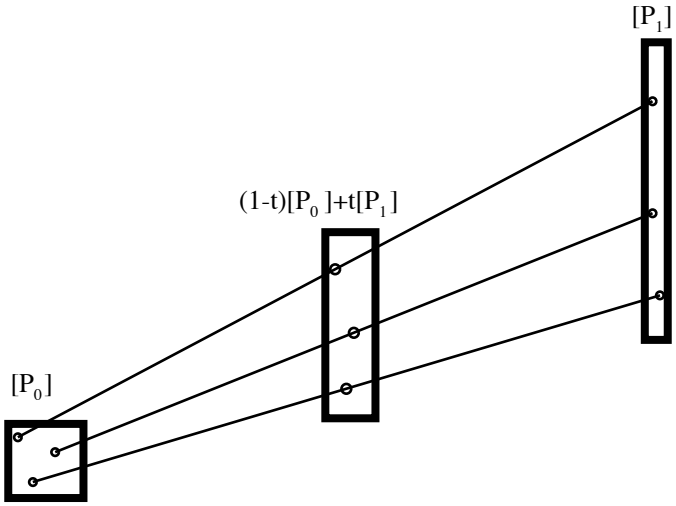


Figure 12.4: The affine map of two vector intervals.

are simply affine maps of the midpoints, widths, and heights of $[\mathbf{P}_0]$ and $[\mathbf{P}_1]$.

For a Bézier curve of the form (12.14), we may regard the de Casteljau algorithm as a repeated application of (12.16) to the control points $\{\mathbf{P}_k\}$. With $\mathbf{P}_k^{(0)}=\mathbf{P}_k$ for $k=0,\ldots,n$, we set

$$\mathbf{P}_k^{(r+1)}\;=\;\mathbf{M}\,(\,\mathbf{P}_{k-1}^{(r)}\,,\,\mathbf{P}_k^{(r)}\,,\,t\,)\quad\text{for }k=r+1,\ldots,n\tag{12.19}$$

and $r=0,\ldots,n-1$. This generates a triangular array of the quantities $\{\mathbf{P}_k^{(r)}\}$ — the final entry $\mathbf{P}_n^{(n)}$ in this array corresponds to the point $\mathbf{P}(t)$ on the curve (12.14), while the entries

$$\mathbf{P}_0^{(0)},\mathbf{P}_1^{(1)},\ldots,\mathbf{P}_n^{(n)}\quad\text{and}\quad\mathbf{P}_n^{(n)},\mathbf{P}_n^{(n-1)},\ldots,\mathbf{P}_n^{(0)}\tag{12.20}$$

on the left– and right–hand sides of the array are the control points for the subsegments of $\mathbf{P}(t)$ over the parameter intervals $[\,0,t\,]$ and $[\,t,1\,]$, respectively. To apply (12.19) to *interval* Bézier curves, we simply replace the quantities $\mathbf{P}_k^{(r)}$ by their interval counterparts, and invoke the appropriate rules of interval arithmetic, as illustrated in Figure 12.5. We discuss the de Casteljau algorithm further in the next section.

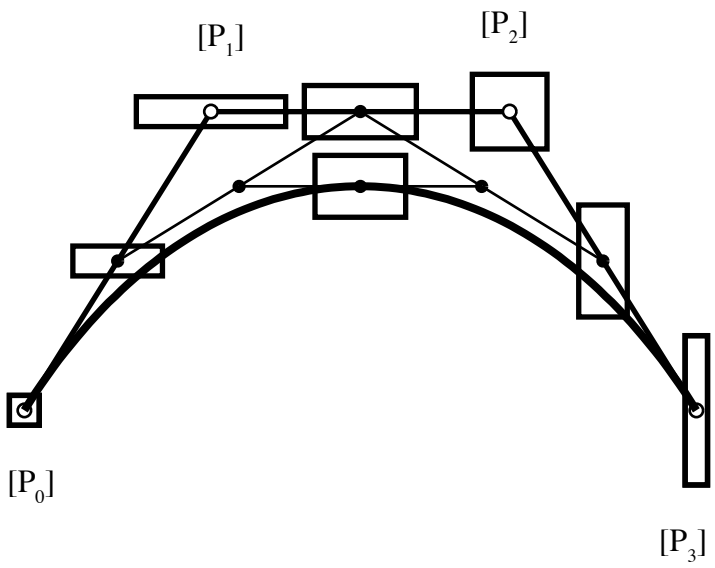


Figure 12.5: Interval de Casteljau algorithm.

12.2.2 Centered form

All the familiar algorithms and characteristics that we associate with Bézier curves — the subdivision and degree elevation algorithms, the variation–diminishing property and convex-hull confinement (see [Far90] for a review) — carry over to Bézier curves with interval coefficients under relatively minor modifications or re–interpretations.

For the case of tensor–product interval control points of the form (12.12), these operations can be more conveniently formulated if the control points are expressed in *centered form*. We re–write each control point $[\,\mathbf{P}_k\,]$ in the following manner:

$$\begin{aligned}
[\,\mathbf{P}_k\,] &= ([\,a_k, b_k\,], [\,c_k, d_k\,]) \\
&= \frac{1}{2}(a_k + b_k, c_k + d_k) \;+\; \frac{1}{2}(b_k - a_k, d_k - c_k)\,[\,-1, +1\,] \\
&= \overline{[\,\mathbf{P}_k\,]} \;+\; \mathbf{e}_k[\,i\,] \qquad (12.21)
\end{aligned}$$

where $[\,i\,]$ denotes the interval $[\,-1, +1\,]$. In (12.21), the point $\overline{[\,\mathbf{P}_k\,]}$ is the *center* of the vector interval $[\,\mathbf{P}_k\,]$, while the vector $\mathbf{e}_k$ is the *error* of $[\,\mathbf{P}_k\,]$. Note that the x and y components of $\mathbf{e}_k$ are necessarily non–negative.

Using centered form, the affine map of two interval points may be written as

$$\begin{aligned}
(1-t)\,[\,\mathbf{P}_0\,] + t\,[\,\mathbf{P}_1\,] &= (1-t)\,\{\overline{[\,\mathbf{P}_0\,]} + \mathbf{e}_0[\,i\,]\} \;+\; t\,\{\overline{[\,\mathbf{P}_1\,]} + \mathbf{e}_1[\,i\,]\} \\
&= \{(1-t)\,\overline{[\,\mathbf{P}_0\,]} + t\,\overline{[\,\mathbf{P}_1\,]}\} + \{(1-t)\,\mathbf{e}_0 + t\,\mathbf{e}_1\}[\,i\,]\,, \qquad (12.22)
\end{aligned}$$

where we assume that $0 \le t \le 1$. Thus, the affine map of two interval points can be computed by independently taking the affine maps of their centers and their errors.

For $0 \le t \le 1$, an interval Bézier curve may be expressed in centered form as follows:

$$\begin{aligned}
[\,\mathbf{P}\,](t) &= \sum_{k=0}^{n} [\,\mathbf{P}_k\,]\, B_k^n(t) \\
&= \sum_{k=0}^{n} \{\overline{[\,\mathbf{P}_k\,]} + \mathbf{e}_k[\,i\,]\}\, B_k^n(t) \\
&= \sum_{k=0}^{n} \overline{[\,\mathbf{P}_k\,]}\, B_k^n(t) \;+\; [\,i\,] \sum_{k=0}^{n} \mathbf{e}_k\, B_k^n(t) \\
&= \overline{[\,\mathbf{P}\,]}(t) + [\,i\,]\,\mathbf{e}(t) \qquad (12.23)
\end{aligned}$$

In bringing the interval $[\,i\,]$ outside the summation sign above we rely on the fact that, for each $k = 0, \ldots, n$, the x and y components of $\mathbf{e}_k$ are non–negative and $B_k^n(t) \ge 0$ for $t \in [\,0, 1\,]$.

Thus, the interval Bézier curve $[\,\mathbf{P}\,](t)$ over $0 \le t \le 1$ can be split into two independent "components" — a *center curve* $\overline{[\,\mathbf{P}\,]}(t)$, and an *error curve* $\mathbf{e}(t)$. Note that the control points $\mathbf{e}_k$ of $\mathbf{e}(t)$ all lie within the first quadrant, and consequently the error curve is itself contained within that quadrant for $t \in [\,0, 1\,]$.

Equation (12.22) suggests that for t outside the unit interval, the error curve should grow monotonically. Writing equation (12.17) in centered form with $[\,a, b\,] = p_0 + e_0[\,i\,]$ and $[\,c, d\,] = p_1 + e_1[\,i\,]$, and noting that

$$\alpha[\,i\,] + \beta[\,i\,] = (|\alpha| + |\beta|)\,[\,i\,] \qquad (12.24)$$

for arbitrary real numbers α and β, we see that the expressions

$$[\,\mathbf{M}\,](p_0 + e_0[\,i\,], p_1 + e_1[\,i\,], t)$$

$$\begin{aligned}
&= \begin{cases} p_0(1-t) + p_1 t \;+\; \{e_0(1-t) - e_1 t\}\,[\,i\,] & \text{for } t < 0\,, \\ p_0(1-t) + p_1 t \;+\; \{e_0(1-t) + e_1 t\}\,[\,i\,] & \text{for } 0 \le t \le 1\,, \\ p_0(1-t) + p_1 t \;+\; \{e_0(t-1) + e_1 t\}\,[\,i\,] & \text{for } t > 1\,, \end{cases} \qquad (12.25) \\
&= p_0(1-t) + p_1 t + \{e_0|1-t| + e_1|t|\}[i] \qquad (12.26)
\end{aligned}$$

describe the behavior of the affine map of two scalar intervals for all real t (see Figure 12.4).

The de Casteljau algorithm (12.19) is essentially a repeated application of the affine map (12.25). Figure 12.5 illustrates for the case $t = \frac{1}{2}$. By studying the behavior of the de Casteljau algorithm applied to an interval Bézier curves outside the unit interval, equation (12.25) leads to the following expressions:

$$[\,\mathbf{P}\,](t) = \begin{cases} \displaystyle\sum_{k=0}^{n} \overline{[\,\mathbf{P}_k\,]}\, B_k^n(t) + \mathbf{e}(t)[\,i\,] & \text{for } 0 \le t \le 1\,, \\ \displaystyle\sum_{k=0}^{n} \overline{[\,\mathbf{P}_k\,]}\, B_k^n(t) + \tilde{\mathbf{e}}(t)[\,i\,] & \text{for } t < 0 \text{ or } t > 1\,, \end{cases} \qquad (12.27)$$

where $\mathbf{e}(t)$ is the error curve for $t \in [\,0, 1\,]$ defined in (12.23), and

$$\tilde{\mathbf{e}}(t) = \sum_{k=0}^{n} (-1)^k \mathbf{e}_k\, B_k^n(t)\,. \qquad (12.28)$$

defines an error curve that is appropriate to the domains $t < 0$ and $t > 1$. (Alternately, we may substitute $|B_k^n(t)| = (-1)^k B_k^n(t)$ when $t < 0$, and $|B_k^n(t)| = (-1)^{n+k} B_k^n(t)$ when $t > 1$, into

$$[\,\mathbf{P}\,](t) \;=\; \sum_{k=0}^{n} \{\overline{[\,\mathbf{P}_k\,]} + \mathbf{e}_k[\,i\,]\}\, B_k^n(t) \tag{12.29}$$

$$= \sum_{k=0}^{n} \overline{[\,\mathbf{P}_k\,]}\, B_k^n(t) \;+\; [\,i\,] \sum_{k=0}^{n} \mathbf{e}_k\, |B_k^n(t)|\,,$$

and set $(-1)^n[\,i\,] = [\,i\,]$ in the case $t > 1$, in order to obtain (12.27)).

12.2.3 Error monotonicity

Both the x and y components of the error curve $\tilde{\mathbf{e}}(t)$ grow monotonically as t departs from the unit interval $[\,0, 1\,]$. To appreciate this, we note that the r–th derivative of the Bézier curve (12.14) can be written as a Bézier curve of degree $n - r$,

$$\mathbf{P}^{(r)}(t) = \frac{n!}{(n-r)!} \sum_{k=0}^{n-r} \Delta^r \mathbf{P}_k\, B_k^{n-r}(t)\,, \tag{12.30}$$

where the quantities $\{\Delta^r \mathbf{P}_k\}$ are the r–th *forward differences* of the control points, given by

$$\Delta^r \mathbf{P}_k = \sum_{j=0}^{r} (-1)^j \binom{r}{j} \mathbf{P}_{k+r-j} \quad \text{for } k = 0, \ldots, n-r\,. \tag{12.31}$$

The forward differences can be generated by the recursion

$$\Delta^{r+1} \mathbf{P}_k = \Delta^r \mathbf{P}_{k+1} - \Delta^r \mathbf{P}_k \tag{12.32}$$

which commences by setting $\Delta \mathbf{P}_k \,(= \Delta^1 \mathbf{P}_k) = \mathbf{P}_{k+1} - \mathbf{P}_k$.

Since the x and y components of the control points of $\tilde{\mathbf{e}}(t)$ are of alternating sign, it is clear from (12.31) that the x and y components of the differences $\Delta^r \tilde{\mathbf{e}}_k$ are all of the *same* sign for $r = 1, \ldots, n$. Thus, the x and y components of all derivatives of $\tilde{\mathbf{e}}(t)$ have the same sign when t lies outside the unit interval, and the magnitude of $\tilde{\mathbf{e}}(t)$ grows monotonically for $t < 0$ or $t > 1$.

Interval arithmetic has a bad reputation for interval width inflation. That is, it often occurs when dealing with interval arithmetic that the widths of the intervals "balloon" so rapidly that the practical value of the interval technique is seriously impaired. For example, interval operations are generally not reversible:

$$([1, 2] + [3, 4]) - [3, 4] = [0, 3]. \tag{12.33}$$

Thus, it is noteworthy that ballooning does not occur with the de Casteljau algorithm when applied within the unit parameter interval. Even after repeated applications of the de Casteljau algorithm to an interval Bézier curve, a point $[\,\mathbf{P}\,](t)$ evaluated on a subdivided region of the curve has the same size as the given point evaluated on the original curve, as long as the subdivision always occurs within the $[\,0, 1\,]$ parameter domain. However, equation (12.27) shows that serious interval inflation will occur when subdividing outside the unit interval.

12.2.4 Envelopes of interval Bézier curves

At first sight, it may seem that making the transition from *interval polynomials* of the form (12.6) to *interval curves* of the form (12.15) is a straightforward matter. However, a number of subtle difficulties arise that deserve attention. It should be noted, for example, that while the bounds on the scalar–valued interval polynomial (12.6) are readily expressible as the two polynomials (12.9), the boundary of the area defined by an interval Bézier curve is *not* (in general) describable by just two polynomial Bézier curves. That boundary is actually the *envelope* of the continuum of rectangles

$$[\,x\,](t)\times[\,y\,](t) \quad \text{for } t \in [\,0,1\,]\,, \tag{12.34}$$

where $[\,x\,](t)$ and $[\,y\,](t)$ are the interval–polynomial components of (12.15).

The region of the plane covered by an interval Bézier curve is actually bounded by Bézier curves and by straight line segments. Consider, for example, the interval Bézier curve in Figure 12.6. There

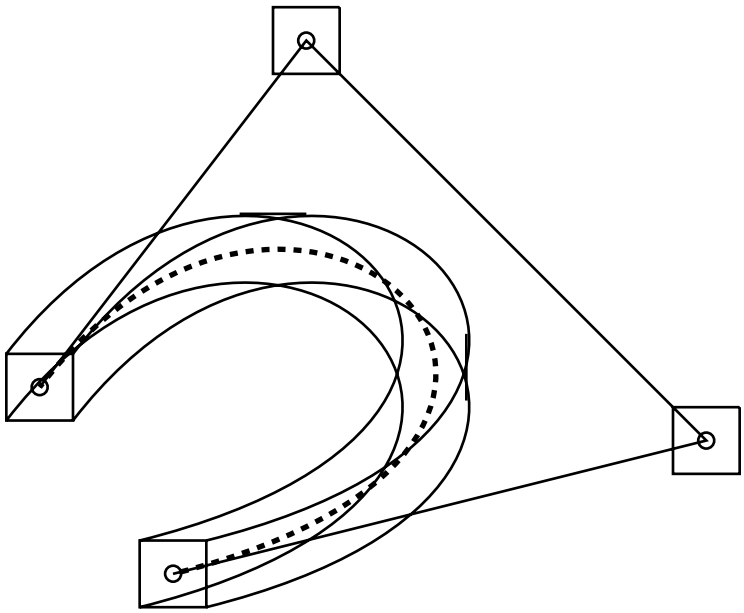


Figure 12.6: The envelope of an interval Bézier curve.

are four possible Bézier curves involved in the envelope for a curve over the unit parameter interval. Those four curves are defined by the four respective rectangle corners of the interval control points.

If the center curve $[\overline{\mathbf{P}}\,](t)$ is progressing NE or SW, the upper envelope bound is defined by control points lying in the top left corner of their respective intervals, and the lower envelope bound is defined by control points lying in the bottom right corner of their respective intervals. Likewise, a curve progressing NW or SE is bounded from above by a curve whose control points lie in top right corners, and from beneath by a curve whose control points lie in bottom left corners. In the case of a horizontal or vertical tangency, straight line segments are introduced into the envelope, equal in width to twice the interval error at the point of tangency.

Thus, we can represent the boundary of the region covered by an interval Bézier curve as a piecewise–Bézier curve of the same degree. Any linear segments that arise in this boundary due to points of horizontal or vertical tangency can be degree–elevated if desired, making the representation of the envelope compatible with standard spline formats.

12.2.5 Interval hodographs

The hodograph of a Bézier curve is simply the first derivative of that curve. The hodograph is itself expressible as a Bézier curve of degree one less than the given curve. If $\{\mathbf{P}_k\}$ are the control points of a Bézier curve of degree n, the control points of the hodograph are $n\Delta\mathbf{P}_k = n(\mathbf{P}_{k+1} - \mathbf{P}_k)$ for $k = 0, \ldots, n-1$ [B86].

We can readily extend this notion to interval Bézier curves, although some caution must be exercised in how the hodograph of an interval Bézier curve is to be interpreted. If the given interval Bézier curve (with control points $[\,\mathbf{P}_k\,]$) is taken to define the *set of all Bézier curves with fixed control points satisfying* $\mathbf{P}_k \in [\,\mathbf{P}_k\,]$, then the hodograph control points can correctly be computed by simply differencing the control points of the original interval Bézier curve. Thus, a point $[\,\mathbf{P}'\,](t)$ on the hodograph defines bounds on the tangent vector for any curve for which $\mathbf{P}_k \in [\,\mathbf{P}_k\,]$.

However, in some applications a given curve is represented by allowing control points to move about as a function of t, within the interval. For example, in [SK91] it was shown that a rational curve can be represented as a polynomial curve for which one of the control points is a rational curve. That "moving control point" can be replaced by an interval which bounds its movement, but the afore–mentioned hodograph does not correctly bound the derivative of the rational curve. The reason is that the moving control point itself moves as a function of t, and therefore the product rule must be applied to correctly represent the derivative of the curve.

12.3 Approximation by interval polynomials

Before proceeding to vector (curve) approximations, it will be convenient to discuss the approximation of scalar functions by interval polynomials expressed in Bernstein form.

Let the function $f(t)$ be differentiable $n+1$ times on the interval $[\,a, b\,]$, and let

$$a \le t_0 < t_1 < \cdots < t_n \le b \tag{12.35}$$

be an ordered sequence of $n+1$ distinct points on that interval. The *Lagrange interpolant* to the sampled values $f_k = f(t_k)$, $k = 0, 1, \ldots, n$ of $f(t)$ at these points is the unique polynomial of degree n given by

$$F_n(t) = \sum_{k=0}^{n} f_k\, L_k(t)\,, \tag{12.36}$$

where the $n+1$ linearly independent polynomials

$$L_k(t) = \prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{t - t_j}{t_k - t_j} \quad \text{for } k = 0, \ldots, n \tag{12.37}$$

constitute the *Lagrange basis* for the sequence of nodes $t_0, t_1, \ldots, t_n$. Since the basis polynomials (12.37) satisfy the conditions

$$L_k(t_j) = \delta_{jk} = \begin{cases} 1 & if \text{j=k}, \\ 0 & otherwise, \end{cases} \tag{12.38}$$

for each $j = 0, \ldots, n$ and $k = 0, \ldots, n$ it is clear that the polynomial $F_n(t)$ reproduces the values of the function $f(t)$ at each of the nodes: $F_n(t_k) = f_k$ for $k = 0, \ldots, n$.

12.3.1 Remainder formulae and interval approximants

At any other point on $[\,a, b\,]$, however, the values of $F_n(t)$ and $f(t)$ disagree in general. Thus, we define the error $E_n(t)$ of the Lagrange interpolant by

$$E_n(t) = f(t) - F_n(t)\,, \tag{12.39}$$

and if we have information on the behavior of the derivative $f^{(n+1)}(t)$ over the interval $[\,a, b\,]$, we may express (12.39) by the *Cauchy remainder* formula [Dav63]:

$$E_n(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (t - t_k) \quad \text{for some } \xi \in (a, b)\,. \tag{12.40}$$

Although, for each t, the value ξ at which the right hand side of (12.40) gives the error $E_n(t)$ of the Lagrange interpolant is not easily determined, if we know *upper and lower bounds* on $f^{(n+1)}(t)$ over $[\,a, b\,]$,

$$f_{\min}^{(n+1)} \le f^{(n+1)}(t) \le f_{\max}^{(n+1)} \quad \text{for all } t \in (a, b) \tag{12.41}$$

then we may write

$$f(t) \in F_n(t) \,+\, \frac{[\,f_{\min}^{(n+1)}, f_{\max}^{(n+1)}\,]}{(n+1)!} \prod_{k=0}^{n} (t - t_k)\,, \tag{12.42}$$

where the right hand side is regarded as *an interval polynomial* $[\,F_{n+1}\,](t)$ *of degree* $n + 1$.

In the particular form (12.42), only the remainder term has a non–degenerate interval coefficient, but if we choose to represent this interval polynomial in another basis — for example, as

$$[\,F_{n+1}\,](t) = \sum_{k=0}^{n+1} [\,a_k, b_k\,] \binom{n+1}{k} \frac{(b-t)^{n+1-k}(t-a)^k}{(b-a)^{n+1}} \tag{12.43}$$

in the Bernstein basis of degree $n + 1$ on $[\,a, b\,]$ — we would find that in general *each* coefficient $[\,a_k, b_k\,]$ is a proper interval of finite width. The formulae giving the interval coefficients of (12.43) in terms of the nodes (12.35), the function values $f_0, f_1, \ldots, f_n$ at those nodes, and the derivative bounds (12.41) for $f(t)$ on $[\,a, b\,]$ are cumbersome to quote in full generality; we shall give explicit formulae below only for the simpler cases of practical interest.

12.3.2 Hermite interpolation

Hermite interpolation is another useful means of polynomial approximation. In general this involves the interpolation of the values and derivatives of $f(t)$ at specified points. Informally, we may regard Hermite interpolation as a limiting form of Lagrange interpolation in which a sequence $m + 1$ consecutive nodes

$$t_k, t_{k+1}, \ldots, t_{k+m+1} \tag{12.44}$$

are allowed to merge; then $F_n(t)$ would be required to match the value $f(t_k)$ and first m derivatives

$$f'(t_k), f''(t_k), \ldots, f^{(m)}(t_k) \tag{12.45}$$

of $f(t)$ at t_k (i.e., $F_n(t)$ has an m–fold osculation to $f(t)$ at t_k).

In particular, suppose that $t_0, t_1, \ldots, t_r$ is a monotone sequence of $r+1$ distinct nodes, and we associate non–negative integers $m_0, m_1, \ldots, m_r$ with these nodes, such that $m_0+m_1+\cdots+m_r+r = n$. Then the unique polynomial $F_n(t)$ that satisfies the interpolation problem

$$F_n^{(i)}(t_k) = f^{(i)}(t_k) \quad \text{for} \;\; i = 0, \ldots, m_r \;\; \text{and} \;\; k = 0, \ldots, r\,, \tag{12.46}$$

where $f^{(0)}(t) \equiv f(t)$ and $F_n^{(0)}(t) \equiv F_n(t)$, has the remainder term

$$E_n(t) = f(t) - F_n(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{r} (t-t_k)^{m_k+1}\,. \tag{12.47}$$

From the above, we can write down an expression analogous to (12.42), namely

$$f(t) \in F_n(t) \;+\; \frac{[\, f_{\min}^{(n+1)}, f_{\max}^{(n+1)}\,]}{(n+1)!} \prod_{k=0}^{r} (t-t_k)^{m_k+1}\,. \tag{12.48}$$

The interval–valued error term on the right has the same coefficient as in (12.43), but its dependence on t is different.

An important instance of the general Hermite problem (12.46) is the symmetric interpolation of values and derivatives to order $(n-1)/2$ of a function $f(t)$ at the end–points of the unit interval $[\,0,1\,]$ by a polynomial $F_n(t)$ of odd degree n,

$$F_n^{(i)}(0) = f^{(i)}(0) \quad \text{and} \quad F_n^{(i)}(1) = f^{(i)}(1) \qquad \text{for} \;\; i = 0, \ldots, (n-1)/2\,. \tag{12.49}$$

If we define the Hermite basis $\{H_k^n(t)\}$ of odd degree n on $[\,0,1\,]$ by demanding that the boundary conditions

$$\left.\frac{d^{\,j} H_k^n}{dt^{\,j}}\right|_{t=0} = \left.\frac{d^{\,j} H_{n-k}^n}{dt^{\,j}}\right|_{t=1} = \delta_{jk} \tag{12.50}$$

be satisfied for $j,k = 0, \ldots, (n-1)/2$ (where δ_{jk} is the Kronecker symbol given by (12.38) above), we can write down the solution to (12.49) in the form

$$F_n(t) \;=\; \sum_{k=0}^{(n-1)/2} f^{(k)}(0) H_k^n(t) + f^{(k)}(1) H_{n-k}^n(t)\,. \tag{12.51}$$

The remainder term (12.40) in this case becomes

$$E_n(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\, t^{(n+1)/2} (t-1)^{(n+1)/2}\,. \tag{12.52}$$

For example, the cubic Hermite basis on $[\,0,1\,]$ is given by

$$\begin{aligned} H_0^3(t) = 2t^3 - 3t^2 + 1\,,\;\; H_1^3(t) = t^3 - 2t^2 + t\,, \\ H_2^3(t) = t^3 - t^2\,,\;\; H_3^3(t) = -2t^3 + 3t^2\,, \end{aligned} \tag{12.53}$$

and plays an important rôle in the construction of cubic splines [dB78].

We now consider some simple special cases. Let the function $f(t)$ be twice differentiable on the interval $[\,0,1\,]$, and let $f_0 = f(0)$ and $f_1 = f(1)$ be its end–point values on that interval. The linear interpolant to these values is just

$$F_1(t) = f_0(1-t) + f_1 t\,, \tag{12.54}$$

and we may write

$$f(t) = F_1(t) \,+\, \frac{f''(\xi)}{2}\, t(t-1) \quad \text{for some } \xi \in (0,1)\,. \tag{12.55}$$

Thus, if the second derivative $f''(t)$ is confined to the interval

$$[\, f'' \,] = [\, f''_{\min}, f''_{\max} \,]\,, \tag{12.56}$$

for all $t \in [\,0,1\,]$, we have

$$f(t) \in F_1(t) \,-\, \frac{[\, f'' \,]}{2}\, t(1-t) \quad \text{for all } t \in [\,0,1\,]\,. \tag{12.57}$$

In order to formulate the above as a quadratic interval polynomial $[\, F_2(t) \,]$, we degree–elevate (12.54) by multiplying the right hand side by $1 = (1-t) + t$ to obtain

$$f(t) \in [\, F_2 \,](t) = \sum_{k=0}^{2} [\, F_{2,k} \,]\, B_k^2(t) \tag{12.58}$$

where

$$[\, F_{2,0} \,] = f_0\,,\ [\, F_{2,1} \,] = \frac{2f_0 + 2f_1 - [\, f'' \,]}{4}\,,\ [\, F_{2,2} \,] = f_1\,. \tag{12.59}$$

Thus, in the case of a linear end–point interpolant, we see that the resulting quadratic interval polynomial bounds $f(t)$ on $[\,0,1\,]$ from below and above by the parabolas

$$\begin{aligned} F_{2,\min}(t) &= f_0 B_0^2(t) \,+\, \frac{2f_0 + 2f_1 - f''_{\max}}{4} B_1^2(t) \,+\, f_1 B_2^2(t)\,, \\ F_{2,\max}(t) &= f_0 B_0^2(t) \,+\, \frac{2f_0 + 2f_1 - f''_{\min}}{4} B_1^2(t) \,+\, f_1 B_2^2(t)\,. \end{aligned} \tag{12.60}$$

For the case of cubic Hermite interpolation to the function values f_0, f_1 and derivatives f'_0, f'_1 at the end–points of $[\,0,1\,]$, the Bernstein–Bézier form of the interpolant is

$$F_3(t) = f_0 B_0^3(t) \,+\, \frac{3f_0 + f'_0}{3} B_1^3(t) \,+\, \frac{3f_1 - f'_1}{3} B_2^3(t) \,+\, f_1 B_3^3(t)\,. \tag{12.61}$$

Thus, if the fourth derivative lies within the interval $[\, f^{(4)} \,] = [\, f^{(4)}_{\min}, f^{(4)}_{\max} \,]$ for all $t \in [\,0,1\,]$, we may write

$$f(t) \in F_3(t) + \frac{[\, f^{(4)} \,]}{24}\, t^2(t-1)^2 = \sum_{k=0}^{4} [\, F_{4,k} \,]\, B_k^4(t) = [\, F_4 \,](t)\,, \tag{12.62}$$

where the coefficients of the quartic interval polynomial on the right are given by

$$\begin{aligned} [\, F_{4,0} \,] = f_0\,,\ [\, F_{4,1} \,] &= f_0 + \frac{1}{4} f'_0\,, \\ [\, F_{4,2} \,] &= \frac{1}{2}(f_0 + f_1) + \frac{1}{6}(f'_0 - f'_1) + \frac{1}{144}[\, f^{(4)} \,]\,, \\ [\, F_{4,3} \,] = f_1 - \frac{1}{4} f'_1\,,\ [\, F_{4,4} \,] &= f_1\,. \end{aligned} \tag{12.63}$$

It should be noted that in all cases of symmetric Hermite interpolation of function values and derivatives to order $(n-1)/2$ at the end points of an interval, only the middle coefficient

$[\,F_{n+1,(n+1)/2}\,]$ of the interpolating interval polynomial $[\,F_{n+1}\,](t)$ has non–zero width, since the residual (12.52) contributes only to the term involving the basis function $B^{n+1}_{(n+1)/2}(t)$. The width of this middle coefficient is just

$$\frac{f^{(n+1)}_{\max} - f^{(n+1)}_{\min}}{\big(\,(n+1)\,n\,\cdots\,\frac{1}{2}(n+3)\,\big)^2}\,. \tag{12.64}$$

Note also that for end–point Hermite interpolation, the width of the interval polynomial $[\,F_{n+1}\,](t)$ always degenerates to zero at $t = 0$ and $t = 1$.

Example 1. With $f(t) = e^t$ we have $f(0) = f'(0) = 1$, $f(1) = f'(1) = e$, and $f^{(4)}(t) \in [\,1, e\,]$ for $t \in [\,0, 1\,]$. The interval control points (12.63) of the Hermite interpolant reflecting these values are then:

$$[\,F_{4,0}\,] = 1\,,\ [\,F_{4,1}\,] = \frac{5}{4}\,,\ [\,F_{4,2}\,] = \frac{1}{144}[\,97 + 48e, 96 + 49e\,]\,, \tag{12.65}$$

$$[\,F_{4,3}\,] = \frac{3}{4}e\,,\ [\,F_{4,4}\,] = e\,.$$

We can gain a better idea of the width of the middle control point from the approximate form $[\,F_{4,2}\,] \approx [\,1.5797, 1.5916\,]$. From (12.10) we now see that the overall width of the approximant $[\,F_4\,](t)$ over $[\,0, 1\,]$ is $(e-1)/720 \approx 0.0024$. Thus, the upper and lower bounds on $f(t) = e^t$ deviate by no more than 1 per cent over the entire interval $[\,0, 1\,]$, and on average much less than this.

Example 2. With $f(t) = \sin(\pi t/2)$ we have $f(0) = 0$, $f'(0) = \pi/2$, $f(1) = 1$, $f'(1) = 0$, and $f^{(4)}(t) \in [\,0, \pi^4/16\,]$ for $t \in [\,0, 1\,]$. In this case, we have

$$[\,F_{4,0}\,] = 0\,,\ [\,F_{4,1}\,] = \frac{\pi}{8}\,,\ [\,F_{4,2}\,] = \frac{1}{2304}[\,192(6+\pi), 192(6+\pi) + \pi^4\,]\,, \tag{12.66}$$

$$[\,F_{4,3}\,] = 1\,,\ [\,F_{4,4}\,] = 1\,.$$

and the interval–valued coefficient is approximately $[\,F_{4,2}\,] \approx [\,0.7618, 0.8041\,]$. The overall width of $[\,F_4\,](t)$ over $[\,0, 1\,]$ is now $\pi^4/11520 \approx 0.0085$, somewhat larger than in the preceding example.

12.3.3 Estimating bounds on derivatives

If the function $f(t)$ to be approximated on $[\,0, 1\,]$ is actually a *polynomial*, we can appeal to its Bernstein–Bézier form,

$$f(t) = \sum_{k=0}^{n} p_k\, B^n_k(t)\,, \tag{12.67}$$

to obtain the derivative bounds required by the approximation procedure. We recall that the r–th derivative of $f(t)$ can be written as

$$f^{(r)}(t) = \frac{n!}{(n-r)!} \sum_{k=0}^{n-r} \Delta^r p_k\, B^{n-r}_k(t)\,, \tag{12.68}$$

and by the convex hull property we then have

$$\frac{n!}{(n-r)!} \min_k \Delta^r p_k \;\le\; f^{(r)}(t) \;\le\; \frac{n!}{(n-r)!} \max_k \Delta^r p_k \quad \text{for } t \in [\,0, 1\,]\,. \tag{12.69}$$

12.4 Approximation by interval Bézier curves

For brevity we restrict our discussion in this paper to the approximation of plane curves; the extension to three dimensions is straightforward. Much of the preceding discussion concerning the interpolation of values sampled from scalar functions applies also to the interpolation of vector–valued functions, i.e., parametric curves. There is, however, an important difference regarding the interpretation of the remainder term that deserves attention.

If $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ is a sequence of points corresponding to $n+1$ distinct parameter values $t_0, t_1, \ldots, t_n$ on a plane parametric curve $\mathbf{P}(t) = \{x(t), y(t)\}$, the Lagrange interpolant $\mathbf{P}_n(t)$ to these points is simply

$$\mathbf{P}_n(t) = \sum_{k=0}^{n} \mathbf{P}_k \, L_k(t) \,, \tag{12.70}$$

where the Lagrange basis is as defined in (12.37) above. But the remainder formula in the vector case is *not* obtained by merely substituting $\mathbf{P}^{(n+1)}(\xi)$ in place of $f^{(n+1)}(\xi)$ in equation (12.40). Rather, we must write the errors for the x and y components of $\mathbf{P}_n(t) = \{X_n(t), Y_n(t)\}$ separately

$$E_{n,x}(t) = x(t) - X_n(t) \quad \text{and} \quad E_{n,y}(t) = y(t) - Y_n(t) \,, \tag{12.71}$$

and we then have

$$x(t) = X_n(t) + \frac{x^{(n+1)}(\xi_1)}{(n+1)!} \prod_{k=0}^{n} (t - t_k) \quad \text{and} \quad y(t) = Y_n(t) + \frac{y^{(n+1)}(\xi_2)}{(n+1)!} \prod_{k=0}^{n} (t - t_k) \tag{12.72}$$

for some $\xi_1, \xi_2 \in (a, b)$, where $\xi_1 \neq \xi_2$ in general. However, defining a vector–valued interval for $\mathbf{P}^{(n+1)}(t)$ over $t \in [\, a, b \,]$ in the form

$$[\, \mathbf{P}^{(n+1)} \,] = [\, x_{\min}^{(n+1)}, x_{\max}^{(n+1)} \,] \times [\, y_{\min}^{(n+1)}, y_{\max}^{(n+1)} \,] \tag{12.73}$$

allows us to express the remainder term as

$$\mathbf{P}(t) \in \mathbf{P}_n(t) + \frac{[\, \mathbf{P}^{(n+1)} \,]}{(n+1)!} \prod_{k=0}^{n} (t - t_k) \,. \tag{12.74}$$

In formulating interval–polynomial approximants to parametric curves, it is natural to consider also *piecewise* interval polynomial forms, i.e., *interval splines*. This extension is not difficult, and we will give here only a brief sketch of some of the pertinent ideas.

It is well known that a curvature–continuous piecewise–cubic curve can be constructed so as to interpolate any ordered sequence of points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ in the plane (which may or may not be sampled from some other curve). Typically, constructing such a curve requires imposing appropriate "end conditions" and then solving a tridiagonal system of linear equations for parametric derivatives $\mathbf{P}'_0, \mathbf{P}'_1, \ldots, \mathbf{P}'_n$ to be assigned to the data points (see [dB78] or [Far90]). For each span k of the spline curve, we then construct the cubic Hermite interpolant $\mathbf{P}(t)$ on $t \in [\, 0, 1 \,]$ to the end–points $\mathbf{P}(0) = \mathbf{P}_{k-1}$, $\mathbf{P}(1) = \mathbf{P}_k$ and end–derivatives $\mathbf{P}'(0) = \mathbf{P}'_{k-1}$, $\mathbf{P}'(1) = \mathbf{P}'_k$,

$$\mathbf{P}(t) = \mathbf{P}(0)H_0^3(t) + \mathbf{P}'(0)H_1^3(t) + \mathbf{P}'(1)H_2^3(t) + \mathbf{P}(1)H_3^3(t) \,. \tag{12.75}$$

If the data points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ were actually sampled from another parametric curve $\mathbf{r}(t)$ and we knew vector–valued bounds on the fourth derivative $\mathbf{r}^{(4)}(t)$ between consecutive points, we could

replace the cubic Hermite arcs (12.75) by quartic interval Bézier arcs of the form

$$[\,\mathbf{P}\,](t) = \sum_{k=0}^{4} [\,\mathbf{P}_{4,k}\,]\, B_k^n(t) \tag{12.76}$$

with control points given by

$$\begin{aligned} [\,\mathbf{P}_{4,0}\,] = \mathbf{P}(0)\,,\ [\,\mathbf{P}_{4,1}\,] = \mathbf{P}(0) + \frac{1}{4}\mathbf{P}'(0)\,, \\ [\,\mathbf{P}_{4,2}\,] = \frac{1}{2}(\mathbf{P}(0) + \mathbf{P}(1)) + \frac{1}{6}(\mathbf{P}'(0) - \mathbf{P}'(1)) + \frac{1}{144}[\,\mathbf{P}^{(4)}\,]\,, \\ [\,\mathbf{P}_{4,3}\,] = \mathbf{P}(1) - \frac{1}{4}\mathbf{P}'(1)\,,\ [\,\mathbf{P}_{4,4}\,] = \mathbf{P}(1)\,. \end{aligned} \tag{12.77}$$

At each of the data points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ the width of the interval spline shrinks to zero.

Example 3. A quarter circle, $\mathbf{P}(t) = (\cos\frac{\pi t}{2}, \sin\frac{\pi t}{2})$, can be approximated as a quartic interval Bézier using the coefficients expressed in equation (12.67):

$$[\,\mathbf{P}_{4,0}\,] = (1,0);\ \ [\,\mathbf{P}_{4,1}\,] = (1, \frac{\pi}{8});\ \ [\,\mathbf{P}_{4,2}\,] = ([0.7618, 0.8041], [0.7618, 0.8041]); \tag{12.78}$$

$$[\,\mathbf{P}_{4,3}\,] = (\frac{\pi}{8}, 1);\ \ [\,\mathbf{P}_{4,4}\,] = (0,1)\,.$$

While a quarter circle can be expressed exactly as a rational curve, the current approximation

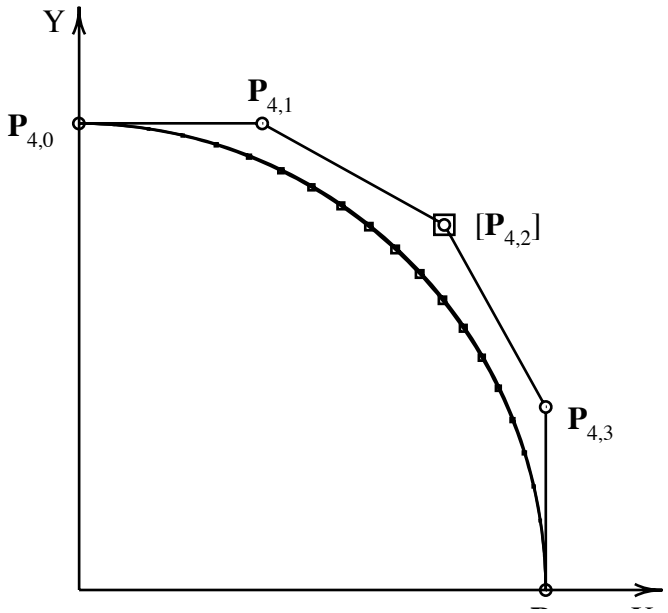


Figure 12.7: Approximate arc length parametrization of circle.

is useful because it approximates an *arc length* parametrization of the circle within two digits of accuracy.

In practice, one might simply choose the middle control point to be at the interval center $[\,\mathbf{P}_{4,2}\,] = (0.78295, 0.78295)$, and use the control point interval as an assurance that the largest possible error, which occurs at $t = 0.5$, is

$$\binom{4}{2}\left(\frac{0.8041 - 0.7618}{2}\right)\left(\frac{1}{2}\right)^4 = 0.0079. \tag{12.79}$$

Chapter 13

Floating Point Error

Most scientific computing is performed in *floating point* arithmetic in the binary number system. For our discussion, consider the floating point number represented using 32 bits of memory, with $d = 23$ bits of fraction and 8 bits of exponent

$$b = \overbrace{s}^{\text{sign}} | \underbrace{e_s}_{\text{exponent sign}} \overbrace{e_7 e_6 e_5 e_4 e_3 e_2 e_1 e_0}^{\text{exponent}} | \overbrace{f_1 f_2 f_3 f_4 \dots f_{21} f_{22} f_{23}}^{\text{fraction}}. \tag{13.1}$$

This string of 32 bits serve as sort of a "code" for representing numbers using the base 2 number system. The decoding formula is to

$$b = (-1)^s f^e = (-1)^s \left[\sum_{i=1}^{d} f_i 2^{-i} \right]^{(-1)^{e_s} \sum j=0^6 e_j 2^j}. \tag{13.2}$$

For example,

$$0|000000100|11010000000000000000000 = .1101_2 \times 2^4 = 1101_2 = 13$$

$$0|100000001|11100000000000000000000 = .111_2 \times 2^{-1} = .0111_2 = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \frac{7}{16}$$

$$1|000000100|10011000000000000000000 = -.10011_2 \times 2^4 = -1001.1_2 = -9\frac{1}{2}$$

One limitation of this floating point number system is that not every number can be represented exactly. For example, the number $\frac{1}{3}$ is a repeating decimal in the binary number system:

$$\frac{1}{3} = 0.0101\bar{0}\bar{1}_2$$

and thus there is no floating point number which exactly equals $\frac{1}{3}$. It lies between the two adjacent floating point numbers

$$0|100000001|10101010101010101010101 < \frac{1}{3} < 0|100000001|10101010101010101010110$$

Therefore, a floating point *number* is best thought of as an *interval* which contains all numbers which are closer to it than to any other floating point number. If d is the number of bits in the fraction of a given floating point number b, then that interval is

$$[b] = [b(1 - 2^{-(d+1)}), b(1 + 2^{-(d+1)})] = b[1 - 2^{-(d+1)}, 1 + 2^{-(d+1)}]. \tag{13.3}$$

This chapter discusses the floating point error incurred in evaluating a Bézier curve. Figure 13.1 shows a Bézier curve which is tangent to the horizontal axis at $t = 0.25$. Due to numerical innacuracies in floating point arithmetic, it is not surprising that when the curve is evaluated at $t = .25$ (using either the de Casteljau algorithm or Horner's method) the answer is not exactly zero. It is interesting to note what happens when we evaluate the curve at several hundred values close to $t = 0.25$. Rather than a producing a smooth graph, the evaluations fluctuate chaotically within a band, as shown in the blowup in figure 13.1.

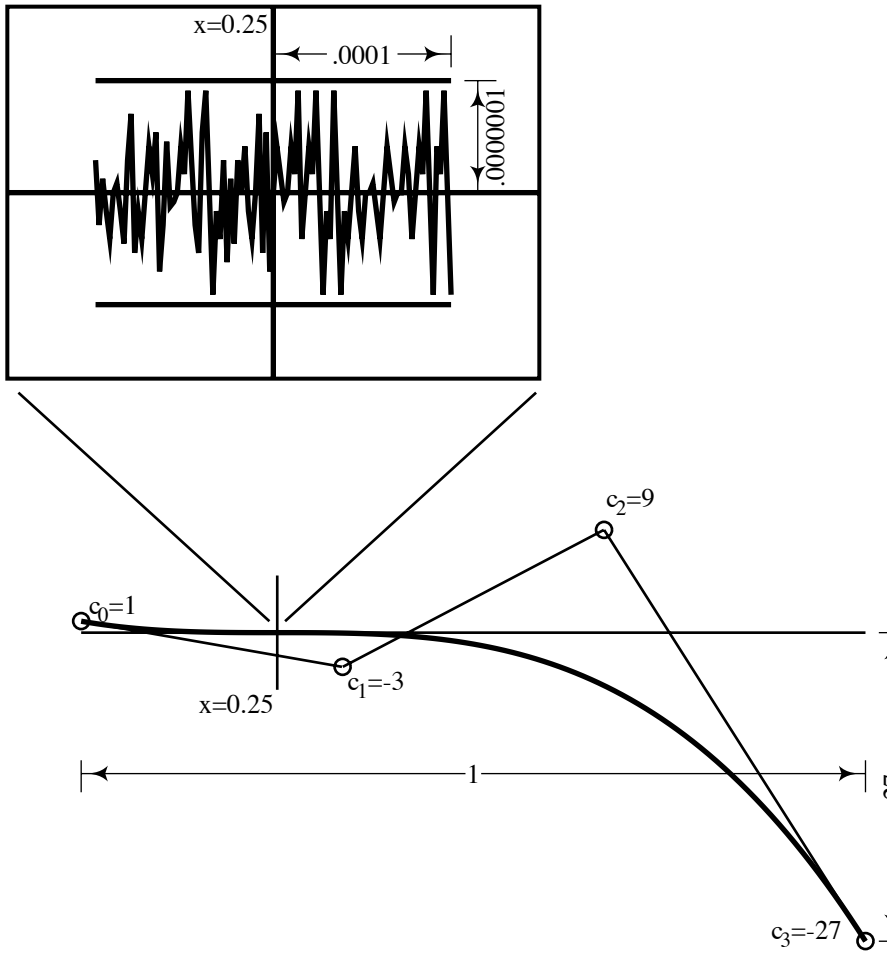


Figure 13.1: Affine map in floating point.

So, how does one determine the effects of floating point error on calculations involving Bézier

curves?

Two sources of error can be considered somewhat independently. The first source of error is in the initial floating point representation of the control points. First

Our previous discussions on interval Bézier curves should be modified slightly if one wishes to accurately bound the uncertainty introduced when using finite–precision arithmetic. Let us first revisit the problem of computing the affine map of two vector intervals shown in Figure 12.4. When finite precision arithmetic is involved, some uncertainty is introduced. In order to robustly represent that uncertainty, the resulting interval should be widened an appropriate amount $\epsilon(t)[i]$ as shown in Figure 13.2. Here, the affine map for $t = \frac{1}{2}$ is shown with the exact arithmetic in solid and floating point error bounds in dashed line.

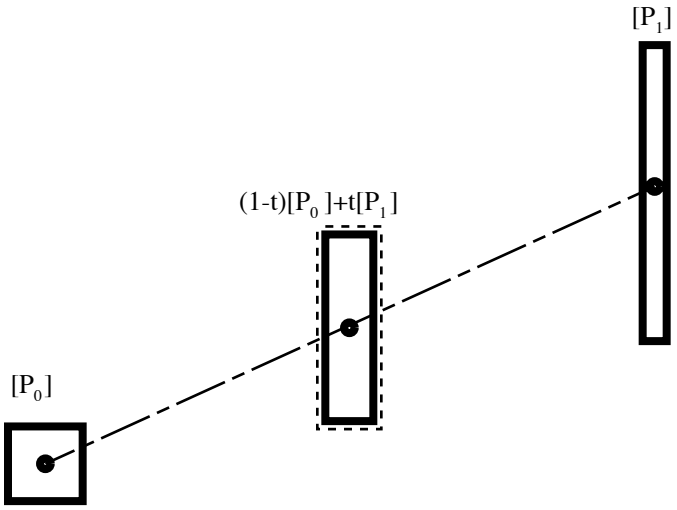


Figure 13.2: Affine map in floating point.

Given a floating point mantissa of d binary digits, the *machine unit* for roundoff is

$$\eta = 2^{-d}. \tag{13.4}$$

If $x * y$, where $* \in \{+, -, \times, \div\}$, denotes an exact computation, and $\mathrm{fl}(x * y)$ denotes the floating point, imprecise computation, then

$$\mathrm{fl}(x * y) \in x * y\,[1 - \eta, 1 + \eta] = x * y\,(1 + \eta[i]). \tag{13.5}$$

Applying this to the affine map equation 12.26 yields (see equation 26 in [FR87])

$$\mathrm{fl}[\,\mathbf{M}\,](p_0 + e_0[\,i\,], p_1 + e_1[\,i\,], t) \in [\,\mathbf{M}\,](p_0 + e_0[\,i\,], p_1 + e_1[\,i\,], t) + \epsilon(t) \tag{13.6}$$

where

$$\begin{aligned} \epsilon(t) &= \begin{cases} \{|p_0|(1-t) - |p_1|t + |p_0(1-t) + p_1 t|\}\,\eta\,[\,i\,] & \text{for } t < 0\,, \\ \{|p_0|(1-t) + |p_1|t + |p_0(1-t) + p_1 t|\}\,\eta\,[\,i\,] & \text{for } 0 \le t \le 1\,, \\ \{|p_0|(t-1) + |p_1|t + |p_0(1-t) + p_1 t|\}\,\eta\,[\,i\,] & \text{for } t > 1\,, \end{cases} && (13.7) \\ &= \{|1-t||p_0| + |t||p_1| + |p_0(1-t) + p_1 t|\}\,\eta\,[\,i\,]\,. && (13.8) \end{aligned}$$

In words, to represent the effects of floating point roundoff in computing an affine map, the rectangle obtained by computing the affine map in exact arithmetic must be fattened an absolute amount $\epsilon(t)$.

Additional discussion on the error propagation in operations on Bernstein–form polynomials can be found in [FR87].

Chapter 14

Free-Form Deformation (FFD)

Free-form deformation (FFD) is a technique for manipulating any shape in a free-form manner. Pierre Bézier used this idea to manipulate large numbers of control points for Béziersurface patches [B́74, B́78], and the power of FFD as a modeling tool was more fully explored in [SP86b]. This chapter discusses the 2D case of FFD.

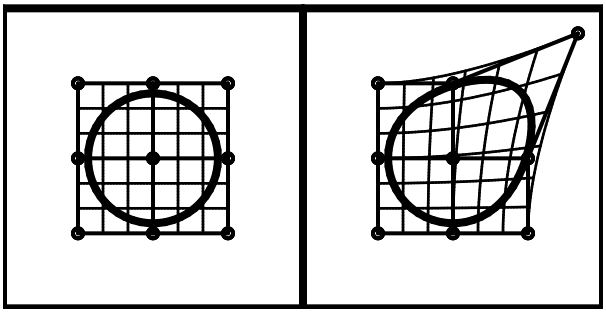


Figure 14.1: FFD example

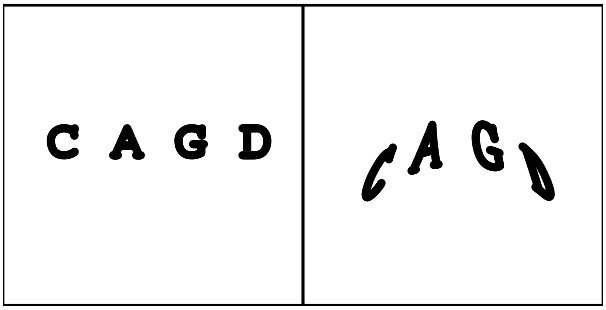


Figure 14.2: FFD example

2D FFD is a map from $R^2 \rightarrow R^2$; that is, it defines a new position for every point in a given rectangular region. In Figure 14.1, the FFD is specified using the nine control points. The undeformed

scene appears at the left, and the right shows what happens the the grid and circle after the control points are moved. The grid helps to visualize how FFD works. FFD is a powerful modeling tool because anything drawn inside of the initial, undeformed rectangle will experience the distortion, such as the text in Figure 14.2.

Denote by (X_{min}, Y_{min}) and (X_{max}, Y_{max}) the corners of a deformation region, and by m and n the degrees of the FFD function (there are $m+1$ vertical columns and $n+1$ horizontal rows of control points).

FFD is a two-step process:

1. Compute the (s,t) coordinates for each point to be deformed. The s and t coordinates of a point in the deformation region range between 0 and 1 (see Figure 14.3). For a point in the

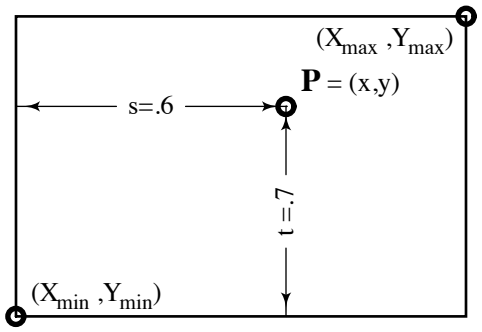


Figure 14.3: FFD local coordinates

rectangular region whose Cartesian coordinates are (x,y),

$$s = \frac{x - X_{min}}{X_{max} - X_{min}}, \quad t = \frac{y - Y_{min}}{Y_{max} - Y_{min}}. \tag{14.1}$$

2. Compute the homogeneous coordinates $\mathbf{X}(s,t) = (X,Y,W)$ of the deformed point using the **rational bivariate tensor product Bernstein polynomial** equation

$$\mathbf{X}(s,t) = \sum_{j=0}^{n} \sum_{i=0}^{m} B_i^m(s) B_j^n(t) \mathbf{P}_{ij} \tag{14.2}$$

where $B_i^n(t)$ and $B_j^m(s)$ are Bernstein polynomials and $\mathbf{P}_{ij} = w_{ij}(x_{ij}, y_{ij}, 1)$ are the homogeneous coordinates of the displaced control point i,j. Note that weights can be assigned to the FFD control points.

If all weights $w_{ij} = 1$ and if the control points form a rectangular lattice

$$\mathbf{P}_{i,j} = \left(X_{min} + \frac{i}{m}(X_{max} - X_{min}), Y_{min} + \frac{j}{n}(Y_{max} - Y_{min}) \right) \tag{14.3}$$

as shown in Figure 14.4, FFD is the identity transformation: all points end up where they started from.

Points outside of the rectangle are not moved. If a shape is only partially inside the FFD region, one can control the degree of continuity between the deformed and undeformed portions of the shape by "freezing" rows of control points, as shown in Figure 14.5.

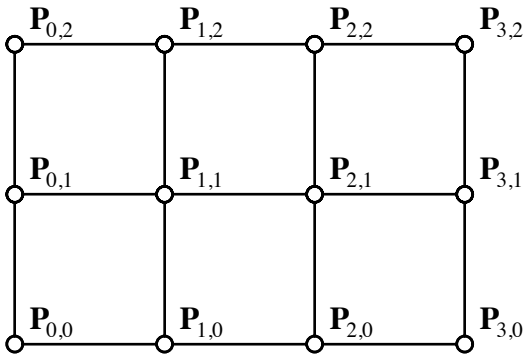


Figure 14.4: FFD undisplaced control points

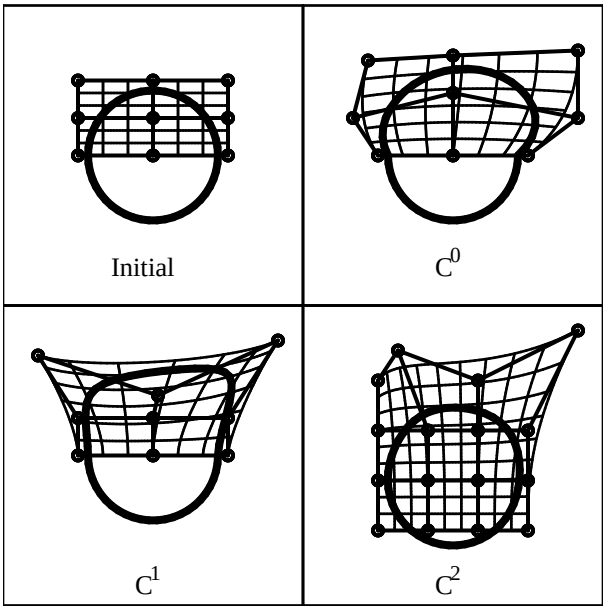
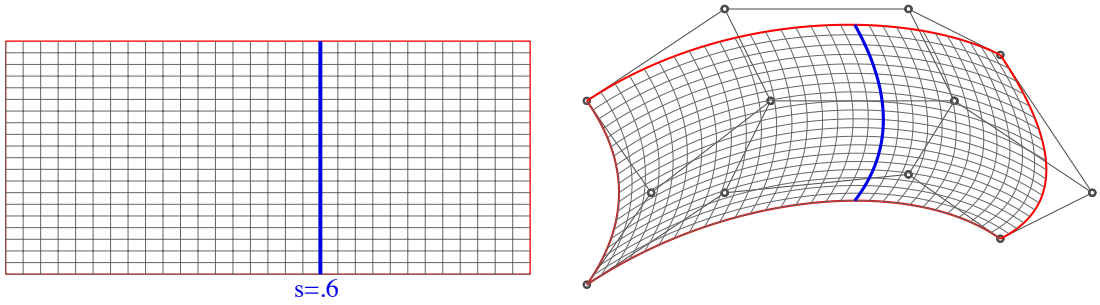


Figure 14.5: Continuity control

14.0.1 Deformed Lines

Each vertical line that undergoes FFD maps to a Bézier curve of degree n, and each horizontal line maps to a Bézier curve of degree m. Thus, the "vertical" curves in Figure 14.6.b are degree-two Bézier curves and the "horizontal" curves are degree three. In particular, we have highlighted the



(a) Undeformed Grid with un-displaced control points. (b) Deformed Grid.

Figure 14.6: FFD Applied to a Grid of Lines.

line $s = .6$ in Figure 14.6.a and its deformation is the degree-two Bézier curve shown in Figure 14.6.b. The control points for this Bézier curves are determined as follows.

Referring to Figure 14.7, consider each horizontal row of control points in the FFD control grid to be a control polygon for a cubic Bézier curve. Call these curves $\mathbf{H}_0(s)$, $\mathbf{H}_1(s)$, and $\mathbf{H}_2(s)$ where the control points for $\mathbf{H}_i(s)$ are $\mathbf{P}_{0,i}$, $\mathbf{P}_{1,i}$, $\mathbf{P}_{2,i}$, $\mathbf{P}_{3,i}$. Then, the three control points for the

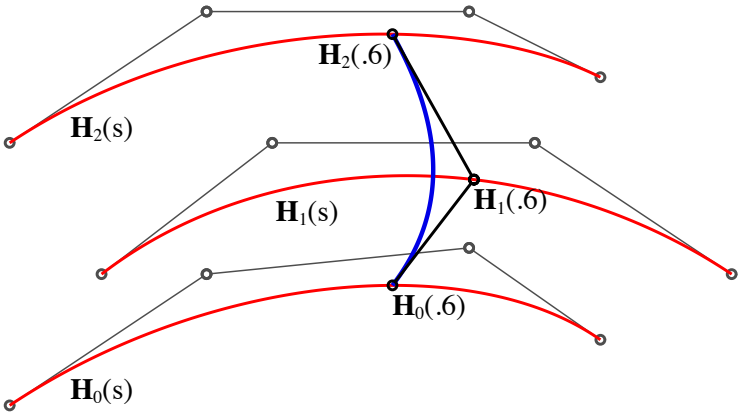
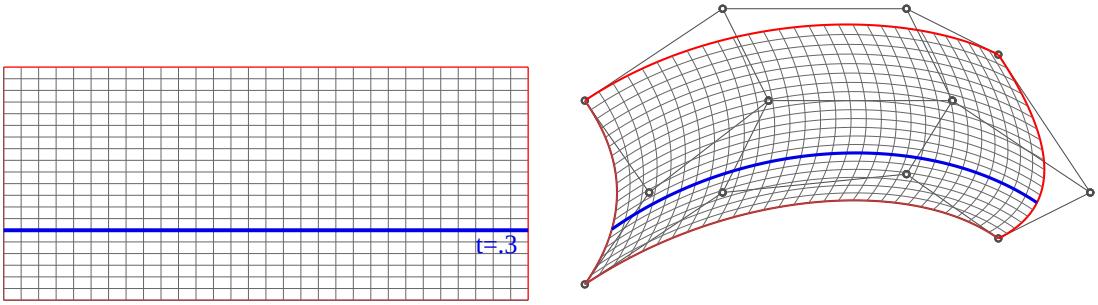


Figure 14.7: Control points of deformed line.

degree-two Bézier curve in Figure 14.6.b are $\mathbf{H}_0(.6)$, $\mathbf{H}_1(.6)$, and $\mathbf{H}_2(.6)$.



(a) Undeformed Grid with horizontal line at $t = .3$. (b) Deformed Grid.

Figure 14.8: FFD Applied to a horizontal line $t = .3$

The horizontal line $t = .3$ is highlighted in Figure 14.8.a and its deformation is the degree-three Bézier curve shown in Figure 14.8.b. The control points for this Bézier curves are determined as follows.

Referring to Figure 14.9, consider each vertical column of control points in the FFD control grid to be a control polygon for a quadratic Bézier curve. Call these curves $\mathbf{V}_0(s)$, $\mathbf{V}_1(s)$, $\mathbf{V}_2(s)$, and $\mathbf{V}_3(s)$ where the control points for $\mathbf{V}_i(s)$ are $\mathbf{P}_{i,0}$, $\mathbf{P}_{i,1}$, $\mathbf{P}_{i,2}$. Then, the four control points for

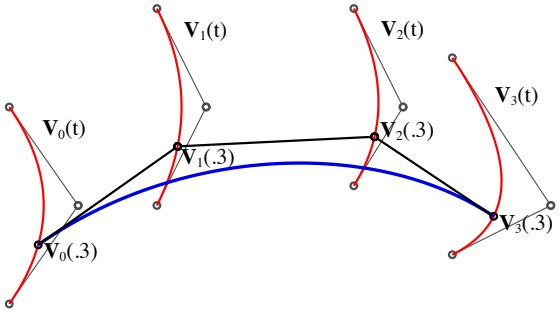


Figure 14.9: Control points of deformed line.

the degree-three Bézier curve in Figure 14.9.b are $\mathbf{V}_0(.3)$, $\mathbf{V}_1(.3)$, $\mathbf{V}_2(.3)$, and $\mathbf{V}_3(.3)$.

Chapter 15

TENSOR-PRODUCT SURFACES

The product of a column vector and a row vector is an example of the mathematical operation of tensor-product:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \otimes \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_0b_0 & a_0b_1 & a_0b_2 & a_0b_3 \\ a_1b_0 & a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_0 & a_2b_1 & a_2b_2 & a_2b_3 \end{bmatrix} \tag{15.1}$$

A *tensor-product surface* $\mathbf{P}(s,t)$ is one whose blending functions are products of pairs of univariate blending functions:

$$\mathbf{P}(s,t) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}\mathbf{P}_{ij}B_i^m(s)B_j^n(t)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}B_i^m(s)B_j^n(t)} \tag{15.2}$$

where the $\mathbf{P}_{ij}$ are control points, w_{ij} are weights, and $B_i^m(s)$ and $B_j^n(t)$ are univariate blending functions. The control points in a tensor-product surface are organized topologically into a rectangular array, and the blending functions corresponding to the control points are likewise organized in an array similar to the one in (15.1).

Tensor-product surfaces can be constructed from any two types of univariate blending functions, and there is no requirement that the two sets of blending functions are of the same type or degree. If the univariate blending functions are Bernstein polynomials, the surface is a Bézier surface patch (see Section 15.1). If the blending functions are B-Splines, the surface is a NURBS surface (see Section 15.6).

15.1 Tensor-Product Bézier Surface Patches

For a Tensor-Product Bézier surface patch, m and n in (15.2) are the degrees of the respective Bernstein polynomials. If $m = n = 3$, the surface is called a "bi-cubic" patch Likewise, the case $m = n = 2$ is called "bi-quadratic," $m = n = 1$ is called "bi-linear," etc. If $m \neq n$, we normally just say that the surface is degree $m \times n$.

Figure 15.1.a shows a tensor-product Bézier surface patch of degree 2×3. Note that the $(m+1) \times (n+1)$ control points that are organized topologically into $m+1$ "rows" and $n+1$ "columns." The control points along with the blue lines connecting them in Figure 15.1.a is called the *control grid* or *control mesh* of the surface (analogous to the control polygon for a curve). The control points

for a bicubic patch are organized as follows:

$$\begin{bmatrix} \mathbf{P}_{03} & \mathbf{P}_{13} & \mathbf{P}_{23} & \mathbf{P}_{33} \\ \mathbf{P}_{02} & \mathbf{P}_{12} & \mathbf{P}_{22} & \mathbf{P}_{32} \\ \mathbf{P}_{01} & \mathbf{P}_{11} & \mathbf{P}_{21} & \mathbf{P}_{31} \\ \mathbf{P}_{00} & \mathbf{P}_{10} & \mathbf{P}_{20} & \mathbf{P}_{30} \end{bmatrix} \tag{15.3}$$

The corresponding blending functions are

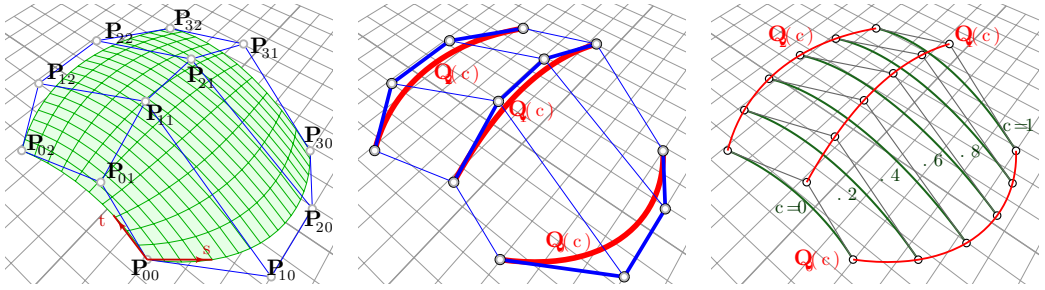
$$\begin{bmatrix} (1-s)^3 \cdot t^3 & 3s(1-s)^2 \cdot t^3 & 3s^2(1-s) \cdot t^3 & s^3 \cdot t^3 \\ (1-s)^3 \cdot 3t^2(1-t) & 3s(1-s)^2 \cdot 3t^2(1-t) & 3s^2(1-s) \cdot 3t^2(1-t) & s^3 \cdot 3t^2(1-t) \\ (1-s)^3 \cdot 3t(1-t)^2 & 3s(1-s)^2 \cdot 3t(1-t)^2 & 3s^2(1-s) \cdot 3t(1-t)^2 & s^3 \cdot 3t(1-t)^2 \\ (1-s)^3 \cdot (1-t)^3 & 3s(1-s)^2 \cdot (1-t)^3 & 3s^2(1-s) \cdot (1-t)^3 & s^3 \cdot (1-t)^3 \end{bmatrix} \tag{15.4}$$

Each row and column of control points can be interpreted as defining a curve. We will refer to the curve defined by control points $\mathbf{P}_{ij},\ j = 0, 1, \ldots n$ as the i^{th} s-control curve and the curve defined by control points $\mathbf{P}_{ij},\ i = 0, 1, \ldots m$ as the j^{th} t-control curve. Thus, there are $m+1$ s-control curves, and $n+1$ t-control curves. (The term "control curve" is not in standard use; it is a helpful concept, but is not used outside of these notes.)

If we fix one of the two parameters of the surface (for example, let $s = c$) then (15.2) reduces to

$$\mathbf{P}(c,t) = \sum_{j=0}^{n} \left[\sum_{i=0}^{m} \mathbf{P}_{ij} B_i^m(c) \right] B_j^n(t) = \sum_{j=0}^{n} \mathbf{Q}_j(c) B_j^n(t) \tag{15.5}$$

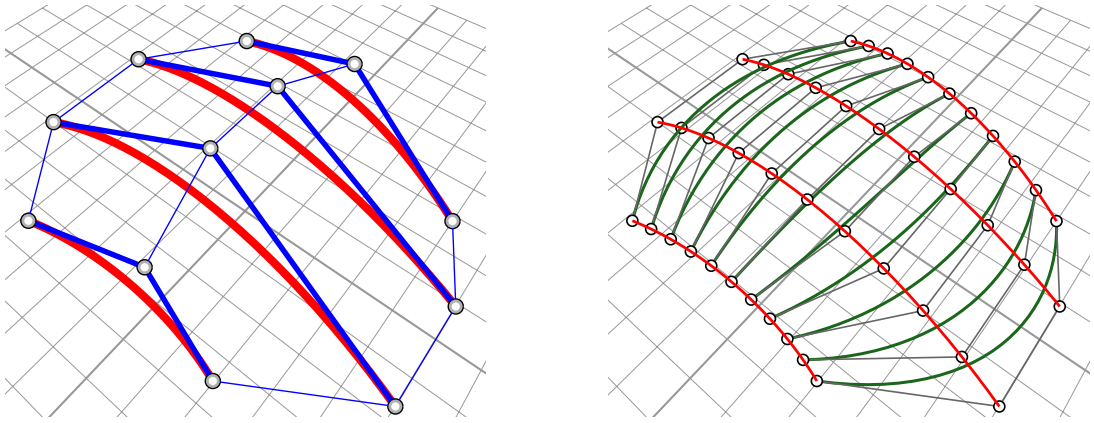
where $\mathbf{Q}_j(c) = \sum_{i=0}^{m} \mathbf{P}_{ij} B_i^m(c)$. $\mathbf{P}(c,t)$ is called an *s-iso-parameter* curve. *Iso* means *constant*, and an iso-parameter curve is a curve on a surface defined by holding one of the parameters constant.



(a) The surface and its control grid. (b) s-Control Curves. (c) Some s-iso-parameter curves $\mathbf{P}(c,t)$.

Figure 15.1: Bézier surface patch of degree 2×3.

A surface can be thought of as a family of iso-parameter curves, and the construction of those iso-parameter curves can be easily described in terms of control curves. Figure 15.1.b shows the s-control curves for the surface in Figure 15.1.a. Imagine that these control curves are wires, and imagine that a bead is free to slide along each wire. These beads serve as control points for the family of s-iso-parameter curves. That is, for a fixed value c, if we position each bead j at $\mathbf{Q}_j(c)$, they define the Bézier control point positions for s-iso-parameter curve $\mathbf{P}(c,t)$. Figure 15.1.c shows six such iso-parameter curves. Of course, $\mathbf{P}(s,t)$ can likewise be viewed as a family of t-iso-parameter curves defined by t-control curves, as illustrated in Figure 15.2.



(a) t-Control curves.

(b) Some t-iso-parameter curves $\mathbf{P}(s,c)$ for the surface in Figure 15.1.a.

Figure 15.2: Surface in Figure 15.1.a viewed as a family of t-iso-parameter curves.

In general, the only control points that lie on a Bézier surface patch are the corners:

$$\mathbf{P}(0,0) = \mathbf{P}_{00}, \quad \mathbf{P}(1,0) = \mathbf{P}_{m0}, \quad \mathbf{P}(0,1) = \mathbf{P}_{0n}, \quad \mathbf{P}(1,1) = \mathbf{P}_{mn}.$$

To evaluate the point $\mathbf{P}(\sigma,\tau)$, we could first evaluate the t-control curves at $t = \tau$, and then evaluate the t-iso-parameter curve at $s = \sigma$. Alternatively, we could first evaluate the s-control curves at $s = \sigma$, and then evaluate the s-iso-parameter curve at $t = \tau$.

Four special iso-parameter curves are the boundary curves $\mathbf{P}(0,t)$, $\mathbf{P}(1,t)$, $\mathbf{P}(s,0)$, and $\mathbf{P}(s,1)$. These boundary curves are Bézier curves defined by the corresponding boundary rows or columns of control points, and hence are control curves. For example, $\mathbf{P}(0,t)$ is the degree-three Bézier curve whose control points are $\mathbf{P}_{00}$, $\mathbf{P}_{10}$, $\mathbf{P}_{20}$, and $\mathbf{P}_{30}$. Likewise, $\mathbf{P}(s,1)$ is the degree-two Bézier curve whose control points are $\mathbf{P}_{30}$, $\mathbf{P}_{31}$, and $\mathbf{P}_{32}$.

15.2 The de Casteljau Algorithm for Bézier Surface Patches

Section 2.2 introduced the notation $\mathbf{P}_{[t_0,t_1]}(t)$ to mean a curve defined over the domain $t \in [t_0,t_1]$. In like manner, we will denote by

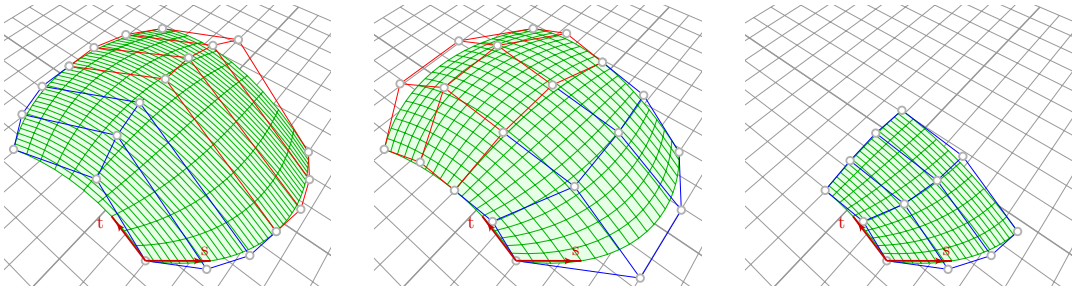
$$\mathbf{P}_{[s_0,s_1]\times[t_0,t_1]}(s,t)$$

the surface defined over the domain $s \in [s_0,s_1]$, $t \in [t_0,t_1]$.

The de Casteljau algorithm can be used to cut a Bézier surface patch $\mathbf{P}_{[s_0,s_1]\times[t_0,t_1]}(s,t)$ into two pieces, either in the s or t parameter directions. Cutting at $s = \sigma$ would produce $\mathbf{P}_{[s_0,\sigma]\times[t_0,t_1]}(s,t)$ and $\mathbf{P}_{[\sigma,s_1]\times[t_0,t_1]}(s,t)$. Cutting at $t = \tau$ would produce $\mathbf{P}_{[s_0,s_1]\times[t_0,\tau]}(s,t)$ and $\mathbf{P}_{[s_0,s_1]\times[\tau,t_1]}(s,t)$.

To subdivide a Bézier surface patch at $s = \sigma$, simply apply the de Casteljau algorithm for curves to subdivide each of the s-control curves at $s = \sigma$. To subdivide a Bézier surface patch at $t = \tau$, subdivide each of the t-control curves at $t = \tau$.

By repeated application of the de Casteljau algorithm, it is possible to obtain a patch over any sub-domain. For example, the patch $\mathbf{P}_{[0,\frac{1}{2}]\times[0,\frac{1}{2}]}$ in Figure 15.3.c can be obtained by splitting $\mathbf{P}_{[0,1]\times[0,1]}$ at $t = \frac{1}{2}$ to obtain $\mathbf{P}_{[0,1]\times[0,\frac{1}{2}]}$ and $\mathbf{P}_{[0,\frac{1}{2}]\times[\frac{1}{2},1]}$, and then splitting $\mathbf{P}_{[0,1]\times[0,\frac{1}{2}]}$ at $s = \frac{1}{2}$ to obtain $\mathbf{P}_{[0,\frac{1}{2}]\times[0,\frac{1}{2}]}$ and $\mathbf{P}_{[\frac{1}{2},1]\times[0,\frac{1}{2}]}$.



(a) Subdivision in the s direction.

(b) Subdivision in the t direction.

(c) $\mathbf{P}_{[0,\frac{1}{2}]\times[0,\frac{1}{2}]}$, obtained by subdividing at $s = \frac{1}{2}$ and then at $t = \frac{1}{2}$.

Figure 15.3: Applying the de Casteljau algorithm to the surface in Figure 15.1.a.

The idea of control curves also leads to a straightforward explanation of how to degree elevate a Bézier surface patch: simply degree elevate all of the s and/or t control curves.

15.3 Tangents and Normals

Tangent and normal vectors for a surface patch can be obtained by computing partial derivatives. As with Bézier curves, it is easiest to compute derivatives at a corner of a Bézier surface patch.

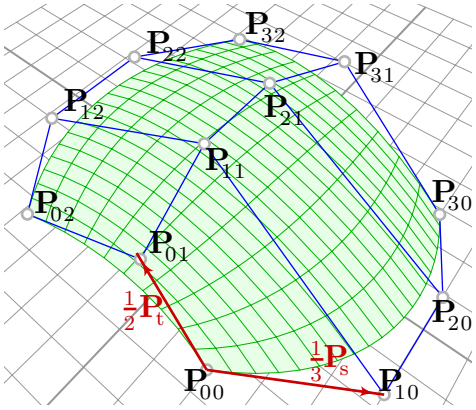


Figure 15.4: Partial derivative vectors for $\mathbf{P}_{[0,1]\times[0,1]}(s,t)$(assuming weights are unity).

For a rational Bézier surface patch $\mathbf{P}_{[s_0,s_1]\times[t_0,t_1]}(s,t)$, the partial derivatives at corner $\mathbf{P}(0,0)$ are

$$\mathbf{P}_s(s_0,t_0) = \frac{m}{s_1-s_0}\frac{w_{10}}{w_{00}}(\mathbf{P}_{10}-\mathbf{P}_{00}), \qquad \mathbf{P}_t(s_0,t_0) = \frac{n}{t_1-t_0}\frac{w_{01}}{w_{00}}(\mathbf{P}_{01}-\mathbf{P}_{00})$$

These partial derivatives are illustrated in Figure 15.4. The second partial derivatives are given by

$$\mathbf{P}_{ss}(s_0,t_0) = \frac{n(n-1)\frac{w_{20}}{w_{00}}(\mathbf{P}_{20}-\mathbf{P}_{00}) - 2n\frac{w_{10}}{w_{00}}\frac{nw_{10}-w_{00}}{w_{00}}(\mathbf{P}_{10}-\mathbf{P}_{00})}{(s_1-s_0)^2}$$

$$\mathbf{P}_{tt}(s_0,t_0)=\frac{n(n-1)\frac{w_{02}}{w_{00}}(\mathbf{P}_{02}-\mathbf{P}_{00})-2n\frac{w_{01}}{w_{00}}\frac{nw_{01}-w_{00}}{w_{00}}(\mathbf{P}_{01}-\mathbf{P}_{00})}{(t_1-t_0)^2}$$

Letting $\hat{\mathbf{P}}_{ij}=\mathbf{P}_{ij}-\mathbf{P}_{00}$,

$$\mathbf{P}_{st}(s_0,t_0)=\frac{mn}{w_{00}^2(s_1-s_0)(t_1-t_0)}(w_{00}w_{11}\hat{\mathbf{P}}_{11}-w_{10}w_{01}\hat{\mathbf{P}}_{10}-w_{10}w_{01}\hat{\mathbf{P}}_{01})$$

A vector that is perpendicular to the surface at $\mathbf{P}(0,0)$ can be obtained by taking the cross product $\mathbf{P}_s(0,0)\times\mathbf{P}_t(0,0)$. To find the normal vector at any other point on the surface, you can first perform de Casteljau subdivisions to move the desired point to a corner.

15.4 Tessellation of Bézier Curves and Surfaces

In computer graphics and geometric modeling, parametric curves and surfaces are often tessellated into piecewise linear segments for various applications, like rendering, mesh generation, and surface/surface intersection. A simple approach is to sample the curve or the surface evenly in its domain and then properly connect the sampling points to form a polyline or a triangular mesh. In this approach, the key point is to determine the sampling number. In the following we give formulae to compute the appropriate sampling number(s) such that the deviation of the approximate piecewise linear segments from the original curve or surface is within given tolerance ϵ.

15.4.1 The curve case

Section 10.6 derives the following result. Consider a degree n Bezier curve defined by

$$r(t)=\sum_{i=0}^{n}P_iB_i^n(t),\quad t\in[0,1].$$

If $L(t)$ is a straight line segment connecting $r(\alpha)$ and $r(\beta)$ where $0\le\alpha<\beta\le1$, then the deviation of $L(t)$ from $r(t)$ within domain $[\alpha,\beta]$ is bounded by (see Wang 1984, Filip et al 1986)

$$\max_{t\in[\alpha,\beta]}\|r(t)-L(t)\|\le\frac{1}{8}\max_{t\in[\alpha,\beta]}\|r''(t)\|(\beta-\alpha)^2.$$

This provides a way to compute the parameter interval (or step size) δ given a tolerance ϵ:

$$\delta\le\begin{cases}\sqrt{\dfrac{8\epsilon}{\max\limits_{t\in[\alpha,\beta]}\|r''(t)\|}}, & \text{if } \|r''(t)\|\neq0\\ 1, & \text{otherwise}\end{cases}$$

The sampling number n_t is the reciprocal of the parameter interval, i.e., $n_t=\lceil1/\delta\rceil$. That is, if the curve is evenly sampled at n_t+1 parameter values, the final polyline will not deviate from the curve greater than ϵ. In practice, the norm of the second derivative of $r(t)$ can be bounded by the Bezier control points:

$$\max_{t\in[\alpha,\beta]}\|r''(t)\|\le n(n-1)\max_{0\le i\le n-2}\|P_{i+2}-2P_{i+1}+P_i\|.$$

15.4.2 The surface case

For a surface $r(s,t)$, $(s,t) \in [0,1]\times[0,1]$, we choose a sub-domain $[\alpha,\beta]\times[\zeta,\eta]$ and construct a piecewise linear function $L(s,t)$ which consists of two triangles $r(\alpha,\beta)r(\zeta,\beta)r(\alpha,\eta)$ and $r(\zeta,\eta)r(\alpha,\eta)r(\zeta,\beta)$. It has been proved that

$$\max_{(s,t)\in[\alpha,\beta]\times[\zeta,\eta]} \|r(s,t) - L(s,t)\| \le \frac{1}{8}(M_1(\beta-\alpha)^2 + 2M_2(\beta-\alpha)(\eta-\zeta) + M_3(\eta-\zeta)^2)$$

where

$$\begin{aligned} M_1 &= \max_{(s,t)\in[\alpha,\beta]\times[\zeta,\eta]} \|r''_{ss}(s,t)\| \\ M_2 &= \max_{(s,t)\in[\alpha,\beta]\times[\zeta,\eta]} \|r''_{st}(s,t)\| \\ M_3 &= \max_{(s,t)\in[\alpha,\beta]\times[\zeta,\eta]} \|r''_{tt}(s,t)\|. \end{aligned}$$

Given tolerance ϵ, we would like to determine the parameter intervals δ_s and δ_t for $s-$ and $t-$directions such that

$$M_1\delta_s^2 + 2M_2\delta_s\delta_t + M_3\delta_t^2 \le 8\epsilon.$$

Case 1. If $M_1 = 0$, the surface is a ruled surface (linear in $s-$direction). Then we choose $\delta_s = 1$ and $\delta_t = \dfrac{\sqrt{M_2^2 + 8M_3\epsilon} - M_2}{M_3}$.

Case 2. If $M_3 = 0$, the surface is also a ruled surface (linear in $t-$direction). Then we choose $\delta_t = 1$ and $\delta_s = \dfrac{\sqrt{M_2^2 + 8M_1\epsilon} - M_2}{M_1}$.

Case 3. In general, the above inequality contains two unknowns δ_s and δ_t. We need to impose an additional constraint to solve for them. Suppose $\delta_s = k\delta_t$. Then $\delta_t = \sqrt{\dfrac{8\epsilon}{M_1k^2 + 2M_2k + M_3}}$. The default value for k is $k = \sqrt{M_3/M_1}$.

The above choice for the value of k is to minimize the number of resulting triangles. The reasoning is as follows. The sampling numbers n_s and n_t are the reciprocal of the parameter intervals δ_s and δ_t, i.e., $n_s = \lceil 1/\delta_s \rceil$ and $n_t = \lceil 1/\delta_t \rceil$. Thus the number of the triangles is $2n_sn_t$. So we have to choose k to maximize $\delta_s\delta_t$. Note that

$$\delta_s\delta_t = \frac{8\epsilon k}{M_1k^2 + 2M_2k + M_3}.$$

This requires us to minimize $M_1k + M_3/k$, which gives a unique solution $k = \sqrt{M_3/M_1}$. Now the surface is sampled evenly at $(n_s+1)(n_t+1)$ parameter values and the triangulation can be formed by properly connecting the sampling points. In this way, it is guaranteed that the final triangle mesh approximates the surface within the tolerance ϵ.

For a degree $m \times n$ Bezier patch

$$r(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}B_i^m(s)B_j^n(t), \quad s,t \in [0,1]$$

bounds on the norms of its second derivatives are given by:

$$\begin{aligned} M_1 &\leq m(m-1) \max_{\substack{0\leq i\leq m-2\\ 0\leq j\leq n}} \|P_{i+2,j} - 2P_{i+1,j} + P_{ij}\| \\ M_2 &\leq mn \max_{\substack{0\leq i\leq m-1\\ 0\leq j\leq n-1}} \|P_{i+1,j+1} - P_{i+1,j} - P_{i,j+1} + P_{ij}\| \\ M_3 &\leq n(n-1) \max_{\substack{0\leq i\leq m\\ 0\leq j\leq n-2}} \|P_{i,j+2} - 2P_{i,j+1} + P_{ij}\|. \end{aligned} \tag{15.6}$$

Remark 1. When tessellating a few connecting patches, to avoid cracks, it is a good idea to compute sampling number for all four boundary curves independently of the sampling numbers for the patch interior (see Rockwood et al. 1989).

Remark 2. The above approach can also be applied to tessellating rational curves or surfaces. But the estimation of the second derivative bounds is very difficult. A simple and efficient way has been developed (see Zheng and Sederberg 2000).

Example
How many rows and columns of triangles are needed to tessellate the patch in Figure 15.5 using the fewest number of triangles such that the approximation error $\epsilon \leq .001$?

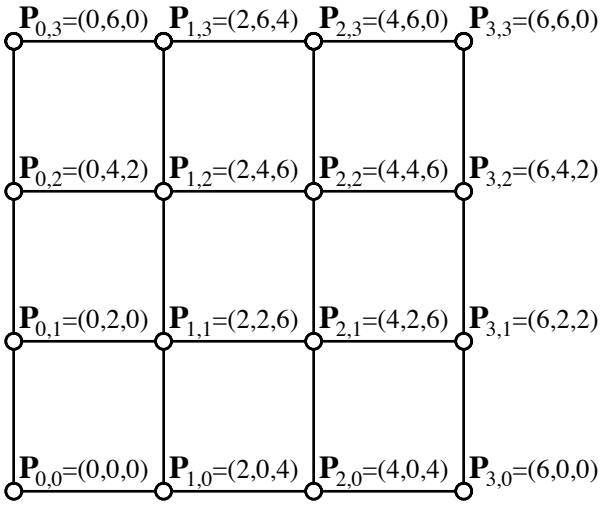


Figure 15.5: Surface Control Grid

Solution Using (15.6), compute bounds on second partial derivatives. In this example, the (x, y) coordinates lie in a rectangular lattice, which means that the the (x, y) components of all second partial derivatives are zero. The control points of the second hodographs for $\mathbf{P}_{ss}$, $\mathbf{P}_{st}$ and $\mathbf{P}_{tt}$ have

z-coordinates as follows:

$$\mathbf{P}_{ss}: \begin{bmatrix} -48 & 24 \\ -24 & -24 \\ -36 & -24 \\ -24 & -24 \end{bmatrix} \quad \mathbf{P}_{st}: \begin{bmatrix} 0 & -36 & 36 \\ -18 & 0 & 0 \\ 18 & 0 & 0 \end{bmatrix} \quad \mathbf{P}_{tt} = \begin{bmatrix} -24 & -12 & -36 & -12 \\ 12 & -12 & -12 & -12 \end{bmatrix}$$

from which $M_1 = 48$, $M_2 = 36$, and $M_3 = 36$. This is not a ruled surface, since $M_1, M_3 \neq 0$, so we proceed to use Case 3. Solve for $k = \sqrt{M_3/M_1} = \sqrt{3}/2$. Then

$$\delta_t = \sqrt{\frac{8\epsilon}{M_1 k^2 + 2M_2 k + M_3}} = \sqrt{\frac{8 * 0.001}{48 * (\sqrt{3}/2)^2 + 2 * 36 * \sqrt{3}/2 + 36}} = 0.0080677$$

and $\delta_s = k\delta_t = .0069861$. The number of rows and columns of triangles is then

$$n_s = \lceil \frac{1}{\delta_s} \rceil = 144; \qquad n_t = \lceil \frac{1}{\delta_t} \rceil = 124.$$

References

Wang, G.-Z. 1984. The subdivision method for finding the intersection between two Bezier curves or surfaces, *Zhejiang University Journal: Special Issue on Computational Geometry*, 108–119 (in Chinese).

Filip, D., Magedson,R. and Markot,R. 1986, Surface algorithms using bounds on derivatives, *Computer Aided Geometric Design* 3, 295–311.

Rockwood,A.,Heaton,K.,and Davis,T. 1989. Real-time rendering of trimmed surfaces. *SIGGRAPH Comput.Graph.* 23(3)(July 1989), 107-116.

Zheng, J. and Sederberg, T. 2000. Estimating tessellation parameter intervals for rational curves and surfaces. *ACM Transactions on Graphics* 19(1), 56–77.

15.5 C^n Surface Patches

As with Bézier curves, it is possible to piece together several individual Bézier surface patches to create a more complicated surface. Figure 15.6 shows a classic model from computer graphics, the Utah teapot. This teapot is modelled using 32 *bicubic* Bézier surface patches. For rendering, each patch is divided into a 9×9 grid of rectangles, and rectangles on the border of a patch are split into two triangles.

The conditions under which a pair of Bézier surface patches are C^n are analogous to the conditions for C^n continuity of a pair of Bézier curves. Two tensor-product surface patches are C^n if and only if every pair of adjacent control curves are C^n, as illustrated in Figure 15.7.

15.6 NURBS Surface

It is not too difficult to construct a Bézier patch that is C^n with a second Bézier patch. However, it is creating a network of Bézier patches that are C^n with each other is difficult for $n > 1$, because the two families of control curves (one in each parameter direction) must be C^n simultaneously.

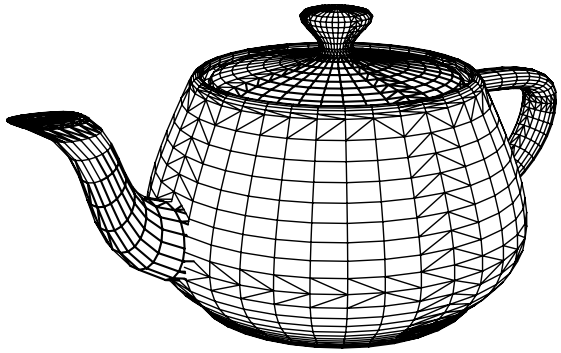
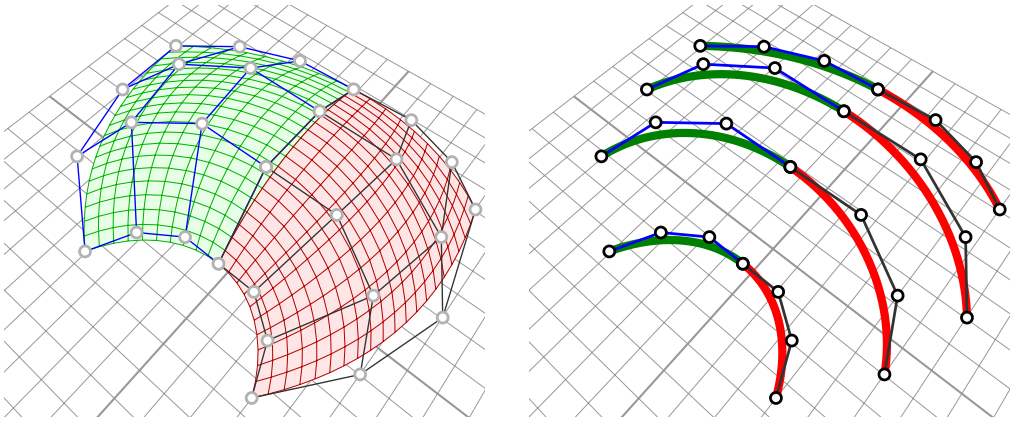


Figure 15.6: Teapot modeled using 32 bicubic Bézier surface patches



(a) Two bicubic Bézier surface patches. (b) Each matching pair of control curves is C^n.

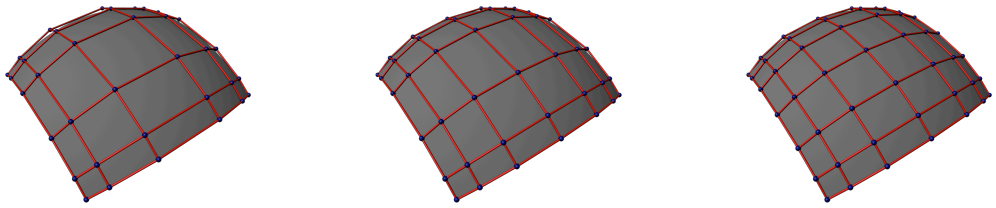
Figure 15.7: Two C^n bicubic Bézier surface patches.

This problem does not exist with B-Spline surfaces, because the control curves in both directions are automatically C^n.

A tensor-product NURBS surface is one for which the blending functions in (15.2) are B-Spline blending functions. In order to define a NURBS surface, we must therefore specify a knot vector for the s blending functions, and a second knot vector for the t blending functions.

The operations on NURBS curves such as knot insertion, finding the Bézier curves that comprise a NURBS curve, etc., extend directly to NURBS surface: We simply operate on each of the control curves. Thus, to insert a knot into the s knot vector for a NURBS surface, we insert the knot into each s-control curve of the NURBS surface. Figure 15.9 shows a bicubic NURBS surface which first undergoes a knot insertion in the t direction, and then in the s direction.

To split a NURBS surface into Bézier surfaces, we simply split each s-control curve into Bézier curves, and then split each resulting t-control curve into Bézier curves. The most efficient way to evaluate a NURBS surface is to first split it into Bézier surfaces, then evaluate the Bézier surfaces.

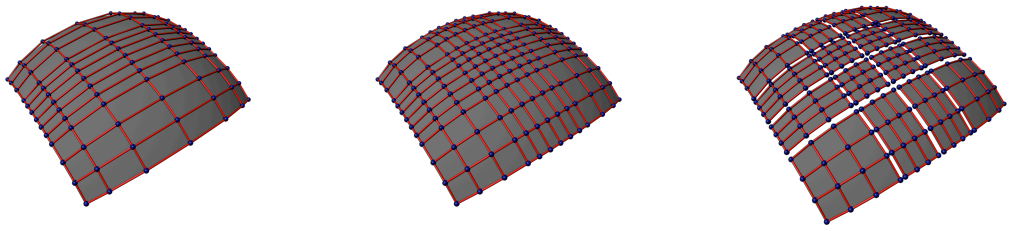


(a) A NURBS surface and its control grid.

(b) After inserting one knot in the t direction.

(c) After inserting one knot in the s direction.

Figure 15.8: Knot insertions into a NURBS surface.



(a) Inserting triple knots in the s direction.

(b) Inserting triple knots in the t direction.

(c) The Bézier surfaces, moved apart.

Figure 15.9: Splitting a NURBS surface into Bézier patches.

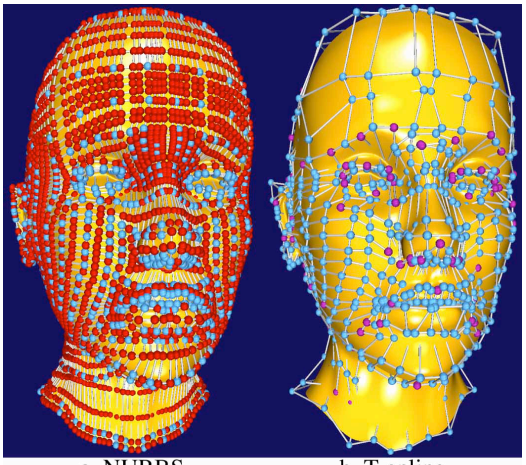
15.7 T-Splines

A serious weakness with NURBS surfaces is that NURBS control points must lie topologically in a rectangular grid. This means that typically, a large number of NURBS control points serve no purpose other than to satisfy topological constraints. They carry no significant geometric information. In Figure 15.10.a, all the red NURBS control points are, in this sense, superfluous.

T-splines are a generalization of NURBS surfaces that are capable of significantly reducing the number of superfluous control points. Figure 15.10.b shows a T-spline control grid which was obtained by eliminating the superfluous control points from the NURBS model. The main difference between a T-mesh (i.e., a T-spline control mesh) and a NURBS control mesh is that T-splines allow a row of control points to terminate. The final control point in a partial row is called a T-junction. The T-junctions are shown in purple in Figure 15.10.b.

Figure 15.11 shows another example in which the superfluous control points in a NURBS are removed to create a T-spline. The T-spline model is geometrically equivalent to the NURBS model, yet has only 1/3 as many control points.

Superfluous control points are a serious nuisance for designers, not merely because they require the designer to deal with more data, but also because they can introduce unwanted ripples in the surface as can be seen by comparing the forehead in the NURBS model in Figure 15.12.a with that of the T-spline model in Figure 15.12.b. Designers can waste dozens of hours on models such as this in tweaking the NURBS control points while attempting to remove unwanted ripples. Figure 15.10.a shows a NURBS head model. Figure 15.10 shows the respective NURBS and T-spline control meshes for the surfaces in Figure 15.12. Over 3/4 of the 4712 NURBS control points are superfluous and



a. NURBS b. T-spline

Figure 15.10: Head modeled (a) as a NURBS with 4712 control points and (b) as a T-spline with 1109 control points. The red NURBS control points are superfluous.

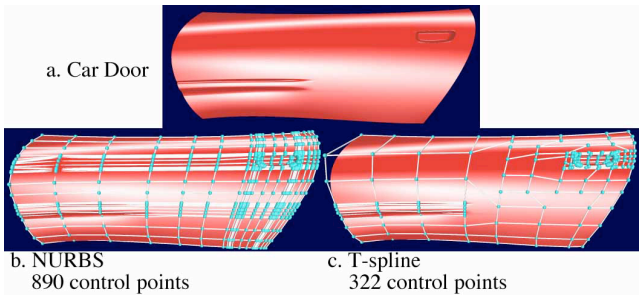
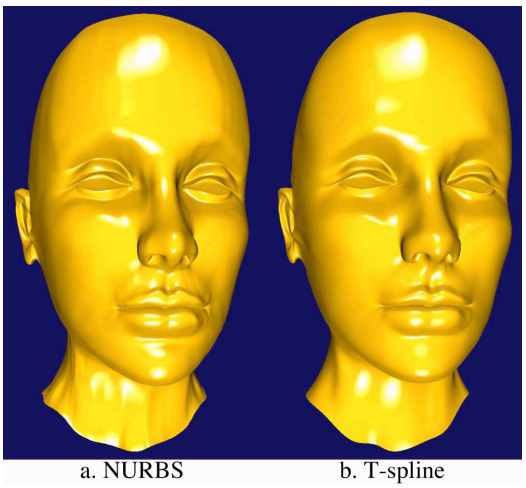


Figure 15.11: Car door modeled as a NURBS and as a T-spline.

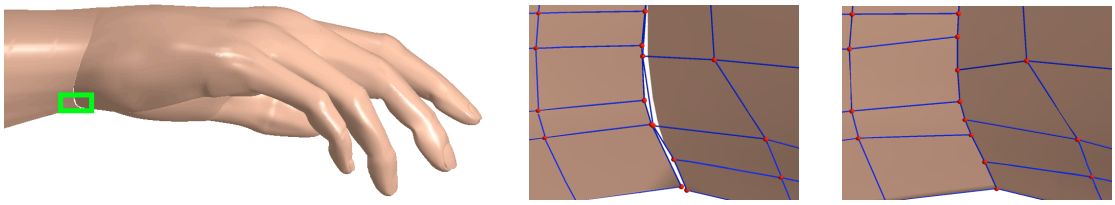


a. NURBS b. T-spline

Figure 15.12: NURBS head model, converted to a T-spline.

do not appear in the T-spline control mesh.

T-splines can be used to merge non-uniform B-spline surfaces that have different knot-vectors. Figure 15.13.a shows a hand model comprised of seven B-spline surfaces. The small rectangular area is blown up in Figure 15.13.b to magnify a hole where neighboring B-spline surfaces do not match exactly. The presence of such gaps places a burden on designers, who potentially must repair a widened gap whenever the model is deformed. Figure 15.13.c shows the model after being converted into a gap-free T-spline, thereby eliminating the need for repair.



(a) Model of a hand comprised of B-spline surfaces. (b) NURBS surfaces. (c) T-Spline.

Figure 15.13: A gap between two B-spline surfaces, fixed with a T-spline.

While the notion of T-splines extends to any degree, we restrict our discussion to cubic T-splines. Cubic T-splines are C^2 in the absence of multiple knots.

15.7.1 Equation of a T-Spline

A control grid for a T-spline surface is called a T-mesh. If a T-mesh forms a rectangular grid, the T-spline degenerates to a B-spline surface.

A T-mesh is basically a rectangular grid that allows T-junctions. The pre-image of each edge in a T-mesh is a line segment of constant s (which we will call an s-edge) or of constant t (which we will call a t-edge). A T-junction is a vertex shared by one s-edge and two t-edges, or by one t-edge and two s-edges.

Knot information for T-splines is expressed using knot intervals, non-negative numbers that indicate the difference between two knots. A knot interval is assigned to each edge in the T-mesh. Figure 15.14 shows the pre-image of a portion of a T-mesh in (s,t) parameter space; the d_i and e_i denote the knot intervals. Knot intervals are constrained by the relationship that the sum of all knot intervals along one side of any face must equal the sum of the knot intervals on the opposing side. For example, in Figure 15.14 on face F_1, $e_3 + e_4 = e_6 + e_7$, and on face F_2, $d_6 + d_7 = d_9$.

The knot intervals must obey the following requirements:

1. The sum of knot intervals on opposing edges of any face must be equal. Thus, for face $\mathbf{F}$ in Figure 15.14, $d_2 + d_6 = d_7$ and $e_6 + e_7 = e_8 + e_9$.

2. If a T-junction on one edge of a face can be connected to a T-junction on an opposing edge of the face (thereby splitting the face into two faces) without violating Rule 1, that edge must be included in the T-mesh.

It is possible to infer a local knot coordinate system from the knot intervals on a T-mesh. To impose a knot coordinate system, we first choose a control point whose pre-image will serve as the origin for the parameter domain $(s,t) = (0,0)$. For the example in Figure 15.15, we designate (s_0, t_0) to be the knot origin.

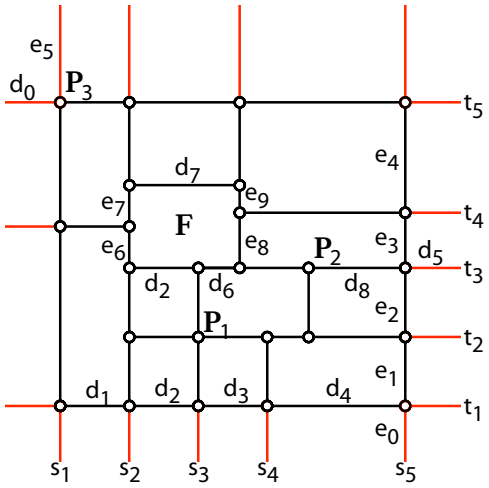


Figure 15.14: Pre-image of a T-mesh.

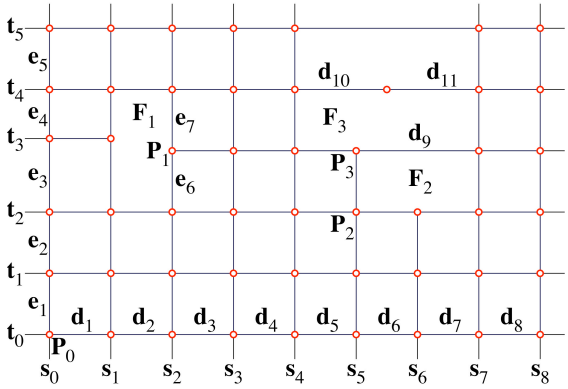


Figure 15.15: Pre-image of a T-mesh.

Once a knot origin is chosen, we can assign an s knot value to each vertical edge in the T-mesh topology, and a t knot value to each horizontal edge in the T-mesh topology. In Figure 15.15, those knot values are labeled s_i and t_i. Based on our choice of knot origin, we have $s_0 = t_0 = 0$, $s_1 = d_1$, $s_2 = d_1 + d_2$, $s_3 = d_1 + d_2 + d_3$, $t_1 = e_1$, $t_2 = e_1 + e_2$, and so forth. Likewise, each control point has knot coordinates. For example, the knot coordinates for $\mathbf{P}_0$ are $(0, 0)$, for $\mathbf{P}_1$ are $(s_2, t_2 + e_6)$, for $\mathbf{P}_2$ are (s_5, t_2), and for $\mathbf{P}_3$ are $(s_5, t_2 + e_6)$.

The knot coordinate system is used in writing an explicit formula for a T-spline surface:

$$\mathbf{P}(s,t) = \frac{\sum_{i=1}^{n} w_i \mathbf{P}_i B_i(s,t)}{\sum_{i=1}^{n} w_i B_i(s,t)}, \tag{15.7}$$

where $\mathbf{P}_i = (x_i, y_i, z_i, w_i)$ are control points in P^4 whose weights are w_i, and whose Cartesian coordinates are $\frac{1}{w_i}(x_i, y_i, z_i)$. Likewise, the Cartesian coordinates of points on the surface are given

by

$$\frac{\sum_{i=1}^{n}(x_i, y_i, z_i)B_i(s,t)}{\sum_{i=1}^{n} w_i B_i(s,t)}. \tag{15.8}$$

The blending functions in (15.7) are $B_i(s,t)$ and are given by

$$B_i(s,t) = N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](s)N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](t) \tag{15.9}$$

where $N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](s)$ is the cubic B-spline basis function associated with the knot vector

$$\mathbf{s}_i = [s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}] \tag{15.10}$$

and $N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](t)$ is associated with the knot vector

$$\mathbf{t}_i = [t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}]. \tag{15.11}$$

as illustrated in Figure 15.16. The designer is free to adjust the weights w_i to obtain additional

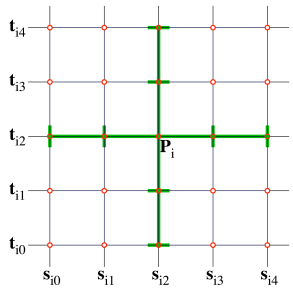


Figure 15.16: Knot lines for blending function $B_i(s,t)$.

shape control, as in rational B-splines. As we shall see in Section 15.7.2, weights also play a role in the local refinement algorithm.

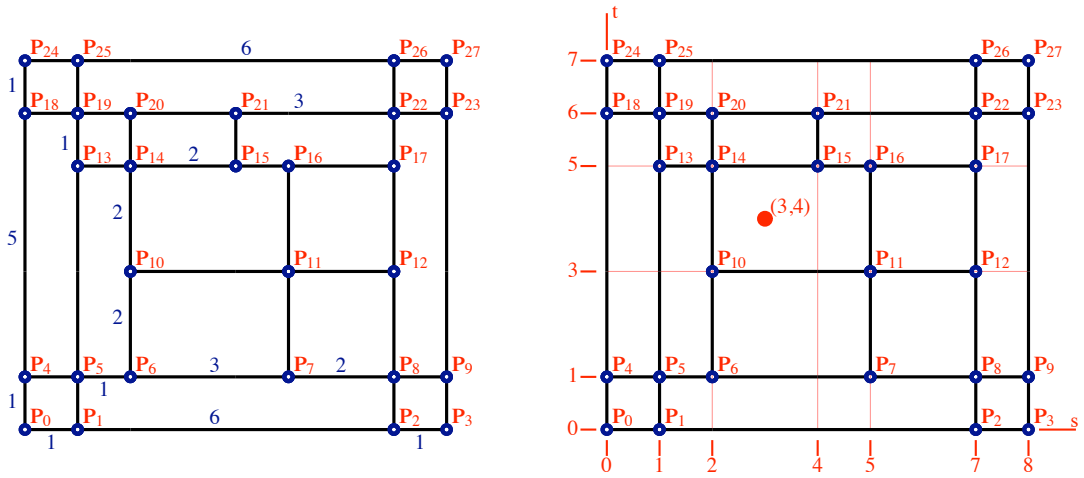
The T-spline equation is very similar to the equation for a tensor-product rational B-spline surface. The difference between the T-spline equation and a B-spline equation is in how the knot vectors $\mathbf{s}_i$ and $\mathbf{t}_i$ are determined for each blending function $B_i(s,t)$. Knot vectors $\mathbf{s}_i$ (15.10) and $\mathbf{t}_i$ (15.11) are *inferred from the T-mesh* neighborhood of $\mathbf{P}_i$. Since we will refer to the rule whereby the knot vectors are inferred, we formally state it as

Rule 1. Knot vectors $\mathbf{s}_i$ (15.10) and $\mathbf{t}_i$ (15.11) for the blending function of $\mathbf{P}_i$ are determined as follows. (s_{i2}, t_{i2}) are the knot coordinates of $\mathbf{P}_i$. Consider a ray in parameter space $\mathbf{R}(\alpha) = (s_{i2} + \alpha, t_{i2})$. Then s_{i3} and s_{i4} are the s coordinates of the first two s-edges intersected by the ray (not including the initial (s_{i2}, t_{i2})). By s-edge, we mean a vertical line segment of constant s. The other knots in $\mathbf{s}_i$ and $\mathbf{t}_i$ are found in like manner.

We illustrate Rule 1 by a few examples. The knot vectors for $\mathbf{P}_1$ in Figure 15.15 are $\mathbf{s}_1 = [s_0, s_1, s_2, s_3, s_4]$ and $\mathbf{t}_1 = [t_1, t_2, t_2 + e_6, t_4, t_5]$. For $\mathbf{P}_2$, $\mathbf{s}_2 = [s_3, s_4, s_5, s_6, s_7]$ and $\mathbf{t}_2 = [t_0, t_1, t_2, t_2 + e_6, t_4]$. For $\mathbf{P}_3$, $\mathbf{s}_3 = [s_3, s_4, s_5, s_7, s_8]$ and $\mathbf{t}_3 = [t_1, t_2, t_2 + e_6, t_4, t_5]$. Once these knot vectors are determined for each blending function, the T-spline is defined using (15.7) and (15.9).

Numerical Example

Figure 15.17.a shows a T-mesh in knot interval form. While it is possible to work entirely in knot interval form, our discussion is simplified if we convert to knot vector form. We do this by assigning the lower-left corner of the domain to have parameter coordinates $(s,t) = (0,0)$. Then, the knot lines are determined from the knot intervals, as shown in Figure 15.17.b.



(a) Model of a hand comprised of B-spline surfaces.

(b) NURBS surfaces.

Figure 15.17: Example T-Mesh.

From (15.7), the equation of the T-Spline shown in Figure 15.17.b is

$$\mathbf{P}(s,t) = \frac{\sum_{i=0}^{27} w_i \mathbf{P}_i B_i(s,t)}{\sum_{i=1}^{n} w_i B_i(s,t)}, \tag{15.12}$$

We now examine the equation $\mathbf{P}(3,4)$ for the T-Spline shown in Figure 15.17.b.

Recall from (15.9) that

$$B_i(3,4) = N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](3) N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](4) \tag{15.13}$$

For example, we have

$$B_{10}(3,4) = N[0,1,2,5,7](3) N[0,1,3,5,6](4), \quad B_{15}(3,4) = N[1,2,4,5,7](3) N[1,3,5,6,7](4),$$

$$B_0(3,4) = N[s_0,s_1,0,1,7](3) N[t_0,t_1,0,1,6](4), \quad B_{27}(3,4) = N[1,7,8,s_2,s_3](3) N[1,6,7,t_2,t_3](4).$$

We now show how to compute $B_{10}(3,4)$. The basis function $N[0,1,2,4,5](s)$ can be written as an explicit B-Spline curve with seven control points whose polar labels (with y-coordinates) are $f(s_0,s_1,0) = 0$, $f(s_1,0,1) = 0$, $f(0,1,2) = 0$, $f(1,2,4) = 1$, $f(2,4,5) = 0$, $f(4,5,s_2) = 0$, $f(5,s_2,s_3) = 0$. The knots s_0, s_1, s_2, s_3 are any real numbers that satisfy $s_0 \le s_1 \le 0$ and $5 \le s_2 \le s_3$. The value of $N[0,1,2,4,5](3)$ is the y-coordinate of the point whose polar label is $f(3,3,3)$. We can compute this as follows:

$$f(1,2,3) = \frac{f(0,1,2) + 3f(1,2,4)}{4} = \frac{3}{4}; \quad f(2,3,4) = \frac{2f(1,2,4) + 2f(2,4,5)}{4} = \frac{2}{4}$$

$$f(3,4,5) = \frac{(s_2-3)f(2,4,5)+f(4,5,s_2)}{s_2-2} = 0; \quad f(2,3,3) = \frac{f(1,2,3)+2f(2,3,4)}{3} = \frac{7}{12}$$

$$f(3,3,4) = \frac{2f(2,3,4)+f(3,4,5)}{3} = \frac{1}{3}; \quad f(3,3,3) = \frac{f(2,3,3)+f(3,3,4)}{2} = \frac{11}{24} = N[0,1,2,4,5](3).$$

Likewise, we can compute $N[0,1,3,5,6](4) = \frac{23}{60}$. Thus, $B_{10}(3,4) = \frac{253}{1440}$.

There are several blending functions that are zero at $s = 3$, $t = 4$. For example,

$$B_4(3,4) = N[s_0, s_1, 0, 1, 2](3)N[t_1, 0, 1, 3, 5](4) = 0 * N[t_1, 0, 1, 3, 5](4) = 0.$$

Likewise,

$$B_2(3,4) = B_9(3,4) = B_{17}(3,4) = B_{18}(3,4) = B_{23}(3,4) = B_{25}(3,4) = B_{26}(3,4) = 0.$$

15.7.2 T-spline Local Refinement

The next three sections discuss an algorithm for local refinement of T-splines. Blending function refinement plays an important role in this algorithm, and is reviewed in Section 15.7.3. The notion of T-spline spaces is introduced in Section 15.7.4. This concept is used in the local refinement algorithm in Section 15.7.5.

15.7.3 Blending Function Refinement

If $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$ is a knot vector and $\tilde{\mathbf{s}}$ is a knot vector with m knots with $\mathbf{s}$ a subsequence of $\tilde{\mathbf{s}}$, then $N[s_0, s_1, s_2, s_3, s_4](s)$ can be written as a linear combination of the $m-4$ B-spline basis functions defined over the substrings of length 5 in $\tilde{\mathbf{s}}$.

We now present all basis function refinement equations for the case $m = 6$. Equations for $m > 6$ can be found by repeated application of these equations.

If $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$, $N(s) = N[s_0, s_1, s_2, s_3, s_4](s)$, and $\tilde{\mathbf{s}} = [s_0, k, s_1, s_2, s_3, s_4]$ then

$$N(s) = c_0 N[s_0, k, s_1, s_2, s_3](s) + d_0 N[k, s_1, s_2, s_3, s_4](s) \tag{15.14}$$

where $c_0 = \frac{k-s_0}{s_3-s_0}$ and $d_0 = 1$. If $\tilde{\mathbf{s}} = [s_0, s_1, k, s_2, s_3, s_4]$,

$$N(s) = c_1 N[s_0, s_1, k, s_2, s_3](s) + d_1 N[s_1, k, s_2, s_3, s_4](s) \tag{15.15}$$

where $c_1 = \frac{k-s_0}{s_3-s_0}$ and $d_1 = \frac{s_4-k}{s_4-s_1}$. If $\tilde{\mathbf{s}} = [s_0, s_1, s_2, k, s_3, s_4]$,

$$N(s) = c_2 N[s_0, s_1, s_2, k, s_3](s) + d_2 N[s_1, s_2, k, s_3, s_4](s) \tag{15.16}$$

where $c_2 = \frac{k-s_0}{s_3-s_0}$ and $d_2 = \frac{s_4-k}{s_4-s_1}$. If $\tilde{\mathbf{s}} = [s_0, s_1, s_2, s_3, k, s_4]$,

$$N(s) = c_3 N[s_0, s_1, s_2, s_3, k](s) + d_3 N[s_1, s_2, s_3, k, s_4](s) \tag{15.17}$$

where $c_3 = 1$ and $d_3 = \frac{s_4-k}{s_4-s_1}$. If $k \le s_0$ or $k \ge s_4$, $N(s)$ does not change.

A T-spline function $B(s,t)$ can undergo knot insertion in either s or t, thereby splitting it into two scaled blending functions that sum to the initial one. Further insertion into these resultant scaled blending functions yields a set of scaled blending functions that sum to the original. For example, Figure 15.18.a shows the knot vectors for a T-spline blending function B_1, and Figure 15.18.b shows a refinement of the knot vectors in Figure 15.18.a. By appropriate application of (15.14)—(15.17), we can obtain

$$B_1(s,t) = c_1^1 \tilde{B}_1(s,t) + c_1^2 \tilde{B}_2(s,t) + c_1^3 \tilde{B}_3(s,t) + c_1^4 \tilde{B}_4(s,t). \tag{15.18}$$

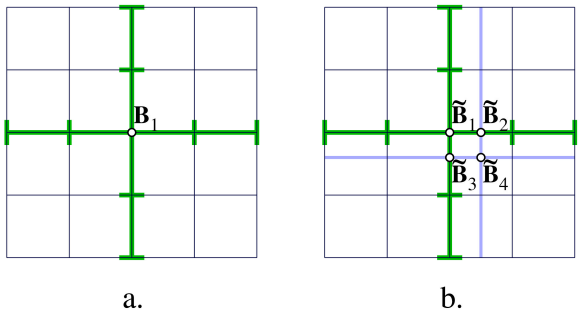


Figure 15.18: Sample Refinement of $B_1(s,t)$.

15.7.4 T-spline Spaces

We define a T-spline space to be the set of all T-splines that have the same T-mesh topology, knot intervals, and knot coordinate system. Thus, a T-spline space can be represented by the diagram of a pre-image of a T-mesh such as in Figure 15.14. Since all T-splines in a given T-spline space have the same pre-image, it is proper to speak of the pre-image of a T-spline space. A T-spline space S_1 is said to be a subspace of S_2 (denoted $S_1 \subset S_2$) if local refinement of a T-spline in S_1 will produce a T-spline in S_2 (discussed in Section 15.7.5). If T_1 is a T-spline, then $T_1 \in S_1$ means that T_1 has a control grid whose topology and knot intervals are specified by S_1.

Figure 15.19 illustrates a nested sequence of T-spline spaces, that is, $S_1 \subset S_2 \subset S_3 \subset \cdots \subset S_n$.

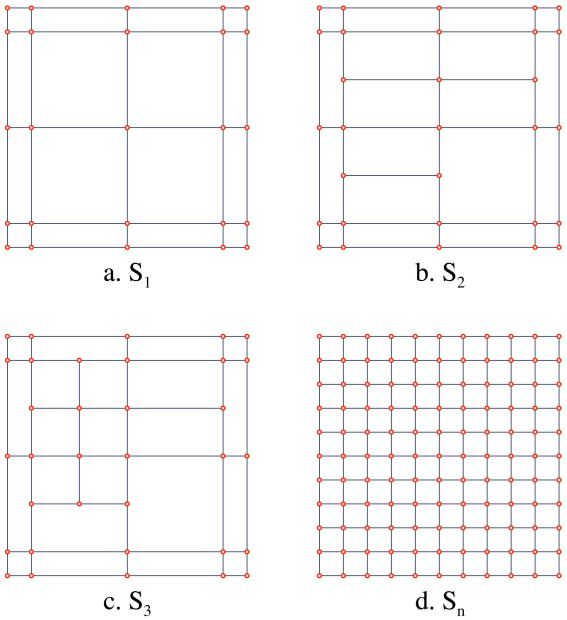


Figure 15.19: Nested sequence of T-spline spaces.

Given a T-spline $\mathbf{P}(s,t) \in S_1$, denote by $\mathbf{P}$ the column vector of control points for $\mathbf{P}(s,t)$, and

given a second T-spline $\tilde{\mathbf{P}}(s,t) \in S_2$, such that $\mathbf{P}(s,t) \equiv \tilde{\mathbf{P}}(s,t)$. Denote by $\tilde{\mathbf{P}}$ the column vector of control points for $\tilde{\mathbf{P}}(s,t)$. There exists a linear transformation that maps $\mathbf{P}$ into $\tilde{\mathbf{P}}$. We can denote the linear transformation

$$M_{1,2}\mathbf{P} = \tilde{\mathbf{P}}. \tag{15.19}$$

The matrix $M_{1,2}$ is found as follows.

$\mathbf{P}(s,t)$ is given by (15.7), and

$$\tilde{\mathbf{P}}(s,t) = \sum_{j=1}^{\tilde{n}} \tilde{\mathbf{P}}_j \tilde{B}_j(s,t) \tag{15.20}$$

Since $S_1 \subset S_2$, each $B_i(s,t)$ can be written as a linear combination of the $\tilde{B}_j(s,t)$:

$$B_i(s,t) = \sum_{j=1}^{\tilde{n}} c_i^j \tilde{B}_j(s,t). \tag{15.21}$$

We require that

$$\mathbf{P}(s,t) \equiv \tilde{\mathbf{P}}(s,t). \tag{15.22}$$

This is satisfied if

$$\tilde{\mathbf{P}}_j = \sum_{i=1}^{n} c_i^j \mathbf{P}_i. \tag{15.23}$$

Thus, the element at row j and column i of $M_{1,2}$ in (15.19) is c_i^j. In this manner, it is possible to find transformation matrices $M_{i,j}$ that maps any T-spline in S_i to an equivalent T-spline in S_j, assuming $S_i \subset S_j$.

The definition of a T-spline subspace $S_i \subset S_j$ means more than simply that the preimage of S_j has all of the control points that the preimage of S_i has.

15.7.5 Local Refinement Algorithm

T-spline local refinement means to insert one or more control points into a T-mesh without changing the shape of the T-spline surface. This procedure can also be called local knot insertion, since the addition of control points to a T-mesh must be accompanied by knots inserted into neighboring blending functions.

The refinement algorithm we now present has two phases: the topology phase, and the geometry phase. The topology phase identifies which (if any) control points must be inserted in addition to the ones requested. Once all required new control points are identified, the Cartesian coordinates and weights for the refined T-mesh are computed using the linear transformation presented in Section 15.7.4. We now explain the topology phase of the algorithm.

An important key to understanding this discussion is to keep in mind how in a T-spline, the blending functions and T-mesh are tightly coupled: To every control point there corresponds a blending function, and each blending function's knot vectors are defined by Rule 1. In our discussion, we temporarily decouple the blending functions from the T-mesh. This means that during the flow of the algorithm, we temporarily permit the existence of blending functions that violate Rule 1, and control points to which no blending functions are attached.

Our discussion distinguishes three possible violations that can occur during the course of the refinement algorithm:

- **Violation 1** A blending function is missing a knot dictated by Rule 1 for the current T-mesh.

- **Violation 2** A blending function has a knot that is not dictated by Rule 1 for the current T-mesh.

- **Violation 3** A control point has no blending function associated with it.

If no violations exist, the T-spline is valid. If violations do exist, the algorithm resolves them one by one until no further violations exist. Then a valid superspace has been found.

The topology phase of our local refinement algorithm consists of these steps:

1. Insert all desired control points into the T-mesh.

2. If any blending function is guilty of Violation 1, perform the necessary knot insertions into that blending function.

3. If any blending function is guilty of Violation 2, add an appropriate control point into the T-mesh.

4. Repeat Steps 2 and 3 until there are no more violations.

Resolving all cases of Violation 1 and 2 will automatically resolve all cases of Violation 3.

We illustrate the algorithm with an example. Figure 15.20.a shows an initial T-mesh into which we wish to insert one control point, $\mathbf{P}_2$. Because the T-mesh in Figure 15.20.a is valid, there are no violations. But if we simply insert $\mathbf{P}_2$ into the T-mesh (Figure 15.20.b) *without changing any of the blending functions*, we introduce several violations. Since $\mathbf{P}_2$ has knot coordinates (s_3, t_2), four blending functions become guilty of Violation 1: those centered at (s_1, t_2), (s_2, t_2), (s_4, t_2), and (s_5, t_2). To resolve these violations, we must insert a knot at s_3 into each of those blending functions, as discussed in Section 15.7.3. The blending function centered at (s_2, t_2) is $N[s_0, s_1, s_2, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$. Inserting a knot $s = s_3$ into the s knot vector of this blending function splits it into two scaled blending functions: $c_2 N[s_0, s_1, s_2, s_3, s_4](s)N[t_0, t_1, t_2, t_3, t_4](t)$ (Figure 15.20.c) and $d_2 N[s_1, s_2, s_3, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$ (Figure 15.20.d) as given in (15.16).

The blending function $c_2 N[s_0, s_1, s_2, s_3, s_4](s)N[t_0, t_1, t_2, t_3, t_4](t)$ in Figure 15.20.c satisfies Rule 1. Likewise, the refinements of the blending functions centered at (s_1, t_2), (s_4, t_2), and (s_5, t_2) all satisfy Rule 1. However, the t knot vector of blending function $d_2 N[s_1, s_2, s_3, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$ shown in Figure 15.20.d is guilty of Violation 2 because the blending function's t knot vector is $[t_0, t_1, t_2, t_3, t_4]$, but Rule 1 does not call for a knot at t_3. This problem cannot be remedied by refining this blending function; we must add an additional control point into the T-mesh.

The needed control point is $\mathbf{P}_3$ in Figure 15.20.e. Inserting that control point fixes the case of Violation 2, but it creates a new case of Violation 1. As shown in Figure 15.20.f, the blending function centered at (s_2, t_3) has an s knot vector that does not include s_3 as required by Rule 1. Inserting s_3 into that knot vector fixes the problem, and there are no further violations of Rule 1.

This algorithm is always guaranteed to terminate, because the only blending function refinements and control point insertions must involve knot values that initially exist in the T-mesh, or that were added in Step 1. In the worst case, the algorithm would extend all partial rows of control points to cross the entire surface. In practice, the algorithm typically requires few if any additional new control points beyond the ones the user wants to insert.

15.7.6 Converting a T-spline into a B-spline surface

This refinement algorithm makes it easy to convert a T-spline $\in S_1$ into an equivalent B-spline surface $\in S_n$: simply compute the transformation matrix $M_{1,n}$ as discussed in Section 15.7.4.

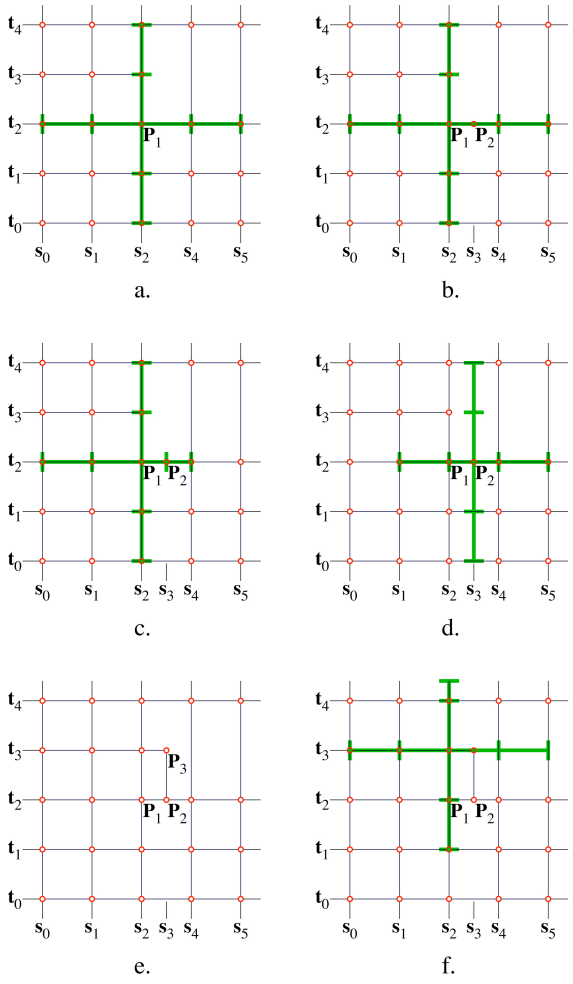


Figure 15.20: Local refinement example.

A *standard* T-spline is one for which, if all weights $w_i = 1$, then $\sum_{i=1}^{n} w_i B_i(s,t) \equiv \sum_{i=1}^{n} B_i(s,t) \equiv 1$. This means that the denominator in (15.7) is identically equal to one; hence, the blending functions provide a partition of unity and the T-spline is polynomial. Thus, an algebraic statement of necessary and sufficient conditions for a T-spline to be standard is each row of $M_{1,n}$ sums to 1.

The insertion algorithm can produce a surprising result: a T-spline for which not all $w_i = 1$ but yet $\sum_{i=1}^{n} w_i B_i(s,t) \equiv 1$. This is called a semi-standard T-spline. Figure 15.21 shows two simple examples of semi-standard T-splines. The integers (1 and 2) next to some edges are knot intervals. To verify that these T-splines are semi-standard, convert them into B-spline surfaces; the control

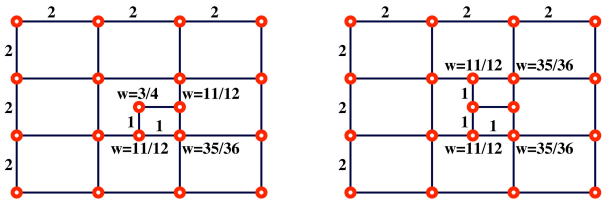


Figure 15.21: Semi-standard T-splines.

point weights will all be one.

The notion of T-spline spaces in Section 15.7.4 enables a more precise definition of semi-standard and non-standard T-splines. A semi-standard T-spline space S is one for which there exists some elements of S for which $\sum_{i=1}^{n} w_i B_i(s,t) \equiv 1$, and not all $w_i = 1$. A non-standard T-spline space is one for which no elements exist for which $\sum_{i=1}^{n} w_i B_i(s,t) \equiv 1$. These definitions are more precise because they allow for the notion of a rational T-spline (weights not all = 1) that is either standard, semi-standard, or non-standard. The distinction is made based on which type of T-spline space it belongs to.

15.8 Efficient Computation of Points and Tangents on a Bézier surface patch.

This section presents an algorithm for computing points and tangents on a tensor-product rational Bézier surface patch that has $O(n^2)$ time complexity.

Define a rational Bézier curve in $\mathbf{R}^3$ with the notation

$$\mathbf{p}(t) = \Pi\left(\mathbf{P}(t)\right) \tag{15.24}$$

with

$$\mathbf{P}(t) = (\mathbf{P}_x(t), \mathbf{P}_y(t), \mathbf{P}_z(t), \mathbf{P}_w(t)) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t) \tag{15.25}$$

where $\mathbf{P}_i = w_i(x_i, y_i, z_i, 1)$ and the projection operator Π is defined $\Pi(x,y,z,w) = (x/w, y/w, z/w)$. In this section, we will use upper case bold-face variables to denote four-tuples (homogeneous points) and lower case bold-face for triples (points in R^3).

The point and tangent of this curve can be found using the familiar construction

$$\mathbf{P}(t) = (1-t)\mathbf{Q}(t) + t\mathbf{R}(t) \tag{15.26}$$

with

$$\mathbf{Q}(t) = \sum_{i=0}^{n-1} \mathbf{P}_i B_i^{n-1}(t) \tag{15.27}$$

and

$$\mathbf{R}(t) = \sum_{i=1}^{n} \mathbf{P}_i B_{i-1}^{n-1}(t) \tag{15.28}$$

where line $\mathbf{q}(t)$—$\mathbf{r}(t) \equiv \Pi(\mathbf{Q}(t))$—$\Pi(\mathbf{R}(t))$ is tangent to the curve, as seen in Figure 16.1. As a sidenote, the correct magnitude of the derivative of $\mathbf{p}(t)$ is given by

$$\frac{d\mathbf{p}(t)}{dt} = n\frac{\mathbf{R}_w(t)\mathbf{Q}_w(t)}{((1-t)\mathbf{Q}_w(t) + t\mathbf{R}_w(t))^2}\left[\mathbf{r}(t) - \mathbf{q}(t)\right] \tag{15.29}$$

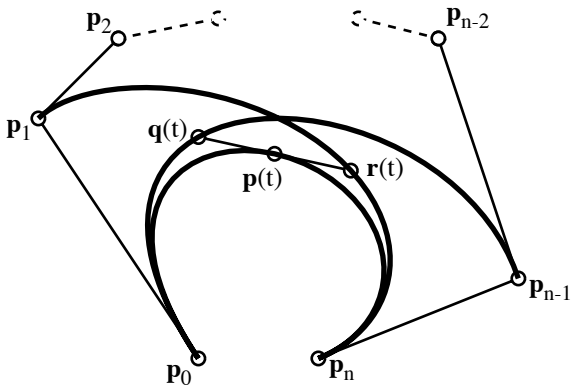


Figure 15.22: Curve example

The values $\mathbf{Q}(t)$ and $\mathbf{R}(t)$ can be found using the modified Horner's algorithm for Bernstein polynomials, involving a pseudo–basis conversion

$$\frac{\mathbf{Q}(t)}{(1-t)^{n-1}} = \hat{\mathbf{Q}}(u) = \sum_{i=0}^{n-1} \hat{\mathbf{Q}}_i u^i \tag{15.30}$$

where $u = \frac{t}{1-t}$ and $\hat{\mathbf{Q}}_i = \binom{n-1}{i}\mathbf{P}_i$, $i = 0, 1, \ldots, n-1$. Assuming the curve is to be evaluated several times, we can ignore the expense of precomputing the $\hat{\mathbf{Q}}_i$, and the nested multiplication

$$\hat{\mathbf{Q}}(u) = [\cdots[[\hat{\mathbf{Q}}_{n-1}u + \hat{\mathbf{Q}}_{n-2}]u + \hat{\mathbf{Q}}_{n-3}]u + \ldots \hat{\mathbf{Q}}_1]u + \hat{\mathbf{Q}}_0 \tag{15.31}$$

can be performed with $n-1$ multiplies and adds for each of the four x, y, z, w coordinates. It is not necessary to post-multiply by $(1-t)^{n-1}$, since

$$\Pi(\mathbf{Q}(t)) = \Pi\left((1-t)^{n-1}\hat{\mathbf{Q}}(u)\right) = \Pi\left(\hat{\mathbf{Q}}(t)\right). \tag{15.32}$$

Therefore, the point $\mathbf{P}(t)$ and its tangent direction can be computed with roughly $2n$ multiplies and adds for each of the four x, y, z, w coordinates.

This method has problems near $t = 1$, so it is best for $.5 \le t \le 1$ to use the form

$$\frac{\mathbf{Q}(t)}{t^{n-1}} = \sum_{i=0}^{n-1} \hat{\mathbf{Q}}_{n-i-1} u^i \tag{15.33}$$

with $u = \frac{1-t}{t}$.

A tensor product rational Bézier surface patch is defined

$$\mathbf{p}(s,t) = \Pi\left(\mathbf{P}(s,t)\right) \tag{15.34}$$

where

$$\mathbf{P}(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{P}_{ij} B_i^m(s) B_j^n(t). \tag{15.35}$$

We can represent the surface $\mathbf{p}(s,t)$ using the following construction:

$$\mathbf{P}(s,t) = (1-s)(1-t)\mathbf{P}^{00}(s,t) + s(1-t)\mathbf{P}^{10}(s,t) + (1-s)t\mathbf{P}^{01}(s,t) + st\mathbf{P}^{11}(s,t) \tag{15.36}$$

where

$$\mathbf{P}^{00}(s,t) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} \mathbf{P}_{ij} B_i^{m-1}(s) B_j^{n-1}(t), \tag{15.37}$$

$$\mathbf{P}^{10}(s,t) = \sum_{i=1}^{m}\sum_{j=0}^{n-1} \mathbf{P}_{ij} B_{i-1}^{m-1}(s) B_j^{n-1}(t), \tag{15.38}$$

$$\mathbf{P}^{01}(s,t) = \sum_{i=0}^{m-1}\sum_{j=1}^{n} \mathbf{P}_{ij} B_i^{m-1}(s) B_{j-1}^{n-1}(t), \tag{15.39}$$

$$\mathbf{P}^{11}(s,t) = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{P}_{ij} B_{i-1}^{m-1}(s) B_{j-1}^{n-1}(t). \tag{15.40}$$

The tangent vector $\mathbf{p}_s(s,t)$ is parallel with the line

$$\Pi\left((1-t)\mathbf{P}^{00}(s,t) + t\mathbf{P}^{01}(s,t)\right) \text{—} \Pi\left((1-t)\mathbf{P}^{10}(s,t) + t\mathbf{P}^{11}(s,t)\right) \tag{15.41}$$

and the tangent vector $\mathbf{p}_t(s,t)$ is parallel with

$$\Pi\left((1-s)\mathbf{P}^{00}(s,t) + s\mathbf{P}^{10}(s,t)\right) \text{—} \Pi\left((1-s)\mathbf{P}^{01}(s,t) + s\mathbf{P}^{11}(s,t)\right). \tag{15.42}$$

The Horner algorithm for a tensor product surface emerges by defining

$$\frac{\mathbf{P}^{kl}(s,t)}{(1-s)^{m-1}(1-t)^{n-1}} = \hat{\mathbf{P}}^{kl}(u,v) = \sum_{i=k}^{m+k-1}\sum_{j=l}^{n+l-1} \hat{\mathbf{P}}_{ij}^{kl} u^i v^j; \qquad k,l = 0,1 \tag{15.43}$$

where $u = \frac{s}{1-s}$, $v = \frac{t}{1-t}$, and $\hat{\mathbf{P}}_{ij}^{kl} = \binom{m-1}{i-k}\binom{n-1}{j-l}\mathbf{P}_{ij}$. The n rows of these four bivariate polynomials can each be evaluated using $m-1$ multiplies and adds per x, y, z, w component, and the final evaluation in t costs $n-1$ multiplies and adds per x, y, z, w component.

Thus, if $m = n$, the four surfaces $\mathbf{P}^{00}(s,t)$, $\mathbf{P}^{01}(s,t)$, $\mathbf{P}^{10}(s,t)$, and $\mathbf{P}^{11}(s,t)$ can each be evaluated using n^2-1 multiplies and n^2-1 adds for each of the four x, y, z, w components, a total of $16n^2-16$ multiplies and $16n^2-16$ adds.

If one wishes to compute a grid of points on this surface which are evenly spaced in parameter space, the four surfaces $\mathbf{P}^{00}(s,t)$, $\mathbf{P}^{01}(s,t)$, $\mathbf{P}^{10}(s,t)$, and $\mathbf{P}^{11}(s,t)$ can each be evaluated even more quickly using forward differencing.

15.9 Curvature at the Corner of a Bézier Surface Patch

We have seen in Section **??** how the curvature of a degree n rational Bézier curve with control points $\mathbf{P}_0, \mathbf{P}_1, \cdots, \mathbf{P}_n$ and weights $\omega_0, \omega_1, \cdots, \omega_n$, at the starting point $\mathbf{P}_0$ is given by

$$k = \frac{n-1}{n} \frac{\omega_0 \omega_2}{\omega_1^2} \frac{h}{a^2} \tag{15.44}$$

where a is the length of edge $\mathbf{P}_0\mathbf{P}_1$, and h is the distance of $\mathbf{P}_2$ to the tangent spanned by $\mathbf{P}_0$ and $\mathbf{P}_1$ (see Figure 15.23). This formula is more intuitive than the one given by classic differential geometry, and is also easier to compute, especially in the rational case.

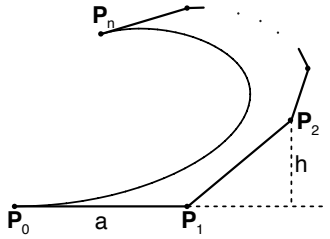


Figure 15.23: Curvature of a Bézier curve.

This section derives similar formulae for Gaussian and mean curvatures of rational Bézier patches. The derived formulae are expressed in terms of simple geometric quantities of the control mesh.

15.9.1 Curvatures of tensor-product rational Bézier surfaces

Suppose a degree $n \times m$ rational Bézier patch is defined by

$$\mathbf{r}(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} \omega_{ij} \mathbf{P}_{ij} B_i^n(u) B_j^m(v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} \omega_{ij} B_i^n(u) B_j^m(v)} \tag{15.45}$$

where $B_i^k(t) = \binom{k}{i}(1-t)^{k-i}t^i$ are Bernstein polynomials; $\mathbf{P}_{ij}$ are the control points, forming a control mesh; and w_{ij} are the weights. When all the weights are the same, the patch reduces to a polynomial surface. We derive the curvature formulae at the bottom-left corner $(u,v) = (0,0)$.

By differential geometry,Gaussian curvature K_g and mean curvature K_m can be computed by the following formulae

$$K_g = \frac{LN - M^2}{EG - F^2}, \tag{15.46}$$

$$K_m = \frac{1}{2} \frac{LG - 2MF + NE}{EG - F^2} \tag{15.47}$$

where E, F, G are the coefficients of the first fundamental form, i.e.,

$$E = \mathbf{r}_u \cdot \mathbf{r}_u, \quad F = \mathbf{r}_u \cdot \mathbf{r}_v, \quad G = \mathbf{r}_v \cdot \mathbf{r}_v, \tag{15.48}$$

L, M, N are the coefficients of the second fundamental form, i.e.,

$$L = \mathbf{r}_{uu} \cdot \mathbf{n}, \quad M = \mathbf{r}_{uv} \cdot \mathbf{n}, \quad N = \mathbf{r}_{vv} \cdot \mathbf{n}, \tag{15.49}$$

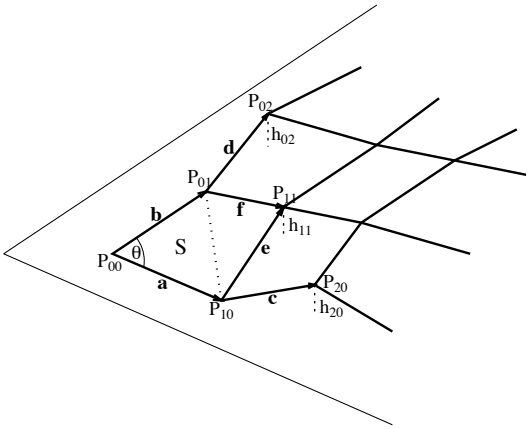


Figure 15.24: Part of a rectangular mesh.

and $\mathbf{n} = \mathbf{r}_u \times \mathbf{r}_v / \|\mathbf{r}_u \times \mathbf{r}_v\|$ is the unit normal vector.

For the rational Bézier patch (15.45), we introduce some notations as follows (see Figure 15.24):

$$\begin{array}{lll} \mathbf{a} = \mathbf{P}_{10} - \mathbf{P}_{00}, & \mathbf{b} = \mathbf{P}_{01} - \mathbf{P}_{00}, & \mathbf{c} = \mathbf{P}_{20} - \mathbf{P}_{10}, \\ \mathbf{d} = \mathbf{P}_{02} - \mathbf{P}_{01}, & \mathbf{e} = \mathbf{P}_{11} - \mathbf{P}_{10}, & \mathbf{f} = \mathbf{P}_{11} - \mathbf{P}_{01}. \end{array} \tag{15.50}$$

Furthermore, let S denote the area of the triangle $\mathbf{P}_{00}\mathbf{P}_{10}\mathbf{P}_{01}$, θ be the angle between vectors $\mathbf{P}_{00}\mathbf{P}_{10}$ and $\mathbf{P}_{00}\mathbf{P}_{01}$, and h_{ij} be the signed distance from $\mathbf{P}_{ij}$ to the plane spanned by $\mathbf{P}_{00}, \mathbf{P}_{10}$ and $\mathbf{P}_{01}$. If $\mathbf{P}_{ij}$ lies on the side of the plane $\mathbf{P}_{00}\mathbf{P}_{10}\mathbf{P}_{01}$ with the direction $\mathbf{P}_{00}\mathbf{P}_{10} \times \mathbf{P}_{00}\mathbf{P}_{01}$, h_{ij} is positive. Otherwise, h_{ij} is negative.

It is easy to check that at $(u, v) = (0, 0)$, we have

$$\mathbf{r}_u(0,0) = n\frac{\omega_{10}}{\omega_{00}}\mathbf{a}, \qquad \mathbf{r}_v(0,0) = m\frac{\omega_{01}}{\omega_{00}}\mathbf{b}. \tag{15.51}$$

For notational simplicity, in the following where there is no ambiguity, we omit the parameter values $(0, 0)$. Thus

$$E = n^2\left(\frac{\omega_{10}}{\omega_{00}}\right)^2 \mathbf{a}\cdot\mathbf{a}, \quad F = nm\left(\frac{\omega_{10}\omega_{01}}{\omega_{00}^2}\right)\mathbf{a}\cdot\mathbf{b}, \quad G = m^2\left(\frac{\omega_{01}}{\omega_{00}}\right)^2 \mathbf{b}\cdot\mathbf{b}. \tag{15.52}$$

Note that for a rational surface $\mathbf{r}(u, v) = \mathbf{R}(u, v)/\omega(u, v)$, the first partial derivative $\mathbf{r}_u(u, v) = (\mathbf{R}_u - \mathbf{r}\,\omega_u)/\omega$, and the second partial derivative $\mathbf{r}_{uu} = \dfrac{\mathbf{R}_{uu} - \mathbf{r}\omega_{uu}}{\omega} - \dfrac{\mathbf{R}_u\omega_u - \mathbf{r}\omega_u^2}{\omega^2} - \mathbf{r}_u\dfrac{\omega_u}{\omega}$. Applying these to the rational Bézier patch (15.45) and letting $(u, v) = (0, 0)$ lead to

$$\mathbf{r}_{uu} = n(n-1)\frac{\omega_{20}}{\omega_{00}}\mathbf{c} + (\cdots)\mathbf{a}$$

where $(\cdots)$ denotes a rather complicated expression that we will not need to be concerned with since subsequent dotting with $\mathbf{n}$ will cause it to vanish. Similarly, we can obtain

$$\mathbf{r}_{uv} = nm\frac{\omega_{11}}{\omega_{00}}\mathbf{f} + (\cdots)\mathbf{a} + (\cdots)\mathbf{b},$$

$$\mathbf{r}_{vv} = m(m-1)\frac{\omega_{02}}{\omega_{00}}\mathbf{d} + (\cdots)\mathbf{b}.$$

Also note that $\mathbf{n} = \mathbf{a}\times\mathbf{b}/\|\mathbf{a}\times\mathbf{b}\|$. By the definition (15.49) of L, M and N, we have

$$L = n(n-1)\frac{\omega_{20}}{\omega_{00}}h_{20}, \quad M = nm\frac{\omega_{11}}{\omega_{00}}h_{11}, \quad N = m(m-1)\frac{\omega_{02}}{\omega_{00}}h_{02}. \tag{15.53}$$

Now substituting (15.52) and (15.53) into (15.46) and (15.47), and doing some simplifications, we get the formulae for Gaussian and mean curvatures

$$K_g = \left(\frac{\omega_{00}}{\omega_{10}\omega_{01}}\right)^2 \frac{\frac{n-1}{n}\,\frac{m-1}{m}\omega_{20}\omega_{02}h_{20}h_{02} - \omega_{11}^2h_{11}^2}{\|\mathbf{a}\times\mathbf{b}\|^2}, \tag{15.54}$$

$$K_m = \frac{\omega_{00}}{\omega_{10}^2\omega_{01}^2}\,\frac{\frac{n-1}{n}\omega_{01}^2\mathbf{b}^2\omega_{20}h_{20} - 2\omega_{10}\omega_{01}(\mathbf{a}\cdot\mathbf{b})\omega_{11}h_{11} + \frac{m-1}{m}\omega_{10}^2\mathbf{a}^2\omega_{02}h_{02}}{2\|\mathbf{a}\times\mathbf{b}\|^2}. \tag{15.55}$$

If we further introduce notations

$$\begin{aligned}
&\tilde{a} = \frac{\omega_{10}}{\omega_{00}}\|\mathbf{a}\|, \qquad \tilde{b} = \frac{\omega_{01}}{\omega_{00}}\|\mathbf{b}\|, \qquad \tilde{S} = \frac{1}{2}\,\|\frac{\omega_{10}}{\omega_{00}}\mathbf{a}\times\frac{\omega_{01}}{\omega_{00}}\mathbf{b}\| = S\frac{\omega_{10}\omega_{01}}{\omega_{00}^2},\\
&\tilde{h}_{11} = \frac{\omega_{11}}{\omega_{00}}h_{11}, \quad \tilde{h}_{20} = \frac{n-1}{n}\,\frac{\omega_{20}}{\omega_{00}}h_{20}, \quad \tilde{h}_{02} = \frac{m-1}{m}\,\frac{\omega_{02}}{\omega_{00}}h_{02},
\end{aligned}$$

then the formulae can be symbolically simplified to

$$K_g = \left(\tilde{h}_{20}\,\tilde{h}_{02} - \tilde{h}_{11}^2\right)/4\tilde{S}^2, \tag{15.56}$$

$$K_m = \left(\tilde{h}_{20}\,\tilde{b}^2 - 2\tilde{h}_{11}\,\tilde{a}\,\tilde{b}\,\cos\theta + \tilde{h}_{02}\,\tilde{a}^2\right)/8\tilde{S}^2. \tag{15.57}$$

Remark 1. Obviously, formulae (15.54) and (15.55) or (15.56) and (15.57) just contain some simple geometric quantities, like (scaled) length or area, related to the control mesh of the rational Bézier patch. Compared to the formulae (15.46) and (15.47), they are more intuitive and also simpler to compute.

Remark 2. Though the equations derived above are valid at the bottom-left corner, by symmetry similar formulae are easily written out at the other three corners of the rational Bézier patch. Moreover, at any point of the surface other than the four corners, the formulae can also be used to calculate curvatures with help of subdividing the surface there.

Remark 3. According to differential geometry, from Gaussian and mean curvatures, we can compute the principal curvatures

$$k_{1,2} = K_m \pm \sqrt{K_m^2 - K_g}.$$

Meanwhile, the principal directions of curvatures can be determined by

$$\frac{du}{dv} = -\frac{M - k_{1,2}F}{L - k_{1,2}E} = -\frac{N - k_{1,2}G}{M - k_{1,2}F}.$$

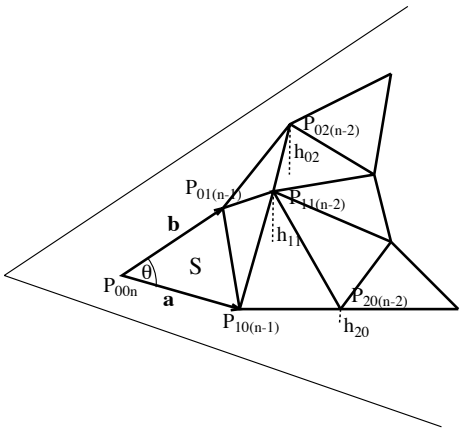


Figure 15.25: Part of a triangular mesh.

15.9.2 Curvatures of triangular rational Bézier surfaces

A triangular rational Bézier patch of degree n is defined by

$$\mathbf{r}(u,v) = \frac{\sum\limits_{i+j+k=n} \omega_{ijk}\mathbf{P}_{ijk}B^n_{ijk}(u,v)}{\sum\limits_{i+j+k+n} \omega_{ijk}B^n_{ijk}(u,v)} \tag{15.58}$$

where $B^n_{ijk}(u,v) = \dfrac{n!}{i!j!k!}u^i v^j(1-u-v)^k$ are Bernstein polynomials; $\mathbf{P}_{ijk}$ are the control points, forming a triangular control mesh (see Figure 15.25); and ω_{ijk} are the weights.

As in the tensor-product case, we only consider the curvatures at corner $(u,v) = (0,0)$. Although we can follow the straightforward approach of Section 15.9.1, i.e., computing the partial derivatives of the triangular Bézier patch, here we take another approach. Since the triangular patch $\mathbf{r}(u,v)$ can be considered as a degree $n \times n$ tensor-product rational Bézier patch, we can utilize the established formulae (15.54) and (15.55).

Let $\mathbf{a} = \mathbf{P}_{10(n-1)} - \mathbf{P}_{00n}$, $\mathbf{b} = \mathbf{P}_{01(n-1)} - \mathbf{P}_{00n}$, S denote the area of the triangle $\mathbf{P}_{00n}\mathbf{P}_{10(n-1)}\mathbf{P}_{01(n-1)}$, θ be the angle between vectors $\mathbf{P}_{00n}\mathbf{P}_{10(n-1)}$ and $\mathbf{P}_{00n}\mathbf{P}_{01(n-1)}$, and h_{ij} be the signed distance from $\mathbf{P}_{ij(n-i-j)}$ to the plane spanned by $\mathbf{P}_{00n}, \mathbf{P}_{10(n-1)}$ and $\mathbf{P}_{01(n-1)}$. Suppose the tensor-product representation of the patch (15.58) is

$$\mathbf{r}(u,v) = \frac{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n} \bar{\omega}_{ij}\bar{\mathbf{P}}_{ij}B^n_i(u)B^n_j(v)}{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n} \bar{\omega}_{ij}B^n_i(u)B^n_j(v)}.$$

It is easy to show that $\bar{\mathbf{P}}_{ij} = \mathbf{P}_{ij(n-i-j)}$ and $\bar{\omega}_{ij} = \omega_{ij(n-i-j)}$ for $(i,j) = (0,0), (1,0)$, and $(0,1)$. Thus the geometric quantities, like $\mathbf{a}$, $\mathbf{b}$, h_{ij}, and etc., are the same as their counterparts in the tensor-product representation except for h_{11}. Therefore we only need to check $\bar{\mathbf{P}}_{11}$ and $\bar{\omega}_{11}$. Considering the mixed derivative of the denominators of the above two representations at $(u,v) = (0,0)$, we get

$$\bar{\omega}_{11} = \frac{n-1}{n}\omega_{11(n-2)} + \frac{\omega_{10(n-1)} + \omega_{01(n-1)} - \omega_{00n}}{n}$$

Similarly, by considering the numerators, we have

$$\bar{\mathbf{P}}_{11} = \frac{n-1}{n}\frac{\omega_{11(n-2)}}{\bar{\omega}_{11}}\mathbf{P}_{11(n-2)} + \frac{\omega_{10(n-1)}\mathbf{P}_{10(n-1)} + \omega_{01(n-1)}\mathbf{P}_{01(n-1)} - \omega_{00n}\mathbf{P}_{00n}}{n\bar{\omega}_{11}}$$

If we denote the signed distance of $\bar{\mathbf{P}}_{11}$ to the plane $\bar{\mathbf{P}}_{00}\bar{\mathbf{P}}_{10}\bar{\mathbf{P}}_{01}$ by $\bar{h}_{11}$, then $\bar{\omega}_{11}\bar{h}_{11} = \dfrac{n-1}{n}\omega_{11(n-2)}h_{11}$. We substitute all the above relations into (15.54) and (15.55), and the formulae of Gaussian and mean curvatures for the triangular rational Bézier patch (15.58) turn out to be

$$K_g = \left(\frac{n-1}{n}\right)^2 \left(\frac{\omega_{00n}}{\omega_{10(n-1)}\omega_{01(n-1)}}\right)^2 \frac{\omega_{20(n-2)}\omega_{02(n-2)}h_{20}h_{02} - \omega_{11(n-2)}^2 h_{11}^2}{\|\mathbf{a}\times\mathbf{b}\|^2}, \tag{15.59}$$

$$K_m = \frac{n-1}{n}\,\frac{\omega_{00n}}{\omega_{10(n-1)}^2\omega_{01(n-1)}^2}\cdot \frac{\omega_{01(n-1)}^2\mathbf{b}^2\omega_{20(n-2)}h_{20} - 2\omega_{10(n-1)}\omega_{01(n-1)}(\mathbf{a}\cdot\mathbf{b})\omega_{11(n-2)}h_{11} + \omega_{10(n-1)}^2\mathbf{a}^2\omega_{02(n-2)}h_{02}}{2\|\mathbf{a}\times\mathbf{b}\|^2}, \tag{15.60}$$

or just (15.56) and (15.57) if the following notations are adopted

$$\tilde{a} = \frac{\omega_{10(n-1)}}{\omega_{00n}}\|\mathbf{a}\|, \qquad \tilde{b} = \frac{\omega_{01(n-1)}}{\omega_{00n}}\|\mathbf{b}\|,$$
$$\tilde{S} = S\frac{\omega_{10(n-1)}\omega_{01(n-1)}}{\omega_{00n}^2}, \qquad \tilde{h}_{ij} = \frac{n-1}{n}\,\frac{\omega_{ij(n-i-j)}}{\omega_{00n}}h_{ij}.$$

15.9.3 Curvature of an Implicit Surface

We complete our discussion of curvature by providing the formulae for the Gaussian and mean curvatures at a point on an implicit surface $f(x,y,z)=0$:

$$K_m = \frac{f_{zz}(f_y^2+f_x^2) - 2f_xf_yf_{xy} + f_{xx}(f_y^2+f_z^2) - 2f_yf_zf_{yz} + f_{yy}(f_x^2+f_z^2) - 2f_zf_xf_{xz}}{2\lambda^3} \tag{15.61}$$

$$K_g = \frac{1}{\lambda^4}[-2f_xf_yf_{xy}f_{zz} + f_{yy}f_{xx}f_z^2 - 2f_yf_zf_{yz}f_{xx} + f_{zz}f_{yy}f_x^2 - 2f_zf_xf_{xz}f_{yy} + f_{xx}f_{zz}f_y^2$$
$$-f_x^2f_{yz}^2 - f_y^2f_{xz}^2 - f_z^2f_{xy}^2 + 2f_xf_yf_{xz}f_{yz} + 2f_yf_zf_{xy}f_{xz} + 2f_xf_zf_{xy}f_{yz}] \tag{15.62}$$

where

$$\lambda = \sqrt{f_x^2+f_y^2+f_z^2}$$

Chapter 16

Computing Points and Tangents on Bézier Surface Patches

This chapter presents an algorithm for computing points and tangents on a tensor-product rational Bézier surface patch that has $O(n^2)$ time complexity.

A rational Béziercurve in $\mathbf{R}^3$ is defined

$$\mathbf{p}(t) = \Pi\left(\mathbf{P}(t)\right) \tag{16.1}$$

with

$$\mathbf{P}(t) = (\mathbf{P}_x(t), \mathbf{P}_y(t), \mathbf{P}_z(t), \mathbf{P}_w(t)) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t) \tag{16.2}$$

where $\mathbf{P}_i = w_i(x_i, y_i, z_i, 1)$ and the projection operator Π is defined $\Pi(x, y, z, w) = (x/w, y/w, z/w)$. We will use upper case bold-face variables to denote four-tuples (homogeneous points) and lower case bold-face for triples (points in R^3).

The point and tangent of this curve can be found using the familiar construction

$$\mathbf{P}(t) = (1-t)\mathbf{Q}(t) + t\mathbf{R}(t) \tag{16.3}$$

with

$$\mathbf{Q}(t) = \sum_{i=0}^{n-1} \mathbf{P}_i B_i^{n-1}(t) \tag{16.4}$$

and

$$\mathbf{R}(t) = \sum_{i=1}^{n} \mathbf{P}_i B_{i-1}^{n-1}(t) \tag{16.5}$$

where line $\mathbf{q}(t)$—$\mathbf{r}(t) \equiv \Pi\left(\mathbf{Q}(t)\right)$—$\Pi\left(\mathbf{R}(t)\right)$ is tangent to the curve, as seen in Figure 16.1. As a sidenote, the correct magnitude of the derivative of $\mathbf{p}(t)$ is given by

$$\frac{d\mathbf{p}(t)}{dt} = n\frac{\mathbf{R}_w(t)\mathbf{Q}_w(t)}{((1-t)\mathbf{Q}_w(t) + t\mathbf{R}_w(t))^2}\left[\mathbf{r}(t) - \mathbf{q}(t)\right] \tag{16.6}$$

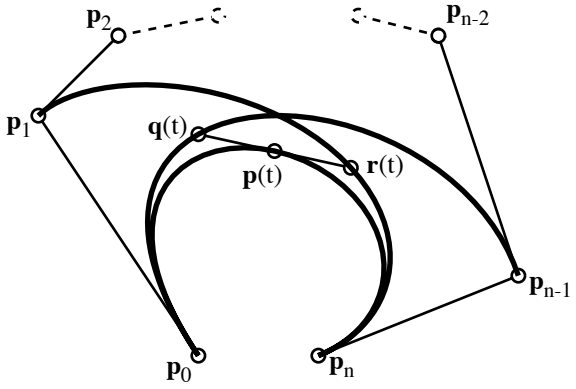


Figure 16.1: Curve example

The values $\mathbf{Q}(t)$ and $\mathbf{R}(t)$ can be found using the modified Horner's algorithm for Bernstein polynomials, involving a pseudo–basis conversion

$$\frac{\mathbf{Q}(t)}{(1-t)^{n-1}} = \hat{\mathbf{Q}}(u) = \sum_{i=0}^{n-1} \hat{\mathbf{Q}}_i u^i \tag{16.7}$$

where $u = \frac{t}{1-t}$ and $\hat{\mathbf{Q}}_i = \binom{n-1}{i}\mathbf{P}_i$, $i = 0, 1, \ldots, n-1$. Assuming the curve is to be evaluated several times, we can ignore the expense of precomputing the $\hat{\mathbf{Q}}_i$, and the nested multiplication

$$\hat{\mathbf{Q}}(u) = [\cdots[[\hat{\mathbf{Q}}_{n-1}u + \hat{\mathbf{Q}}_{n-2}]u + \hat{\mathbf{Q}}_{n-3}]u + \ldots \hat{\mathbf{Q}}_1]u + \hat{\mathbf{Q}}_0 \tag{16.8}$$

can be performed with $n-1$ multiplies and adds for each of the four x, y, z, w coordinates. It is not necessary to post-multiply by $(1-t)^{n-1}$, since

$$\Pi\left(\mathbf{Q}(t)\right) = \Pi\left((1-t)^{n-1}\hat{\mathbf{Q}}(u)\right) = \Pi\left(\hat{\mathbf{Q}}(t)\right). \tag{16.9}$$

Therefore, the point $\mathbf{P}(t)$ and its tangent direction can be computed with roughly $2n$ multiplies and adds for each of the four x, y, z, w coordinates.

This method has problems near $t = 1$, so it is best for $.5 \leq t \leq 1$ to use the form

$$\frac{\mathbf{Q}(t)}{t^{n-1}} = \sum_{i=0}^{n-1} \hat{\mathbf{Q}}_{n-i-1} u^i \tag{16.10}$$

with $u = \frac{1-t}{t}$.

A tensor product rational Béziersurface patch is defined

$$\mathbf{p}(s,t) = \Pi\left(\mathbf{P}(s,t)\right) \tag{16.11}$$

where

$$\mathbf{P}(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{P}_{ij} B_i^m(s) B_j^n(t). \tag{16.12}$$

We can represent the surface $\mathbf{p}(s,t)$ using the following construction:

$$\mathbf{P}(s,t) = (1-s)(1-t)\mathbf{P}^{00}(s,t) + s(1-t)\mathbf{P}^{10}(s,t) + (1-s)t\mathbf{P}^{01}(s,t) + st\mathbf{P}^{11}(s,t) \tag{16.13}$$

where

$$\mathbf{P}^{00}(s,t) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\mathbf{P}_{ij}B_i^{m-1}(s)B_j^{n-1}(t), \tag{16.14}$$

$$\mathbf{P}^{10}(s,t) = \sum_{i=1}^{m}\sum_{j=0}^{n-1}\mathbf{P}_{ij}B_{i-1}^{m-1}(s)B_j^{n-1}(t), \tag{16.15}$$

$$\mathbf{P}^{01}(s,t) = \sum_{i=0}^{m-1}\sum_{j=1}^{n}\mathbf{P}_{ij}B_i^{m-1}(s)B_{j-1}^{n-1}(t), \tag{16.16}$$

$$\mathbf{P}^{11}(s,t) = \sum_{i=1}^{m}\sum_{j=1}^{n}\mathbf{P}_{ij}B_{i-1}^{m-1}(s)B_{j-1}^{n-1}(t). \tag{16.17}$$

The tangent vector $\mathbf{p}_s(s,t)$ is parallel with the line

$$\Pi\left((1-t)\mathbf{P}^{00}(s,t) + t\mathbf{P}^{01}(s,t)\right) — \Pi\left((1-t)\mathbf{P}^{10}(s,t) + t\mathbf{P}^{11}(s,t)\right) \tag{16.18}$$

and the tangent vector $\mathbf{p}_t(s,t)$ is parallel with

$$\Pi\left((1-s)\mathbf{P}^{00}(s,t) + s\mathbf{P}^{10}(s,t)\right) — \Pi\left((1-s)\mathbf{P}^{01}(s,t) + s\mathbf{P}^{11}(s,t)\right). \tag{16.19}$$

The Horner algorithm for a tensor product surface emerges by defining

$$\frac{\mathbf{P}^{kl}(s,t)}{(1-s)^{m-1}(1-t)^{n-1}} = \hat{\mathbf{P}}^{kl}(u,v) = \sum_{i=k}^{m+k-1}\sum_{j=l}^{n+l-1}\hat{\mathbf{P}}_{ij}^{kl}u^i v^j; \quad k,l = 0,1 \tag{16.20}$$

where $u = \frac{s}{1-s}$, $v = \frac{t}{1-t}$, and $\hat{\mathbf{P}}_{ij}^{kl} = \binom{m-1}{i-k}\binom{n-1}{j-l}\mathbf{P}_{ij}$. The n rows of these four bivariate polynomials can each be evaluated using $m-1$ multiplies and adds per x,y,z,w component, and the final evaluation in t costs $n-1$ multiplies and adds per x,y,z,w component.

Thus, if $m = n$, the four surfaces $\mathbf{P}^{00}(s,t)$, $\mathbf{P}^{01}(s,t)$, $\mathbf{P}^{10}(s,t)$, and $\mathbf{P}^{11}(s,t)$ can each be evaluated using n^2-1 multiplies and n^2-1 adds for each of the four x,y,z,w components, a total of $16n^2-16$ multiplies and $16n^2-16$ adds.

If one wishes to compute a grid of points on this surface which are evenly spaced in parameter space, the four surfaces $\mathbf{P}^{00}(s,t)$, $\mathbf{P}^{01}(s,t)$, $\mathbf{P}^{10}(s,t)$, and $\mathbf{P}^{11}(s,t)$ can each be evaluated even more quickly using forward differencing.

Chapter 17

Algebraic Geometry for CAGD

Initially, the field of computer aided geometric design and graphics drew most heavily from differential geometry, approximation theory, and vector geometry. Since the early 1980's, some of the tools of algebraic geometry have been introduced into the CAGD literature. This chapter presents some of those tools, which can address the following problems:

1. Given a planar curve defined parametrically as $x = \frac{x(t)}{w(t)}$, $y = \frac{y(t)}{w(t)}$ where $x(t)$, $y(t)$, and $w(t)$ are polynomials, find an implicit equation $f(x, y) = 0$ which defines the same curve. This process of parametric to implicit conversion will be referred to as *implicitization.*

2. Given the (x, y) coordinates of a point which lies on a parametric curve $x = \frac{x(t)}{w(t)}$, $y = \frac{y(t)}{w(t)}$, find the parameter value t which corresponds to that point. This problem will be referred to as the *inversion* problem.

3. Compute the points of intersection of two parametric curves using the implicitization and inversion techniques.

Section 1.4 presents some preliminary terminology and theorems. Sections 17.1 through 17.4 discuss the implicitization and inversion of planar curves, and Section 17.5 applies those tools to computing curve intersections. Section 17.8 discusses some special properties of parametric cubic curves and Section 17.9 overviews surface implicitization. Section 17.11 discusses Gróbner bases.

17.1 Implicitization

It was noted that there are basically two ways that a planar curve can be defined: parametrically ($x = x(t)/w(t)$, $y = y(t)/w(t)$) and implicitly ($f(x, y) = 0$).

Obviously, the parametric equation of a curve has the advantage of being able to quickly compute the (x, y) coordinates of several points on the curve for plotting purposes. Also, it is simple to define a curve *segment* by restricting the parameter t to a finite range, for example $0 \leq t \leq 1$. On the other hand, the implicit equation of a curve enables one to easily determine whether a given point lies on the curve, or if not, which side of the curve it lies on.

Given these two different equations for curves, it is natural to wonder if it is possible to convert between representations for a given curve. The answer is that it is *always* possible to find an implicit equation of a parametric curve, but a parametric equation can generally be found only

for implicit curves of degree two or one. The process of finding the implicit equation of a curve which is expressed parametrically is referred to as *implicitization.* In Section 17.4, we will discuss how this can be accomplished using an important algebraic tool, the *resultant,* and Section 17.3 discusses resultants. Section 17.2 suggests how someone might tackle the implicitization problem before learning about resultants. Section 17.5 applies these ideas to the problem of intersecting two parametric curves.

17.2 Brute Force Implicization

Consider this simple example of parametric-to-implicit conversion: Given a line

$$x = t + 2 \qquad y = 3t + 1,$$

we can easily find an implicit equation which identically represents this line by solving for t as a function of x

$$t = x - 2$$

and substituting into the equation for y:

$$y = 3(x - 2) + 1$$

or $3x - y - 5 = 0$. Note that this implicit equation defines *precisely* the same curve as does the parametric equation. We can also identify two inversion equations (for finding the parameter value of a point on the line): $t = x - 2$ or $t = (y - 1)/3$.

This approach to implicitization also works for degree two parametric curves. Consider the parabola

$$x = t^2 + 1 \qquad y = t^2 + 2t - 2.$$

Again, we can solve for t as a function of x:

$$t = \pm\sqrt{x - 1}$$

and substitute into the equation for y:

$$y = (\sqrt{x - 1})^2 \pm 2\sqrt{x - 1} - 2.$$

We can isolate the radical and square both sides

$$(y - (x - 1) + 2)^2 = (\pm 2\sqrt{x - 1})^2$$

to yield

$$x^2 - 2xy + y^2 - 10x + 6y + 13 = 0$$

which is the desired implicit equation. Again, this implicit equation defines exactly the same curve as does the parametric equation.

We run into trouble if we try to apply this implicitization technique to curves of degree higher than two. Note that the critical step is that we must be able to express t as a function of x. For cubic and quartic equations, this can be done, but the resulting expression is hopelessly complex. For curves of degree greater than four, it is simply not possible.

We cannot obtain an inversion equation for this parabola the way we did for the straight line. For example, suppose we want to find the parameter of the point $(5, -2)$ which we know to lie on the curve. The brute force approach would be to find the values of t which satisfy the equation

$$x \;=\; 5 \;=\; t^2 \;+\; 1$$

and then to compare them with the values of t which satisfy the equation

$$y \;=\; -2 \;=\; t^2 \;+\; 2t \;-\; 2.$$

In the first case, we find t = -2 or 2, and in the second case, t = -2 or 0. The value of t which satisfies both equations is -2, which must therefore be the parameter value of the point (5,-2).

This unsuccessful attempt at implicitization and inversion motivates the following discussion of *resultants,* which will provide an elegant, general solution to the implicitization and inversion problems.

17.3 Polynomial Resultants

17.3.1 Definition of the Resultant of Two Polynomials

Polynomial resultants address the question of whether two polynomials have a common root. Consider the two polynomials

$$f(t) \;=\; \sum_{i=0}^{n} a_i t^i \qquad\qquad g(t) \;=\; \sum_{i=0}^{n} b_i t^i \tag{17.1}$$

The resultant of $f(t)$ and $g(t)$, written $R(f, g)$, is an expression for which $R(f, g) \;=\; 0$ if and only if$f(t)$ and $g(t)$ have a common root.

Consider the resultant of two polynomials given in factored form:

$$f(t) = (t - f_1)(t - f_2)\cdots(t - f_m), \qquad g(t) = (t - g_1)(t - g_2)\cdots(t - g_n) \tag{17.2}$$

where f_1, f_2, ... f_m are the roots of $f(t)$ and g_1, g_2, ... g_n are the roots of $g(t)$. The resultant of $f(t)$ and $g(t)$ is the unique polynomial expression that will be zero if and only if at least one f_i is the same as at least one g_i:

$$R(f, g) = \prod_{i=1}^{m}\prod_{j=1}^{n}(f_i - g_j)$$

For example, the resultant of $f(t) = t^2 - 7t + 12 = (t-3)(t-4)$ and $g(t) = t^2 - 3t + 2 = (t-2)(t-1)$ is

$$R(f, g) = (3-2)(3-1)(4-2)(4-1) = 12 \tag{17.3}$$

while the resultant of $f(t) = t^2 - 7t + 12 = (t-3)(t-4)$ and $g(t) = t^2 - 5t + 6 = (t-2)(t-3)$ is

$$R(f, g) = (3-2)(3-3)(4-2)(4-3) = 0.$$

Of course, if $f(t)$ and $g(t)$ are given in factored form, it is a trivial matter to detect if they have any common roots. However, it takes a fair amount of computation to determine all roots of a polynomial, and those roots are typically not rational and often complex. Therefore, of much greater value would an equation for a resultant that does *not* require the polynomials to be given in factored form, but can be computed directly from equations (17.1). The good news is that such equations for a resultant do exist and are relatively easy to compute, without needing to first compute any polynomial roots. Such an equation for the resultant can be written in closed form. Furthermore, if the polynomial coefficients are integers, the resultant will also be an integer, even if the roots are complex!

17.3.2 Resultant of Two Degree One Polynomials

The simplest resultant is for the case when $f(t)$ and $g(t)$ are degree one:

$$f(t) = a_1 t + a_0; \qquad g(t) = b_1 t + b_0.$$

The root of $f(t)$ and the root of $g(t)$ are easily found:

$$a_1 t + a_0 = 0 \;\rightarrow\; t = -\frac{a_0}{a_1}; \qquad b_1 t + b_0 = 0 \;\rightarrow\; t = -\frac{b_0}{b_1};$$

Since each polynomial has exactly one root, $f(t)$ and $g(t)$ have a *common* root if and only if

$$-\frac{a_0}{a_1} = \frac{b_0}{b_1}, \quad \text{or} \quad a_1 b_0 - b_1 a_0 = 0.$$

The resultant of $f(t)$ and $g(t)$ is thus

$$R(f, g) = a_1 b_0 - b_1 a_0. \tag{17.4}$$

We can also derive this resultant using matrix algebra. This will make more sense if we use the homogeneous form

$$f(T, U) = a_1 T + a_0 U; \qquad g(T, U) = b_1 T + b_0 U$$

where $t = T/U$ The equations $f(T, U) = 0$ and $g(T, U) = 0$ can be written

$$\begin{bmatrix} a_1 & a_0 \\ b_1 & b_0 \end{bmatrix} \begin{bmatrix} T \\ U \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{17.5}$$

This is a system of homogeneous linear equations. We know from linear algebra that the necessary and sufficient condition for this system to have a solution is if the determinant of the matrix is zero, that is, if

$$\begin{vmatrix} a_1 & a_0 \\ b_1 & b_0 \end{vmatrix} = a_1 b_0 - b_1 a_0 = 0$$

This is the same equation as the resultant that we arrived at earlier.

This resultant can answer for us the question of whether two degree-one polynomials have a common root.

Example: Do $f(t) = 2t + 1$ and $g(t) = t + 3$ have a common root? They have a common root if and only if their resultant is zero. Since $R(f, g) = 2 \cdot 3 - 1 \cdot 1 = 5 \neq 0$, they do not have a common root.

Example: Do $f(t) = t - 2$ and $g(t) = 3t - 6$ have a common root? Since $R(f, g) = 1 \cdot (-6) - 3 \cdot (-2) = 0$, they do have a common root.

17.3.3 Resultants of Degree-Two Polynomials

The degree one resultant is obvious. Higher-degree resultants are not so obvious. Consider the degree two polynomials

$$f(t) = a_2 t^2 + a_1 t + a_0, \qquad g(t) = b_2 t^2 + b_1 t + b_0.$$

It does not work so well to solve for the roots of these two polynomials and check for a common root. The roots of $f(t)$ and $g(t)$ are, respectively,

$$t = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2a_0}}{2a_2}, \quad t = \frac{-b_1 \pm \sqrt{b_1^2 - 4b_2b_0}}{2b_2}$$

So, $f(t)$ and $g(t)$ have a common root if and only if

$$\frac{-a_1 + \sqrt{a_1^2 - 4a_2a_0}}{2a_2} = \frac{-b_1 + \sqrt{b_1^2 - 4b_2b_0}}{2b_2}, \text{ or } \frac{-a_1 + \sqrt{a_1^2 - 4a_2a_0}}{2a_2} = \frac{-b_1 - \sqrt{b_1^2 - 4b_2b_0}}{2b_2}, \text{ or}$$

$$\frac{-a_1 - \sqrt{a_1^2 - 4a_2a_0}}{2a_2} = \frac{-b_1 + \sqrt{b_1^2 - 4b_2b_0}}{2b_2}, \text{ or } \frac{-a_1 - \sqrt{a_1^2 - 4a_2a_0}}{2a_2} = \frac{-b_1 - \sqrt{b_1^2 - 4b_2b_0}}{2b_2}$$

But, the resultant we are looking for is a single expression in terms of a_0, a_1, a_2, b_0, b_1, and b_2 that will be zero if and only if one or more of the above expressions is true. We could accomplish that some algebraic manipulation of these equations, but there is an easier way, one that extends to polynomials of any degree.

For this degree-two case, the idea is to create a pair of degree-one polynomials $h_1(t)$ and $h_2(t)$ that will have a common root if and only if $f(t)$ and $g(t)$ have a common root. Those polynomials are:

$$\begin{aligned} h_1(t) &= a_2g(t) - b_2f(t) = a_2(b_2t^2 + b_1t + b_0) - b_2(a_2t^2 + a_1t + a_0) \\ &= (2,1)t + (2,0) \end{aligned} \tag{17.6}$$

where the notation $(a_ib_j) = a_ib_j - a_jb_i$. We also create

$$\begin{aligned} h_2(t) &= (a_2t + a_1)g(t) - (b_2t + b_1)f(t) \\ &= (a_2t + a_1)(b_2t^2 + b_1t + b_0) - (b_2t + b_1)(a_2t^2 + a_1t + a_0) \\ &= (a_2b_0)t + (a_1b_0) \end{aligned} \tag{17.7}$$

It is easy to verify that $h_1(t)$ and $h_2(t)$ will each vanish for any value of t that is a common root of $f(t)$ and $g(t)$. Therefore, any common root of $f(t)$ and $g(t)$ is also a common root of $h_1(t)$ and $h_2(t)$. Since we already have a resultant for two linear polynomials (17.4), the resultant of our two quadratic polynomials is

$$R(f,g) = \begin{vmatrix} (a_2b_1) & (a_2b_0) \\ (a_2b_0) & (a_1b_0) \end{vmatrix} \tag{17.8}$$

Example We again compute the resultant of $f(t) = t^2 - 7t + 12 = (t-3)(t-4)$ and $g(t) = t^2 - 3t + 2 = (t-2)(t-1)$, which we saw from (17.3) is equal to 12, but this time we use (**??**).

$$R(f,g) = \begin{vmatrix} (a_2b_1) & (a_2b_0) \\ (a_2b_0) & (a_1b_0) \end{vmatrix} = R(f,g) = \begin{vmatrix} 4 & -10 \\ -10 & 22 \end{vmatrix} = -12 \tag{17.9}$$

We should here note that a resultant computed by (**??**) and a resultant computed using (17.8) can differ by a sign, and that their absolute value will be equal if $a_2 = b_2 = 1$.

17.3.4 Resultants of Degree-Three Polynomials

We illustrate by finding the resultant of two cubic polynomials

$$f(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \quad g(t) = b_3t^3 + b_2t^2 + b_1t + b_0.$$

In other words, we want to determine whether there exists a value α such that $f(\alpha) = g(\alpha) = 0$ without having to actually find all roots of both polynomials and comparing. We begin by forming three auxiliary polynomials $h_1(t)$, $h_2(t)$ and $h_3(t)$ as follows:

$$\begin{aligned} h_1(t) &= a_3g(t) - b_3f(t) \\ &= (a_3b_2)t^2 + (a_3b_1)t + (a_3b_0) \end{aligned}$$

where $(a_ib_j) \equiv (a_ib_j - a_jb_i)$ and

$$\begin{aligned} h_2(t) &= (a_3t + a_2)g(t) - (b_3t + b_2)f(t) \\ &= (a_3b_1)t^2 + [(a_3b_0) + (a_2b_1)]t + (a_2b_0) \end{aligned}$$

$$\begin{aligned} h_3(t) &= (a_3t^2 + a_2t + a_1)g(t) - (b_3t^2 + b_2t + b_1)f(t) \\ &= (a_3b_0)t^2 + (a_2b_0)t + (a_1b_0) \end{aligned}$$

Note that if there exists a value α such that $f(\alpha) = g(\alpha) = 0$, then $h_1(\alpha) = h_2(\alpha) = h_3(\alpha) = 0$. We can therefore say that $f(t)$ and $g(t)$ have a common root if and only if the set of equations

$$\begin{bmatrix} (a_3b_2) & (a_3b_1) & (a_3b_0) \\ (a_3b_1) & (a_3b_0) + (a_2b_1) & (a_2b_0) \\ (a_3b_0) & (a_2b_0) & (a_1b_0) \end{bmatrix} \begin{Bmatrix} t^2 \\ t \\ 1 \end{Bmatrix} = 0$$

has a solution. [1] However, we know from linear algebra that this set of homogeneous linear equations can have a solution if and only if

$$\begin{vmatrix} (a_3b_2) & (a_3b_1) & (a_3b_0) \\ (a_3b_1) & (a_3b_0) + (a_2b_1) & (a_2b_0) \\ (a_3b_0) & (a_2b_0) & (a_1b_0) \end{vmatrix} = 0$$

and therefore,

$$R(f,g) = \begin{vmatrix} (a_3b_2) & (a_3b_1) & (a_3b_0) \\ (a_3b_1) & (a_3b_0) + (a_2b_1) & (a_2b_0) \\ (a_3b_0) & (a_2b_0) & (a_1b_0) \end{vmatrix}$$

This same approach can be used to construct the resultant of polynomials of any degree.

Let's try this resultant on a couple of examples. First, let $f(t) = t^3 - 2t^2 + 3t + 1$ and $g(t) = 2t^3 + 3t^2 - t + 4$. For this case,

$$R(f,g) = \begin{vmatrix} 7 & -7 & 2 \\ -7 & -5 & -11 \\ 2 & -11 & 13 \end{vmatrix} = -1611$$

[1] Actually, we have only shown that this is a necessary condition. The proof that it is also sufficient can be found in [GSA84].

We aren't so much interested in the actual numerical value of the resultant, just whether it equals zero or not. In this case, $R(f,g) = -1611 \neq 0$, so we conclude that $f(t)$ and $g(t)$ do *not* have a common root.

Consider next the pair of polynomials $f(t) = t^3 - t^2 - 11t - 4$ and $g(t) = 2t^3 - 7t^2 - 5t + 4$. In this case,

$$R(f,g) = \begin{vmatrix} -5 & 17 & 12 \\ 17 & -60 & -32 \\ 12 & -32 & -64 \end{vmatrix} = 0$$

Since $R(f,g) = 0$, $f(t)$ and $g(t)$ *do* have a common root. Note that the resultant simply determines the existence or non-existence of a common root, but it does not directly reveal the value of a common root, if one exists. In fact, if the resultant is zero, there may actually be several common roots. Section 17.4 discusses how to compute the common root(s).

17.3.5 Resultants of Higher Degree Polynomials

The formulation of the resultant that we have presented is known as Bezout's resultant. For two polynomials of equal degree, Bezout's resultant is the determinant of an $n \times n$ matrix. The pattern you can observe in the resultants of degree two and degree three polynomials extends to any degree. Resultants for two polynomials of different degree also exist.

Another formulation for resultants, called Sylvester's resultant, is the determinant of a square matrix of size $2n \times 2n$. The pattern for Sylvester's resultant is even easier to see. For two degree-three polynomials, Sylvester's resultant is

$$R(f,g) = \begin{vmatrix} a_3 & a_2 & a_1 & a_0 & 0 & 0 \\ 0 & a_3 & a_2 & a_1 & a_0 & 0 \\ 0 & 0 & a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & b_0 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 & b_0 \end{vmatrix}$$

Sylvester's resultant is equivalent to Bezout's resultant. Clearly, Bezout's resultant is more simple to expand.

17.4 Determining the Common Root

We present two basic approaches to finding the common root of two polynomials: by solving a set of linear equations, or by using Euclid's algorithm.

Linear Equation Approach Our intuitive development of the resultant of two cubic polynomials led us to a set of three linear equations in three "unknowns": t^2, t and 1. In general, we could create the resultant of two degree n polynomials $f(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_1 t + a_0$, $g(t) = b_n t^n + b_{n-1} t^{n-1} + \cdots + b_1 t + b_0$, as the determinant of the coefficient matrix of n homogeneous linear equations:

$$\begin{bmatrix} (a_n b_{n-1}) & . & . & . & (a_n b_0) \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ (a_n b_0) & . & . & . & (a_1 b_0) \end{bmatrix} \begin{Bmatrix} t^{n-1} \\ t^{n-2} \\ . \\ . \\ t \\ 1 \end{Bmatrix} = 0$$

It may be a bit confusing at first to view this as a set of homogeneous *linear* equations, since the unknowns are all powers of t. Let us temporarily switch to homogeneous variables T and U:

$$\begin{bmatrix} (a_n b_{n-1}) & . & . & . & (a_n b_0) \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ (a_n b_0) & . & . & . & (a_1 b_0) \end{bmatrix} \begin{Bmatrix} T^{n-1} \\ T^{n-2}U \\ . \\ . \\ TU^{n-2} \\ U^{n-1} \end{Bmatrix} = 0$$

where $t = T/U$. After solving for any two adjacent terms $T^{n-i}U^{i-1}$, the common root of $f(t)$ and $g(t)$ can be obtained as $t = \frac{T^{n-i+1}U^{i-2}}{T^{n-i}U^{i-1}}$.

Cramer's Rule There are several well known methods for solving for the $T^{n-i}U^{i-1}$. One way is to apply Cramer's rule. A non-trivial solution exists (that is, a solution other than all $T^{n-i}U^{i-1} = 0$) only if the determinant of the matrix is zero. But, that implies that the n equations are linearly dependent and we can discard one of them without losing any information. We discard the *last* equation, and can then solve for $n-1$ homogeneous equations in n homogeneous unknowns using Cramer's rule. It turns out that occasionally we run into trouble if we discard an equation other than the last one. We illustrate Cramer's rule for the case $f(t) = t^3 - t^2 - 11t - 4$ and $g(t) = 2t^3 - 7t^2 - 5t + 4$. Recall that this is the pair for which we earlier found that $R(f, g) = 0$. We have the set of equations

$$\begin{bmatrix} -5 & 17 & 12 \\ 17 & -60 & -32 \\ 12 & -32 & -64 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = 0$$

Discarding the last equation, we obtain

$$\begin{bmatrix} -5 & 17 & 12 \\ 17 & -60 & -32 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = 0$$

from which we find the common root using Cramer's rule:

$$t = \frac{TU}{U^2} = -\frac{\begin{vmatrix} -5 & 12 \\ 17 & -32 \end{vmatrix}}{\begin{vmatrix} -5 & 17 \\ 17 & -60 \end{vmatrix}} = 4$$

Gauss Elimination A numerically superior algorithm for solving this set of equations is to perform Gauss elimination. Two other advantages of Gauss elimination are that it can be used to determine whether the determinant is zero to begin with, and also it reveals *how many* common roots there are. We will illustrate this approach with three examples, using integer preserving Gauss elimination. We choose the integer preserving Gauss elimination because then the lower right hand element of the upper triangular matrix is the value of the determinant of the matrix.

Example 1

Our first example is one we considered earlier: $f(t) = t^3 - 2t^2 + 3t + 1$ and $g(t) = 2t^3 + 3t^2 - t + 4$. We set up the following set of linear equations, and triangularize the matrix using integer preserving Gauss elimination:

$$\begin{bmatrix} 7 & -7 & 2 \\ -7 & -5 & -11 \\ 2 & -11 & 13 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} =$$

$$\begin{bmatrix} 7 & -7 & 2 \\ 0 & -84 & 0 \\ 0 & 0 & -1611 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = 0$$

We observe that the only solution to this set of equations is $T = U = 0$, and conclude that $f(t)$ and $g(t)$ do not have a common root. Note that the lower right element -1611 is the determinant of the original matrix, or the resultant.

Example 2

We next examine the pair of polynomials $f(t) = t^3 - t^2 - 11t - 4$ and $g(t) = 2t^3 - 7t^2 - 5t + 4$. In this case, we have

$$\begin{bmatrix} -5 & 17 & 12 \\ 17 & -60 & -32 \\ 12 & -32 & -64 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} =$$

$$\begin{bmatrix} -5 & 17 & 12 \\ 0 & 11 & -44 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = 0$$

Again, the bottom right element is the value of the determinant, which verifies that the resultant is zero. It is now simple to compute the solution: $TU = 4U^2$, $T^2 = 4TU$. Since $t = T/U$, the common root is $t = 4$.

Example 3

For our final example we analyze the polynomials $f(t) = t^3 - 6t^2 + 11t - 6$ and $g(t) = t^3 - 7t^2 + 14t - 8$. Our linear equations now are:

$$\begin{bmatrix} -1 & 3 & -2 \\ 3 & -9 & 6 \\ -2 & 6 & -4 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = \begin{bmatrix} -1 & 3 & -2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} T^2 \\ TU \\ U^2 \end{Bmatrix} = 0$$

In this case, not only is the resultant zero, but the matrix is rank 1. This means that there are *two* common roots, and they can be found as the solution to the quadratic equation $-t^2 + 3t - 2$, which is $t = 1$ and $t = 2$. Another way of saying this is that $-t^2 + 3t - 2$ is the *Greatest Common Divisor* of $f(t)$ and $g(t)$.

Euclid's GCD Algorithm An alternative approach to finding the common root(s) of two polynomials is to use Euclid's algorithm. This ancient algorithm can be used to find the *Greatest Common Divisor* of two integers or two polynomials. A clear proof of Euclid's algorithm can be found in [Kur80]. Our presentation consists of a series of examples. This algorithm works beautifully in exact integer arithmetic but we have experienced numerical instability in floating point.

Integer Example

We illustrate first on a pair of integers: 42 and 30. For the first step, we assign the larger to be the numerator, and the other to be the denominator:

Step 1: $\frac{42}{30} = 1$ remainder 12.

We now take the remainder of the first step and divide it into the denominator of the first step:

Step 2: $\frac{30}{12} = 2$ remainder 6.

We continue dividing the remainder of the preceding step into the denominator of the preceding step until we obtain a zero remainder. This happens to occur in the third step for this problem:

Step 3: $\frac{12}{6} = 2$ exactly.

According to Euclid's algorithm, the second to last remainder is the GCD. In this case, the second to last remainder is 6, which is clearly the largest integer that evenly divides 30 and 42.

Polynomial Example 1

We illustrate how Euclid's algorithm works for polynomials by using the same three examples we used in the previous section. For the polynomials $f(t) = t^3 - 2t^2 + 3t + 1$ and $g(t) = 2t^3 + 3t^2 - t + 4$, we have:

- Step 1: $\frac{2t^3 + 3t^2 - t + 4}{t^3 - 2t^2 + 3t + 1} = 2$ remainder $7t^2 - 7t + 2$.
- Step 2: $\frac{t^3 - 2t^2 + 3t + 1}{7t^2 - 7t + 2} = \frac{t - 1}{7}$ remainder $\frac{12t+9}{7}$
- Step 3: $\frac{7t^2 - 7t + 2}{(12t + 9)/7} = \frac{196t - 343}{48}$ remainder $\frac{1253}{16}$
- Step 4: $\frac{(12t + 9)/7}{1253/16} = \frac{192t}{8771} + \frac{144}{8771}$ remainder 0.

In this case, the GCD is $\frac{1253}{16}$, which is merely a constant, and so $f(t)$ and $g(t)$ do not have a common root.

Polynomial Example 2

We next analyze the polynomials $f(t) = t^3 - t^2 - 11t - 4$ and $g(t) = 2t^3 - 7t^2 - 5t + 4$:

- Step 1: $\frac{2t^3 - 7t^2 - 5t + 4}{t^3 - t^2 - 11t - 4} = 2$ remainder $-5t^2 + 17t + 12$.
- Step 2: $\frac{t^3 - t^2 - 11t - 4}{-5t^2 + 17t + 12} = -5t - \frac{12}{25}$ remainder $\frac{-11t + 44}{25}$
- Step 3: $\frac{-5t^2 + 17t + 12}{(-11t + 44)/25} = 125t + \frac{75}{11}$ remainder 0.

In this case, the GCD is $\frac{-11t + 44}{25}$, and the common root is $t = 4$.

Polynomial Example 3

Finally, consider $f(t) = t^3 - 6t^2 + 11t - 6$ and $g(t) = t^3 - 7t^2 + 14t - 8$:

- Step 1: $\frac{t^3 - 6t^2 + 11t - 6}{t^3 - 7t^2 + 14t - 8} = 1$ remainder $t^2 - 3t + 2$
- Step 2: $\frac{t^3 - 7t^2 + 14t - 8}{t^2 - 3t + 2} = t - 4$ remainder 0.

The GCD is $t^2 - 3t + 2$, and the common roots are the roots of the equation $t^2 - 3t + 2 = 0$ which are $t = 1$ and $t = 2$.

You may have realized that there is a close connection between Euclid's algorithm and resultants, and obviously Euclid's algorithm does everything for us that resultants do.

We are now prepared to apply these tools to the problems of implicitizing and inverting curves.

17.5 Implicitization and Inversion

We discussed in the previous section a tool for determining whether two polynomials have a common root. We want to apply that tool to converting the parametric equation of a curve given by $x = \frac{x(t)}{w(t)}$, $y = \frac{y(t)}{w(t)}$ into an implicit equation of the form $f(x, y) = 0$. We proceed by forming two auxiliary polynomials:

$$p(x,t) = w(t)x - x(t) \qquad q(y,t) = w(t)y - y(t)$$

Note that $p(x,t) = q(y,t) = 0$ only for values of x, y, and t which satisfy the relationships $x = \frac{x(t)}{w(t)}$ and $y = \frac{y(t)}{w(t)}$. View $p(x,t)$ as a polynomial in t whose coefficients are linear in x, and view $q(y,t)$ as a polynomial in t whose coefficients are linear in y. If

$$x(t) = \sum_{i=0}^{n} a_i t^i, \qquad y(t) = \sum_{i=0}^{n} b_i t^i, \qquad w(t) = \sum_{i=0}^{n} d_i t^i$$

then

$$\begin{aligned} p(x,t) &= (d_n x - a_n)t^n + (d_{n-1}x - a_{n-1})t^{n-1} + \cdots \\ &\quad + (d_1 x - a_1)t + (d_0 x - a_0) \\ q(y,t) &= (d_n y - b_n)t^n + (d_{n-1}y - b_{n-1})t^{n-1} + \cdots \\ &\quad + (d_1 y - b_1)t + (d_0 y - b_0) \end{aligned}$$

If we now compute the resultant of $p(x,t)$ and $q(y,t)$, we do not arrive at a numerical value, but rather a *polynomial* in x and y which we shall call $f(x,y)$. Clearly, any (x,y) pair for which $f(x,y) = 0$ causes the resultant of p and q to be zero. But, if the resultant is zero, then we know that there exists a value of t for which $p(x,t) = q(y,t) = 0$. In other words, all (x,y) for which $f(x,y) = 0$ lie on the parametric curve and therefore $f(x,y) = 0$ is the implicit equation of that curve. This should be clarified by the following examples.

Implicitization Example 1

Let's begin by applying this technique to the parabola we implicitized earlier using a brute force method:

$$x = t^2 + 1 \qquad y = t^2 + 2t - 2.$$

We begin by forming $p(x,t) = -t^2 + (x-1)$ and $q(y,t) = -t^2 - 2t + (y+2)$. The resultant of two quadratic polynomials $a_2t^2 + a_1t + a_0$ and $b_2t^2 + b_1t + b_0$ is

$$\begin{vmatrix} (a_2b_1) & (a_2b_0) \\ (a_2b_0) & (a_1b_0) \end{vmatrix}$$

and so the resultant of $p(x,t)$ and $q(y,t)$ is

$$R(p,q) = \begin{vmatrix} 2 & x-y-3 \\ x-y-3 & 2x-2 \end{vmatrix} =$$

$$-x^2 + 2xy - y^2 + 10x - 6y - 13$$

which is the implicit equation we arrived at earlier.

We can write an inversion equation for this curve – something which eluded us in our ad hoc approach:

$$\begin{bmatrix} 2 & x-y-3 \\ x-y-3 & 2x-2 \end{bmatrix} \begin{Bmatrix} t \\ 1 \end{Bmatrix} = 0$$

From which $t = \frac{-x+y+3}{2}$ or $t = \frac{-2x+2}{x-y-3}$.

Implicitization Example 2

We now implicitize the cubic curve for which

$$x = \frac{2t^3 - 18t^2 + 18t + 4}{-3t^2 + 3t + 1}$$

$$y = \frac{39t^3 - 69t^2 + 33t + 1}{-3t^2 + 3t + 1}$$

We begin by forming $p(x,t)$ and $q(y,t)$:

$$p(x,t) = -2t^3 + (-3x + 18)t^2 + (3x - 18)t + (x - 4)$$

$$q(y,t) = -39t^3 + (-3y + 69)t^2 + (3y - 33)t + (y - 1)$$

Recalling from Section 17.3 that the resultant of two cubic polynomials $a_3t^3 + a_2t^2 + a_1t + a_0$ and $b_3t^3 + b_2t^2 + b_1t + b_0$ is

$$\begin{vmatrix} (a_3b_2) & (a_3b_1) & (a_3b_0) \\ (a_3b_1) & (a_3b_0)+(a_2b_1) & (a_2b_0) \\ (a_3b_0) & (a_2b_0) & (a_1b_0) \end{vmatrix},$$

we have

$$R(p,q) = f(x,y) =$$

$$\begin{vmatrix} -117x + 6y + 564 & 117x - 6y - 636 & 39x - 2y - 154 \\ 117x - 6y - 636 & -69x - 2y + 494 & -66x + 6y + 258 \\ 39x - 2y - 154 & -66x + -6y + 258 & 30x - 6y - 114 \end{vmatrix}.$$

We can expand the determinant to get

$$f(x,y) = -156195x^3 + 60426x^2y - 7056xy^2 + 224y^3 + 2188998x^2 - 562500xy + 33168y^2 - 10175796x + 1322088y + 15631624$$

We can obtain an inversion equation using Cramer's rule:

$$t = \frac{T^2}{TU} = -\frac{\begin{vmatrix} (117x - 6y - 636) & (39x - 2y - 154) \\ (-69x - 2y + 494) & (-66x + 6y + 258) \end{vmatrix}}{\begin{vmatrix} (-117x + 69y + 564) & (39x - 2y - 154) \\ (117x - 6y - 636) & (-66x + 6y + 258) \end{vmatrix}}$$

Alternately, we could use Gauss elimination to compute the parameter of a point on the curve. Cramer's rule has the appeal that it actually generates an *equation.*

We have intentionally carried out all computations in exact integer arithmetic to emphasize the rational, non-iterative nature of implicitization and inversion. Since the coefficients of the implicit equation are obtained from the coefficients of the parametric equations using only multiplication, addition and subtraction, it is possible to obtain an implicit equation which *precisely* defines the same point set as is defined by the parametric equations.

17.6 Implicitization in Bézier Form

From the paper [SP86a], a Bézier curve can be implicitized as follows. (Note that the value l_{ij} in these notes is equivalent to the value $L_{j,k+1}$ in the paper).

A degree 2 Bézier curve can be implicitized:

$$f(x,y) = \begin{vmatrix} l_{21}(x,y) & l_{20}(x,y) \\ l_{20}(x,y) & l_{10}(x,y) \end{vmatrix}$$

and a degree 3 Bézier curve can be implicitized

$$f(x,y) = \begin{vmatrix} l_{32}(x,y) & l_{31}(x,y) & l_{30}(x,y) \\ l_{31}(x,y) & l_{30}(x,y) + l_{21}(x,y) & l_{20}(x,y) \\ l_{30}(x,y) & l_{20}(x,y) & l_{10}(x,y) \end{vmatrix}$$

where

$$l_{ij}(x,y) = \binom{n}{i}\binom{n}{j} w_i w_j \begin{vmatrix} x & y & 1 \\ x_i & y_i & 1 \\ x_j & y_j & 1 \end{vmatrix}$$

with n the degree of the curve, and x_i, y_i, w_i the coordinates and weight of the i^{th} control point.

For a general degree n curve, the implicit equation is

$$f(x,y) = \begin{vmatrix} L_{n-1,n-1}(x,y) & \cdot & \cdot & \cdot & L_{0,n-1}(x,y) \\ \cdot & & \cdot & \cdot & \cdot \\ L_{n-1,0}(x,y) & \cdot & \cdot & \cdot & L_{0,0}(x,y) \end{vmatrix}$$

where

$$L_{i,j} = \sum_{\substack{m \le \min(i,j) \\ k+m=i+j+1}} l_{km}.$$

17.6.1 Inversion of Bézier Curves

Inversion is accomplished by solving a set of equations. For the degree-two case, we solve

$$\begin{bmatrix} l_{21}(x,y) & l_{20}(x,y) \\ l_{20}(x,y) & l_{10}(x,y) \end{bmatrix} \begin{Bmatrix} t \\ (1-t) \end{Bmatrix} = 0. \tag{17.10}$$

For example, using the top and bottom rows of the matrix, we can compute two different inversion equations:

$$t = \frac{l_{20}}{l_{20} - l_{21}}, \quad \text{and} \quad t = \frac{l_{10}}{l_{10} - l_{20}}. \tag{17.11}$$

For the degree-three case, we have

$$\begin{bmatrix} l_{32}(x,y) & l_{31}(x,y) & l_{30}(x,y) \\ l_{31}(x,y) & l_{30}(x,y) + l_{21}(x,y) & l_{20}(x,y) \\ l_{30}(x,y) & l_{20}(x,y) & l_{10}(x,y) \end{bmatrix} \begin{Bmatrix} t^2 \\ t(1-t) \\ (1-t)^2 \end{Bmatrix} = 0 \tag{17.12}$$

from which we seek an inversion equation

$$t = \frac{f(x,y)}{g(x,y)}.$$

where $f(x,y)$ and $g(x,y)$ are degree-two polynomials in x and y. It turns out that we can obtain an inversion equation for which $f(x,y)$ and $g(x,y)$ are linear in x and y, as follows. If $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$ are not collinear, there exist constants c_1 and c_2,

$$c_1 = \frac{w_0 \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}{3w_2 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}, \qquad c_2 = -\frac{w_0 \begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}{3w_1 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}$$

such that if we multiple the first row of the matrix in (17.12) by c_1 and the second row by c_2 and add them to the third row, we obtain

$$\begin{bmatrix} l_{32}(x,y) & l_{31}(x,y) & l_{30}(x,y) \\ l_{31}(x,y) & l_{30}(x,y) + l_{21}(x,y) & l_{20}(x,y) \\ 0 & l_a(x,y) & l_b(x,y) \end{bmatrix} \begin{Bmatrix} t^2 \\ t(1-t) \\ (1-t)^2 \end{Bmatrix} = 0 \tag{17.13}$$

where

$$l_a(x,y) = c_1 l_{31}(x,y) + c_2[l_{30}(x,y) + l_{21}(x,y)] + l_{20}(x,y)$$

and

$$l_b(x,y) = c_1 l_{30}(x,y) + c_2 l_{20}(x,y) + l_{10}(x,y).$$

The bottom row of (17.12.a) expresses the equation

$$l_a(x,y)t(1-t) + l_b(x,y)(1-t)^2 = 0 \rightarrow l_a(x,y)t + l_b(x,y)(1-t) = 0$$

from which we can obtain an inversion equation

$$t = \frac{l_b(x,y)}{l_b(x,y) - l_a(x,y)}. \tag{17.14}$$

Example
Find an inversion equation for a polynomial Bézier curve with control points

$$\mathbf{P}_0 = (1,0), \quad \mathbf{P}_1 = (5,0), \quad \mathbf{P}_2 = (5,2), \quad \mathbf{P}_3 = (4,3).$$

Solution

$$l_{32} = 3\begin{vmatrix} x & y & 1 \\ 4 & 3 & 1 \\ 5 & 2 & 1 \end{vmatrix} = 3x + 3y - 21, \qquad l_{31} = 3\begin{vmatrix} x & y & 1 \\ 4 & 3 & 1 \\ 5 & 0 & 1 \end{vmatrix} = 9x + 3y - 45,$$

$$l_{30} = \begin{vmatrix} x & y & 1 \\ 4 & 3 & 1 \\ 1 & 0 & 1 \end{vmatrix} = 3x - 3y - 3, \qquad l_{21} = 9\begin{vmatrix} x & y & 1 \\ 5 & 2 & 1 \\ 5 & 0 & 1 \end{vmatrix} = 18x - 90,$$

$$l_{20} = 3\begin{vmatrix} x & y & 1 \\ 5 & 2 & 1 \\ 1 & 0 & 1 \end{vmatrix} = 6x - 12y - 6, \qquad l_{10} = 3\begin{vmatrix} x & y & 1 \\ 5 & 0 & 1 \\ 1 & 0 & 1 \end{vmatrix} = -12y.$$

$$c_1 = \frac{\begin{vmatrix} 1 & 0 & 1 \\ 5 & 0 & 1 \\ 4 & 3 & 1 \end{vmatrix}}{3\begin{vmatrix} 5 & 0 & 1 \\ 5 & 2 & 1 \\ 4 & 3 & 1 \end{vmatrix}} = 2, \qquad c_2 = -\frac{\begin{vmatrix} 1 & 0 & 1 \\ 5 & 2 & 1 \\ 4 & 3 & 1 \end{vmatrix}}{3\begin{vmatrix} 5 & 0 & 1 \\ 5 & 2 & 1 \\ 4 & 3 & 1 \end{vmatrix}} = -1$$

$$l_a(x,y) = c_1 l_{31}(x,y) + c_2[l_{30}(x,y) + l_{21}(x,y)] + l_{20}(x,y) = 3x - 3y - 3$$

$$l_b(x,y) = c_1 l_{30}(x,y) + c_2 l_{20}(x,y) + l_{10}(x,y) = -6y$$

So,

$$t = \frac{l_b(x,y)}{l_b(x,y) - l_a(x,y)} = \frac{-6y}{-3x - 3y + 3} = \frac{2y}{x + y - 1}.$$

Note that if we plug $(1,0)$ into the inversion equation, we obtain $0/0$. That is because $(1,0)$ is a *double point* on the curve, i.e., a point of self-intersection. There are actually two t values on the curve that map to $(1,0)$: $t = 0$ and $t = 2$, as you can verify. At a double point, an inversion equation will always give $0/0$.

17.7 Curve Inversion Using Linear Algebra

We now present a simple approach to curve inversion that does not involve resultants. For a rational curve of degree n, we can generally write the inversion equation in the form:

$$t = \frac{f(x,y)}{g(x,y)} \tag{17.15}$$

where $f(x,y)$ and $g(x,y)$ are polynomials of degree $n-2$ if $n > 2$, and degree one if $n = 2$. Since we are mainly interested in curves of degree two and three, $f(x,y)$ and $g(x,y)$ are just degree one in those cases.

Suppose we want to find the inversion equation for a rational cubic curve (in homogeneous form)

$$x(t) = x_0 + x_1 t + x_2 t^2 + x_3 t^3$$

$$y(t) = y_0 + y_1 t + y_2 t^2 + y_3 t^3$$

$$w(t) = w_0 + w_1 t + w_2 t^2 + w_3 t^3$$

Denote

$$f(x, y, w) = a_2 x + b_2 y + c_2 w; \qquad g(x, y, w) = a_1 x + b_1 y + c_1 w. \tag{17.16}$$

To find an inversion equation for a cubic curve, we need just determine the coefficients $a_1, a_2, b_1, b_2, c_1, c_2$ such that

$$\frac{f(x(t), y(t), w(t))}{g(x(t), y(t), w(t))} \equiv t \tag{17.17}$$

or

$$t \cdot g(x(t), y(t), w(t)) - f(x(t), y(t), w(t)) \equiv 0 \tag{17.18}$$

or

$$\begin{gathered} t \cdot [a_1(x_0 + x_1 t + x_2 t^2 + x_3 t^3) + b_1(y_0 + y_1 t + y_2 t^2 + y_3 t^3) + c_1(w_0 + w_1 t + w_2 t^2 + w_3 t^3)] \\ -[a_2(x_0 + x_1 t + x_2 t^2 + x_3 t^3) + b_2(y_0 + y_1 t + y_2 t^2 + y_3 t^3) + c_2(w_0 + w_1 t + w_2 t^2 + w_3 t^3)] \equiv 0 \end{gathered}$$

or

$$\begin{array}{rc} t^4 & [a_1 x_3 + b_1 y_3 + c_1 w_3] \\ +t^3 & [a_1 x_2 + b_1 y_2 + c_1 w_2 - a_2 x_3 - b_2 y_3 - c_2 w_3] \\ +t^2 & [a_1 x_1 + b_1 y_1 + c_1 w_1 - a_2 x_2 - b_2 y_2 - c_2 w_2] \\ +t^1 & [a_1 x_0 + b_1 y_0 + c_1 w_0 - a_2 x_1 - b_2 y_1 - c_2 w_1] \\ + & [a_2 x_0 + b_2 y_0 + c_2 w_0] \equiv 0. \end{array} \tag{17.19}$$

This equation will be identically equal to zero if and only if all of the coefficients of the power of t are all zero. Thus, we can solve for the a_i, b_i, and c_i from the set of linear equations:

$$\begin{bmatrix} x_3 & y_3 & w_3 & 0 & 0 & 0 \\ x_2 & y_2 & w_2 & -x_3 & -y_3 & -w_3 \\ x_1 & y_1 & w_1 & -x_2 & -y_2 & -w_2 \\ x_0 & y_0 & w_0 & -x_1 & -y_1 & -w_1 \\ 0 & 0 & 0 & -x_0 & -y_0 & -w_0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{17.20}$$

Of course, degree two curves are even more simple. Note that we can also find inversion equations for 3D curves using this method.

17.8 Curve-Curve Intersections

Given one curve defined by the implicit equation $f(x, y) = 0$ and a second curve defined by the parametric equations $x = x(t)$, $y = y(t)$, we replace all occurrences of x in the implicit equation by $x(t)$, and replace all occurrences of y in the implicit equation by $y(t)$. These substitutions create a polynomial $f(x(t), y(t)) = g(t)$ whose roots are the parameter values of the points of intersection. Of course, if we start off with two parametric curves, we can first implicitize one of them.

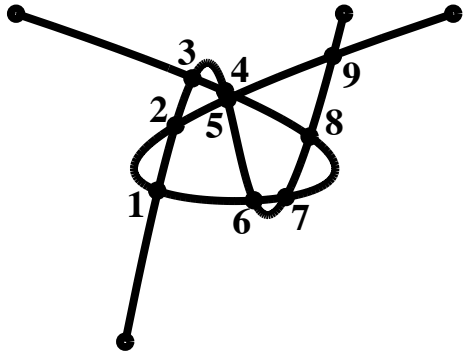


Figure 17.1: Two cubic curves intersecting nine times

We illustrate this process by intersecting the curve

$$x = \frac{2t_1^3 - 18t_1^2 + 18t_1 + 4}{-3t_1^2 + 3t_1 + 1}$$

$$y = \frac{39t_1^3 - 69t_1^2 + 33t_1 + 1}{-3t_1^2 + 3t_1 + 1}$$

with the curve

$$x = \frac{-52t_2^3 + 63t_2^2 - 15t_2 + 7}{-37t_2^2 + 3t_2 + 1}$$

$$y = \frac{4}{-37t_2^2 + 3t_2 + 1}$$

The two curves intersect nine times, which is the most that two cubic curves can intersect. [2] We already implicitized the first curve (in Section 17.3), so our intersection problem requires us to make the substitutions $x = \frac{-52t_2^3 + 63t_2^2 - 15t_2 + 7}{-37t_2^2 + 3t_2 + 1}$ and $y = \frac{4}{-37t_2^2 + 3t_2 + 1}$ into the implicit equation of curve 1:

$$f(x,y) = -156195x^3 + 60426x^2y - 7056xy^2 + 224y^3 +$$
$$2188998x^2 - 562500xy + 33168y^2 - 10175796x + 1322088y +$$
$$15631624.$$

After multiplying through by $(-37t_2^2 + 3t_2 + 1)^3$, we arrive at the intersection equation:

$$984100t_2^9 - 458200t_2^8 + 8868537t_2^7 - 9420593t_2^6 + 5949408t_2^5$$

[2] Bezout's theorem states that two curves of degree m and n respectively, intersect in mn points, if we include complex points, points at infinity, and multiple intersections.

$$- \ 2282850t_2^4 \ + \ 522890t_2^3 \ - \ 67572t_2^2 \ + \ 4401t_2 \ - \ 109 \ = \ 0.$$

Again, we have carried out this process in exact integer arithmetic to emphasize that this equation is an *exact* representation of the intersection points.

We now compute the roots of this degree 9 polynomial. Those roots are the parameter values of the points of intersection. The (x, y) coordinates of those intersection points can be easily found from the parametric equation of the second curve, and the parameter values on the first curve for the intersection points can be found from the inversion equations. The results are tabulated below.

Intersection Number	t_1 **Parameter of Curve 1**	**Coordinates of Point**	t_2 **Parameter of Curve 2**
1	0.0621	(4.2982,2.3787)	0.3489
2	0.1098	(4.4556,2.9718)	0.1330
3	0.1785	(4.6190,3.4127)	0.9389
4	0.3397	(4.9113,3.2894)	0.9219
5	0.4212	(4.9312,3.2186)	0.0889
6	0.6838	(5.1737,2.2902)	0.5339
7	0.8610	(5.4676,2.3212)	0.5944
8	0.9342	(5.6883,2.8773)	0.8463
9	0.9823	(5.9010,3.6148)	0.0369

The most common curve intersection algorithms are currently based on subdivision. Tests indicate that this implicitization algorithm is faster than subdivision methods for curves of degree two and three, and subdivision methods are faster for curves of degree five and greater [Sederberg et al '86].

17.9 Surfaces

Implicitization and inversion algorithms exist for surfaces, also (see [Sederberg '83] or [Sederberg et al. '84b]). But, whereas curve implicitization yields implicit equations of the same degree as the parametric equations, surface implicitization experiences a degree explosion. A triangular surface patch, whose parametric equations are of the form

$$x \ = \ \frac{\sum_{i+j\le n} x_{ij} s^i t^j}{\sum_{i+j\le n} w_{ij} s^i t^j} \qquad y \ = \ \frac{\sum_{i+j\le n} y_{ij} s^i t^j}{\sum_{i+j\le n} w_{ij} s^i t^j}$$

$$z \ = \ \frac{\sum_{i+j\le n} z_{ij} s^i t^j}{\sum_{i+j\le n} w_{ij} s^i t^j} \qquad i, j \ge 0,$$

generally has an implicit equation of degree n^2. A tensor product surface patch, whose parametric equations are of the form

$$x \ = \ \frac{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} x_{ij} s^i t^j}{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} w_{ij} s^i t^j} \qquad y \ = \ \frac{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} y_{ij} s^i t^j}{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} w_{ij} s^i t^j}$$

$$z \ = \ \frac{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} z_{ij} s^i t^j}{\sum_{i\,=\,0}^{n} \sum_{j\,=\,0}^{m} w_{ij} s^i t^j},$$

generally has an implicit equation of degree 2mn. Thus, a bicubic patch generally has an implicit equation $f(x, y, z) = 0$ of degree 18. Such an equation has 1330 terms!

Algebraic geometry shares important information on the nature of intersections of parametric surfaces. Recall that Bezout's theorem states that two surfaces of degree m and n respectively intersect in a curve of degree mn. Thus, two bicubic patches generally intersect in a curve of degree 324.

We have noted that bilinear patches have an implicit equation of degree 2; quadratic patches have an implicit equation of degree 4; biquadratic patches have an implicit equation of degree 8, etc. It seems highly curious that there are gaps in this sequence of degrees. Are there no parametric surfaces whose implicit equation is degree 3 or 5 for example? It turns out that parametric surfaces of degree n only *generally* have implicit equations of degree n^2 and that under certain conditions that degree will decrease. To understand the nature of those conditions, we must understand why the implicit equation of a parametric surface is generally n^2. The *degree* of a surface can be thought of either as the degree of its implicit equation, or as the number of times it is intersected by a line. Thus, the degree of the implicit equation of a parametric surface can be found by determining the number of times it is intersected by a line. Consider a parametric surface given by

$$x = \frac{x(s,t)}{w(s,t)} \quad y = \frac{y(s,t)}{w(s,t)} \quad z = \frac{z(s,t)}{w(s,t)},$$

where the polynomials are of degree n. One way we can compute the points at which a line intersects the surface is by intersecting the surface with two planes which contain the line. If one plane is $Ax + By + Cz + Dw = 0$, its intersection with the surface is a curve of degree n in s, t space: $Ax(s,t) + By(s,t) + Cz(s,t) + Dw(s,t) = 0$. The second plane will also intersect the surface in a degree n curve in parameter space. The points at which these two section curves intersect will be the points at which the line intersects the surface. According the Bezout's theorem, two curves of degree n intersect in n^2 points. Thus, the surface is generally of degree n^2.

17.10 Base Points

It may happen that there are values of s, t for which $x(s,t) = y(s,t) = z(s,t) = w(s,t) = 0$. These are known as *base points* in contemporary algebraic geometry. If a base point exists, any plane section curve will contain it. Therefore, it will belong to the set of intersection points of any pair of section curves. However, since a base point maps to something that is undefined in x, y, z space, it does not represent a point at which the straight line intersects the surface, and thus *the existence of a base point diminishes the degree of the implicit equation by one.* Thus, if there happen to be r base points on a degree n parametric surface, the degree of its implicit equation is $n^2 - r$.

For example, it is well known that any quadric surface can be expressed in terms of degree 2 parametric equations. However, in general, a degree 2 parametric surface has an implicit equation of degree 4, and we conclude that there must be two base points. Consider the parametric equations of a sphere of radius r centered at the origin:

$$\begin{aligned} x &= 2rs^u \\ y &= 2rtu \\ z &= r(u^2 - s^2 - t^2) \\ w &= s^2 + t^2 + u^2 \end{aligned}$$

Homogeneous parameters s, t, u are used to enable us to verify the existence of the two base points: $s = 1,\ t = i,\ u = 0$ and $s = 1,\ t = -i,\ u = 0$.

17.11 Ideals and Varieties

This section presents a brief overview of ideals and varieties and suggests some ways how these topics fit into CAGD. An excellent treatment of ideals and varieties and their application to CAGD can be found in [CLO92].

17.11.1 Ideals of Integers

An ideal I of integers is an infinite set of integers such that, if $a, b \in I$, then $a + b \in I$ and if c is any integer, then $c \cdot a \in I$. Ideals can be defined by a set of "generators" as follows. If $\{i_1, \ldots, i_n\}$ is a set of integers, then we denote by

$$I = \langle i_1, i_2, \ldots, i_n \rangle$$

the ideal generated by $\{i_1, \ldots, i_n\}$. This means that I is the infinite set of all integers that can be expressed as $c_1 i_1 + c_2 i_2 + \ldots + c_n i_n$ where the c_j are integers. We refer to $\{i_1, \ldots, i_n\}$ as a generating set of I.

For example, consider the ideal $I = \langle 6, 9 \rangle$. Obviously, all members of the generating set belong to the ideal: in this case, $6, 9 \in I$. Also, zero belongs to every ideal. For this ideal, we can see that $21 \in I$, because $21 = 6 \cdot 2 + 9 \cdot 1$, and $3 \in I$, because $3 = 2 \cdot 6 - 1 \cdot 9$.

A powerful concept in ideal theory is that every ideal has more than one set of generators.
Theorem: Two ideals $A = \langle i_1, \ldots, i_n \rangle$ and $B = \langle j_1, \ldots, j_m \rangle$ are equivalent if and only if

$$i_k \in B, k = 1, \ldots, n \quad \text{and} \quad j_k \in A, \ k = 1, \ldots, m.$$

The proof is straightforward.

From this theorem, we see that $\langle 6, 9 \rangle = \langle 12, 15 \rangle$ because $6, 9 \in \langle 12, 15 \rangle$ and $12, 15 \in \langle 6, 9 \rangle$. We also see that $\langle 6, 9 \rangle = \langle 3 \rangle$! An ideal that can be generated by a single generator is called a principal ideal. All ideals of integers are principal ideals. Furthermore, the single generator of $A = \langle i_1, \ldots, i_n \rangle$ is the greatest common divisor of $i_1, \ldots, i_n$.

17.11.2 Ideals of Polynomials in One Variable

Given a set of polynomials in t with real coefficients $\{f_1(t), f_2(t), \ldots f_n(t)\}$,

$$I = \langle f_1(t), f_2(t), \ldots, f_n(t) \rangle$$

is the ideal generated by $\{f_1(t), f_2(t), \ldots f_n(t)\}$ and is defined as the infinite set of polynomials in t that can be created as $f_1(t)g_1(t) + f_2(t)g_2(t) + \ldots + f_n(t)g_n(t)$ where the $g_i(t)$ are any polynomials in t with real coefficients. All ideals of polynomials in one variable are principle ideals, and the single generator is the GCD of all polynomials in the ideal.

17.11.3 Polynomials in Several Variables

In general, a polynomial in n variables $x_1, \ldots, x_n$ is defined

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{\tau} c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}. \tag{17.21}$$

Each summand $c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ is called a *term*, $x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ is a *monomial*, and c_i is the *coefficient* of the monomial. By convention, any given monomial occurs in at most one term in a polynomial.

$k[x_1, \ldots, x_n]$ signifies the set of all polynomials in the variables $x_1, \ldots, x_n$ whose coefficients belong to a field k. For example, $R[x, y]$ is the set of all polynomials

$$\sum c_i x^{e_{1,i}} y^{e_{2,i}} \tag{17.22}$$

where $c_i \in R$ and $e_{1,i}, e_{2,i} \in \{0, 1, 2, \ldots\}$. Thus, "$f \in R[x, y, z]$" means that f is a polynomial whose variables are x, y and z and whose coefficients are real numbers. All polynomials in this chapter have coefficients that are real numbers.

Term order

It is often useful to list the terms of a polynomial in decreasing order, beginning with the *leading term*. This is done using a *term order* — a way to compare any two distinct terms of a polynomial and declare which is "greater. "

For linear polynomials, term order amounts to merely declaring an order on the variables. For example, the terms of the polynomial

$$2x + 3y - 4z$$

are in proper order if we declare $x > y > z$. If we declare $y > z > x$, the proper order would be $3y - 4z + 2x$. For non-linear polynomials, we begin by declaring an order on the variables, and then we must also choose one of several schemes that decide how the exponents in a polynomial influence term order. One such scheme is called *lexicographical order* (nicknamed *lex*), defined as follows. If the variables of a polynomial are ordered $x_1 > x_2 > \ldots > x_n$, then given two distinct terms $T_i = c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ and $T_j c_j x_1^{e_{1,j}} x_2^{e_{2,j}} \cdots x_n^{e_{n,j}}$, $T_i > T_j$ if

1. $e_{1,i} > e_{1,j}$, or if
2. $e_{1,i} = e_{1,j}$ and $e_{2,i} > e_{2,j}$, or, in general, if
3. $e_{k,i} = e_{k,j}$ for $k = 1, \ldots, m-1$ and $e_{m,i} > e_{m,j}$.

For example, the polynomial

$$3x^2y^2z + 4xy^3z^2 + 5x^3z + 6y^2 + 7xz^3 + 8$$

using lex with $x > y > z$ would be written $5x^3z + 3x^2y^2z + 4xy^3z^2 + 7xz^3 + 6y^2 + 8$ and its leading term is $5x^3z$. Using lex with $z > x > y$ it would be written $7z^3x + 4z^2xy^3 + 5zx^3 + 3zx^2y^2 + 6y^2 + 8$ and the leading term would be $7z^3x$. Or using lex with $y > z > x$ would be written $4y^3z^2x + 3y^2zx^2 + 6y^2 + 7z^3x + 5zx^3 + 8$ and the leading term would be $4y^3z^2x$.

Another choice for term order is the *degree lexicographical order* (abbreviated *deglex*). If the variables are ordered $x_1 > x_2 > \ldots > x_n$, then using deglex, $T_i > T_j$ if

1. $e_{1,i} + e_{2,i} + \ldots + e_{n,i} > e_{1,j} + e_{2,j} + \ldots + e_{n,j}$, or
2. $e_{1,i} + e_{2,i} + \ldots + e_{n,i} = e_{1,j} + e_{2,j} + \ldots + e_{n,j}$ and $T_i > T_j$ with respect to lex.

Using deglex with $x > y > z$, the terms of $3x^2y^2z + 4xy^3z^2 + 5x^3z + 6y^2 + 7xz^3 + 8$ would be ordered $4xy^3z^2 + 3x^2y^2z + 5x^3z + 7xz^3 + 6y^2 + 8$.

As obverved in the lex and deglex examples, term orders ignore the coefficient of a term, so a term order might more properly be called a monomial order.

Other term orders can also be defined, such as degree reverse lexicographical order. The precise requirements for any term order are discussed in reference [AL94], page 18.

The n-dimensional real affine space is denoted R^n and is the set of n-tuples:

$$R^n = (a_1, \ldots, a_n) : a_1, \ldots, a_k \in R \tag{17.23}$$

17.11.4 Polynomial Ideals and Varieties

The *polynomial ideal* generated by $f_1, ..., f_s \in k[x_1, ..., x_n]$, denoted $\langle f_1, ..., f_s \rangle$, is defined

$$I = \langle f_1, ..., f_s \rangle = \{p_1 f_1 + ... + p_s f_s : p_i \in k[x_1, ..., x_n]\}. \tag{17.24}$$

The polynomials $f_1, ..., f_s$ are called *generators* of this ideal. As in integer ideals, any polynomial ideal can be defined using different generating sets.

Consider a set of polynomials $f_1, f_2, \cdots, f_s \in k[x_1, ..., x_n]$. Let $(a_1, ..., a_n)$ be a point in k^n satisfying $f_i(a_1, ..., a_n) = 0$, $i = 1, ..., s$. The set of all such points $(a_1, ..., a_n)$ is called the *variety* defined by $f_1, ..., f_s$, and is denoted by $V(f_1, ..., f_s)$:

$$V(f_1, ..., f_s) = \{(a_1, ..., a_n) \in k^n | f_i(a_1, ..., a_n) = 0, i = 0, \ldots, s\}. \tag{17.25}$$

A more familiar way to refer to a variety is simply the set of solutions to a set of polynomial equations.

A variety defined by a single polynomial—called a *hypersurface*—is the most familiar instance of a variety. A hypersurface in R^2 is a planar curve defined using an implicit equation, and a hypersurface in R^3 is what is normally called an implicit surface in CAGD. For example, $V(x^2 + y^2 - 1)$ is a circle defined in terms of the implicit equation $x^2 + y^2 - 1 = 0$ and $V(2x + 4y - z + 1)$ is the plane whose implicit equation is $2x + 4y - z + 1 = 0$. A variety $V(f_1, \ldots, f_s)$ defined by more than one polynomial ($s > 1$) is the intersection of the varieties $V(f_1) \ldots V(f_s)$.

An ideal is a set of polynomials that is infinite in number. The *variety of an ideal* is the set of all points $(a_1, ..., a_n)$ that make each polynomial in the ideal vanish. It is easy to see that the variety of an ideal is the variety of generating set for the ideal. Two generating sets for the same ideal define the same variety.

This is a very powerful concept, because some generating sets are more useful than others. For example, consider the ideal

$$I = \langle x + y + z - 6, x - y - z, x - 2y + z - 3 \rangle$$

which can also be generated by $\langle x - 3, y - 1, z - 2 \rangle$. This second set of generators is much more helpful, because we can immediately see that $V(I)$ is $x = 3, y = 1, z = 2$. One method for converting the generating set $\langle x + y + z - 6, x - y - z, x - 2y + z - 3 \rangle$ into $\langle x - 3, y - 1, z - 2 \rangle$ is to use the familiar Gauss elimination method.

As another example, consider the ideal

$$I = \langle t^4 - 3t^3 + 4t^2 - t - 4, t^6 - 4t^3 + 3 \rangle$$

An alternative generator for this ideal is simply $\langle t - 1 \rangle$, which can be obtained using Euclid's algorithm. Thus, $t = 1$ is the variety of this ideal, or, the common zero.

As we saw in our discussion of ideals of integers, necessary and sufficient conditions for $\langle f_1, \ldots, f_n \rangle = \langle g_1, \ldots, g_m \rangle$ are $f_1, \ldots, f_n \in \{g_1, \ldots, g_m\}$ and $g_1, \ldots, g_m \in \{f_1, \ldots, f_n\}$. This general process can be used to prove that Gauss elimination and Euclid's algorithm, respectively, are algorithms that create equivalent generating sets for ideals.

17.11.5 Gröbner Bases

Gauss elimination and Euclid's algorithm create generating sets which make it simple to compute the variety of an ideal (or, the set of all solutions to a set of polynomial equations). These algorithms are

special cases of a completely general method for robustly finding all solutions to a set of polynomial equations in any number of variables. This method is based on the notion of *Gröbner bases*.

A *Gröbner basis* of an ideal I is a set of polynomials $\{g_1, \ldots, g_t\}$ such that the leading term of any polynomial in I is divisible by the leading term of at least one of the polynomials $g_1, \ldots, g_t$. This, of course, requires that a term order be fixed for determining the leading terms: different term orders produce different Gröbner bases. Several excellent books have been written on Gröbner bases that do not presuppose that the reader has an advanced degree in mathematics [CLO92, AL94, BW93]

A Gröbner basis is a particularly attractive set of generators for an ideal, as illustrated by two familiar examples. If $\{f_1, \ldots, f_s\}$ are polynomials in one variable, the Gröbner basis of $\langle f_1, \ldots, f_n \rangle$ consists of a single polynomial: the GCD of $f_1, \ldots, f_s$. If $\{f_1, \ldots, f_s\}$ are linear polynomials in several variables, the Gröbner basis is an uppertriangular form of a set of linear equations. The Gröbner basis of these special cases provides significant computational advantage and greater insight, and the same is true of the Gröbner basis of a more general ideal.

Gröbner bases are the fruit of Bruno Buchberger's Ph.D. thesis [Buc65], and are named in honor of his thesis advisor. Buchberger devised an algorithm for computing Gröbner bases [Buc85, CLO92]. Also, commercial software packages such as Maple and Mathematica include capabilities for computing Gröbner bases.

Chapter 18

Implicitization using Moving Lines

This chapter presents a new way of looking at implicitization that has some geometric meaning. It also shows how to express the determinant in more compact form, how to locate the double point of a Béziercurve, and discusses the concept of duality.

18.1 Definition

A pencil of lines can be described by the equation

$$(a_0x + b_0y + c_0)(1 - t) + (a_1x + b_1y + c_1)t = 0 \tag{18.1}$$

where the equations $a_0x + b_0y + c_0 = 0$ and $a_1x + b_1y + c_1 = 0$ define any two distinct lines.

It is known that any conic section can be "generated by the intersection of corresponding lines of two related pencils in a plane" [Som51], p.388. In other words, given two distinct pencils, $(a_{00}x+b_{00}y+c_{00})(1-t)+(a_{10}x+b_{10}y+c_{10})t = 0$ and $(a_{01}x+b_{01}y+c_{01})(1-t)+(a_{11}x+b_{11}y+c_{11})t = 0$, to each value of t corresponds exactly one line from each pencil, and those two lines intersect in a point. The locus of points thus created for $-\infty \le t \le \infty$ is a conic section, as illustrated in Figure 18.1. This is reviewed in section 18.2.

This chapter examines the extension of that idea to higher degrees. A degree n family of lines intersects a degree m family of lines in a curve of degree $m + n$, which is discussed in section 18.3. Section 18.4 shows that *any* rational curve can be described as the intersection of two families of lines, from which the multiple points and the implicit equation of the curve can be easily obtained. For example, any cubic rational curve can be described as the intersection of a pencil of lines and a quadratic family of lines. The pencil axis lies at the double point of the cubic curve. Section 18.5 discusses the family of lines which is tangent to a given rational curve. Such families of lines are useful for analyzing the singularities of the curve, such as double points, cusps, and inflection points, and also for calculating derivative directions.

18.1.1 Homogeneous Points and Lines

In projective geometry, the point whose homogeneous coordinates are (X, Y, W) has Cartesian coordinates $(x, y) = (X/W, Y/W)$. Of course, X, Y, and W cannot all be zero. The equation of a line in homogenous form is

$$aX + bY + cW = 0 \tag{18.2}$$

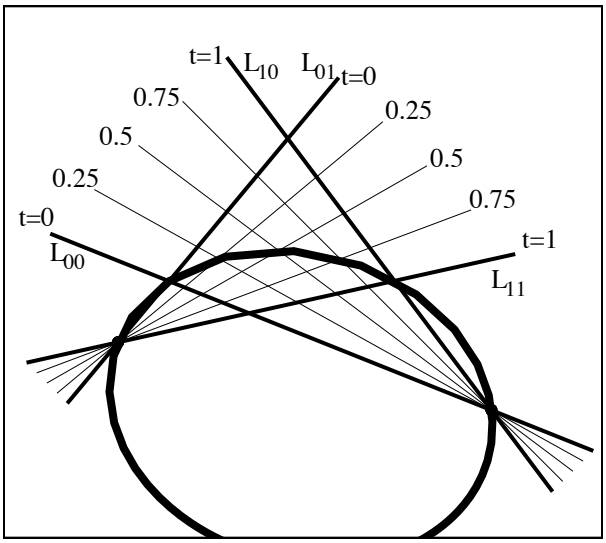


Figure 18.1: Intersection of Two Pencils of Lines

(a, b, and c not all zero).

Given two triples, (a, b, c) and (d, e, f), the operation of cross product is defined

$$(a, b, c) \times (d, e, f) = (bf - ec, dc - af, ae - db) \tag{18.3}$$

and the dot product is defined

$$(a, b, c) \cdot (d, e, f) = ad + be + cf. \tag{18.4}$$

In this chapter, all single characters in bold typeface signify a triple. In particular, the symbol $\mathbf{P}$ denotes the triple (X, Y, W), and $\mathbf{L}$ symbolizes the triple (a, b, c). When we refer to "the line $\mathbf{L}$", we mean

$$\{(X, Y, W) | \mathbf{L} \cdot \mathbf{P} = (a, b, c) \cdot (X, Y, W) = aX + bY + cW = 0\}. \tag{18.5}$$

Thus, we can say that a point $\mathbf{P}$ lies on a line $\mathbf{L} = (a, b, c)$ if and only if $\mathbf{P} \cdot \mathbf{L} = 0$.

The cross product has the following two applications: The line $\mathbf{L}$ containing two points $\mathbf{P}_1 = (X_1, Y_1, W_1)$ and $\mathbf{P}_2 = (X_2, Y_2, W_2)$ is given by

$$\mathbf{L} = \mathbf{P}_1 \times \mathbf{P}_2. \tag{18.6}$$

The point $\mathbf{P}$ at which two lines $\mathbf{L}_1 = (a_1, b_1, c_1)$ and $\mathbf{L}_2 = (a_2, b_2, c_2)$ intersect is given by

$$\mathbf{P} = \mathbf{L}_1 \times \mathbf{L}_2. \tag{18.7}$$

This illustrates the principle of *duality*. Loosely speaking, general statements involving points and lines can be expressed in a reciprocal way. For example, the statement "a unique line passes through two distinct points" has a dual expression, "a unique point lies at the intersection of two distinct lines".

In general, any triple (α, β, γ) can be interpreted as a line $(\{(X, Y, W) | = \alpha X + \beta Y + \gamma W = 0\})$ or as a point (whose Cartesian coordinates are $(\frac{\alpha}{\gamma}, \frac{\beta}{\gamma})$). To remove the ambiguity, triples which are

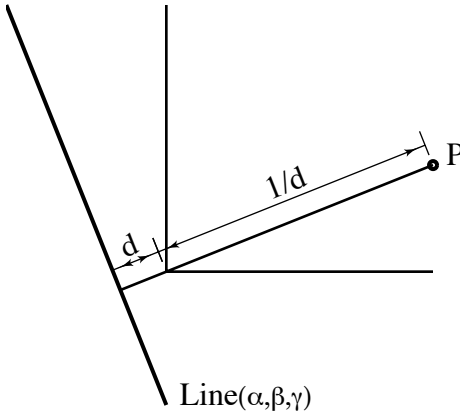


Figure 18.2: Dual Point and Line

to be interpreted as points are preceded by the prefix 'Point' (*i.e.* Point(α, β, γ)), and triples which signify lines are preceded by 'Line'. The *dual* of Line(α, β, γ) is Point(α, β, γ), and vice versa. Dual points and lines are related geometrically as follows:

- If $d = \frac{\gamma}{\sqrt{\alpha^2+\beta^2}}$ is the distance from Line(α, β, γ) to the origin, then $1/d$ is the distance from Point(α, β, γ) to the origin.
- The line from the origin to Point(α, β, γ) is perpendicular to Line(α, β, γ).

This relationship is illustrated in Figure 18.2.

Discussion of the duals of Bézier and B-spline curves can be found in [Hos83, Sab87].

18.1.2 Curves and Moving Lines

A homogeneous point whose coordinates are functions of a variable is denoted by appending the variable, enclosed in square brackets:

$$\mathbf{P}[t] = (X[t], Y[t], W[t]) \tag{18.8}$$

which amounts to the rational curve

$$x = \frac{X[t]}{W[t]}; \qquad y = \frac{Y[t]}{W[t]}. \tag{18.9}$$

If these functions are polynomials in Bernstein form

$$X[t] = \sum_{i=0}^{n} X_i B_i^n[t]; \quad Y[t] = \sum_{i=0}^{n} Y_i B_i^n[t]; \quad W[t] = \sum_{i=0}^{n} W_i B_i^n[t] \tag{18.10}$$

with $B_i^n[t] = \binom{n}{i}(1-t)^{n-i}t^i$, then equation (18.8) defines a rational Bézier curve

$$\mathbf{P}[t] = \sum_{i=0}^{n} \mathbf{P}_i B_i^n[t] \tag{18.11}$$

with homogeneous control points $\mathbf{P}_i = (X_i, Y_i, W_i)$. Customarily, these control points are thought of as having Cartesian coordinates $(\frac{X_i}{W_i}, \frac{Y_i}{W_i})$ with weights W_i. If $W[t] \equiv \gamma$, that is, if $W[t]$ is a constant, then the curve is referred to as a *polynomial curve*. This happens if $W_0 = W_1 = \ldots = W_n = \gamma$. In this chapter, the word *curve* generally means *rational curve* unless it is referred to as polynomial curve.

Likewise,

$$\mathbf{L}[t] = (a[t], b[t], c[t]) \tag{18.12}$$

denotes the family of lines

$$a[t]x + b[t]y + c[t] = 0. \tag{18.13}$$

Such a family of lines is traditionally known as a *pencil* of lines if $a[t]$, $b[t]$, and $c[t]$ are linear functions of t. In the general case, [Win23] refers to families of lines as *line equations*, and families of points (*i.e.*, curves) as *point equations*. In this chapter, it is more comfortable to refer to a parametric family of lines as in equation (18.12) as a *moving line*. Also, a curve $\mathbf{P}[t]$ will be referred to as a *moving point* when the varying position of point $\mathbf{P}[t]$ is of interest.

A moving point $\mathbf{P}[t]$ *follows* a moving line $\mathbf{L}[t]$ (or, equivalently, the moving line *follows* the moving point) if

$$\mathbf{P}[t] \cdot \mathbf{L}[t] \equiv 0, \tag{18.14}$$

that is, if point $\mathbf{P}[t]$ lies on line $\mathbf{L}[t]$ for all values of t. Two moving points $\mathbf{P}_1[t]$ and $\mathbf{P}_2[t]$ follow a moving line $\mathbf{L}[t]$ if

$$\mathbf{P}_1[t] \times \mathbf{P}_2[t] \equiv \mathbf{L}[t]k[t] \tag{18.15}$$

where $k[t]$ is a scalar rational function of degree zero or greater. Two moving lines $\mathbf{L}_1[t]$ and $\mathbf{L}_2[t]$ *intersect* at a moving point $\mathbf{P}[t]$ if

$$\mathbf{L}_1[t] \times \mathbf{L}_2[t] \equiv \mathbf{P}[t]k[t]. \tag{18.16}$$

18.1.3 Weights and Equivalency

Two homogeneous points (X_1, Y_1, W_1) and (X_2, Y_2, W_2) represent the same point in Cartesian coordinates $(X_1/W_1, Y_1/W_1) = (X_2/W_2, Y_2/W_2)$ if and only if $(X_1, Y_1, W_1) = k(X_2, Y_2, W_2)$ where k is a non-zero constant. Likewise, lines $\mathbf{L}_1$ and $\mathbf{L}_2$ are the same if and only if $\mathbf{L}_1 = k\mathbf{L}_2$. This fact can be extended to curves and moving lines. For example, two curves $\mathbf{P}_1[t]$ and $\mathbf{P}_2[t]$ are equivalent if $\mathbf{P}_1[t] \equiv k[t]\mathbf{P}_2[t]$ where $k[t]$ is a scalar rational function of t. Two curves with identical shape are not equivalent if the parametrizations are different each other.

Even though two homogeneous points (or lines) may map to the same Cartesian point (or line), they do not always create identical results. For example, a control point $\mathbf{P}_i$ of a rational curve as in equation (18.11) cannot be replaced with a scale of itself without altering the curve, for scaling would change its weight. Of course, if all the control points were scaled by the same value, the curve would be unchanged.

18.2 Pencils and Quadratic Curves

18.2.1 Pencils of lines

Given any two lines $\mathbf{L}_0 = (a_0, b_0, c_0)$ and $\mathbf{L}_1 = (a_1, b_1, c_1)$, a pencil of lines $\mathbf{L}[t]$ can be expressed

$$\mathbf{L}[t] = \mathbf{L}_0(1 - t) + \mathbf{L}_1 t. \tag{18.17}$$

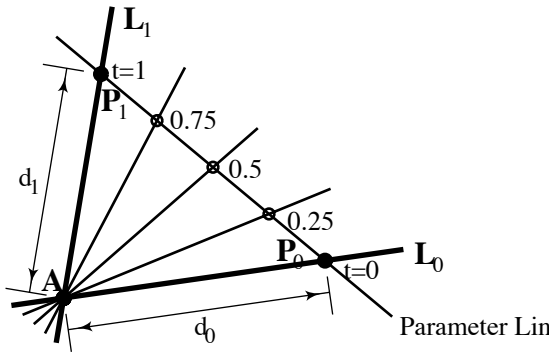


Figure 18.3: Pencil of Lines

All lines in the pencil contain the point at which $\mathbf{L}_0$ and $\mathbf{L}_1$ intersect. We will refer to this point as the pencil *axis*. There is a one-to-one correspondence between lines in the pencil and parameter values t.

The rate at which the moving line $\mathbf{L}[t]$ rotates about its axis as t varies is most easily visualized by introducing a *parameter line* as shown in Figure 18.3. The parameter line is a degree 1 polynomial Bézier curve

$$\mathbf{P}[t] = (1-t)\mathbf{P}_0 + t\mathbf{P} \tag{18.18}$$

which follows $\mathbf{L}[t]$. $\mathbf{P}_0$ lies on $\mathbf{L}_0$ a distance d_0 from the axis, and $\mathbf{P}_1$ lies on $\mathbf{L}_1$ a distance d_1 from the axis where

$$\frac{d_0}{d_1} = \frac{\sqrt{a_0^2 + b_0^2}}{\sqrt{a_1^2 + b_1^2}}. \tag{18.19}$$

If $\mathbf{L}[t]$ is to rotate counter clockwise as t increases, then

$$a_0 b_1 > a_1 b_0. \tag{18.20}$$

If parameter values are marked off evenly along the parameter line as shown, the line $\mathbf{L}[t]$ passes through the parameter line at the point corresponding to t.

Notice that d_0 and d_1 control the rate at which $\mathbf{L}[t]$ rotates. Figure 18.4 shows an example in which d_0 is larger than d_1, with the effect that lines defined by evenly spaced increments are concentrated near $\mathbf{L}_0$.

An alternate way to specify a pencil of lines is with an axis $\mathbf{P}_A = (X_A, Y_A, W_A)$ and a rational linear Bézier curve

$$\mathbf{P}[t] = \mathbf{P}_0(1-t) + \mathbf{P}_1 t = (X_0, Y_0, W_0)(1-t) + (X_1, Y_1, W_1)t. \tag{18.21}$$

A pencil $\mathbf{L}[t]$ can then be defined as the set of lines connecting $\mathbf{P}_A$ with points on $\mathbf{P}[t]$:

$$\mathbf{L}[t] = \mathbf{P}_A \times \mathbf{P}[t] = \mathbf{P}_A \times (\mathbf{P}_0(1-t) + \mathbf{P}_1 t). \tag{18.22}$$

This representation of a pencil is related to the representation in equation (18.17). The two lines $\mathbf{L}_0$ and $\mathbf{L}_1$ can be expressed

$$\begin{aligned} \mathbf{L}_0 &= \mathbf{P}_A \times \mathbf{P}_0 \\ \mathbf{L}_1 &= \mathbf{P}_A \times \mathbf{P}_1 \end{aligned} \tag{18.23}$$

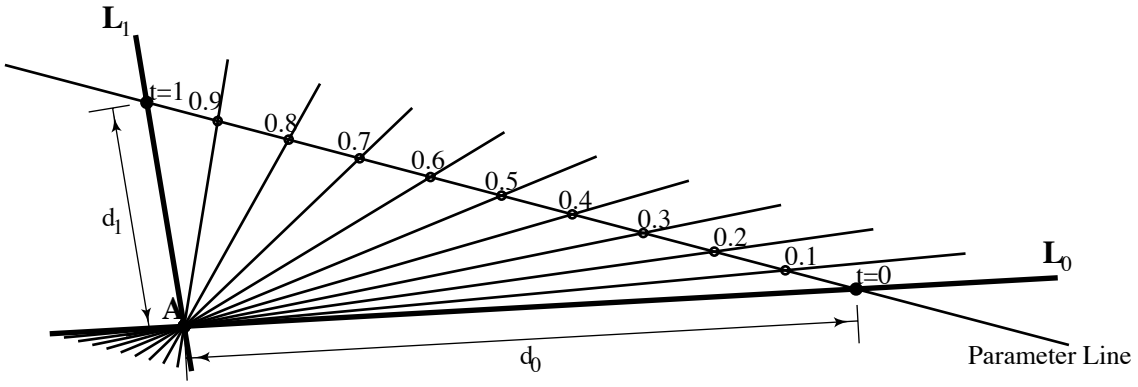


Figure 18.4: Pencil of Lines

and the parameter line is positioned with distances d_0 and d_1 which satisfy

$$\frac{d_0}{d_1} = \frac{\overline{\mathbf{P}_A\mathbf{P}_0}W_0}{\overline{\mathbf{P}_A\mathbf{P}_1}W_1}, \tag{18.24}$$

as shown in Figure 18.5. Thus, we see that every pencil of lines follows a degree one polynomial curve.

18.2.2 Intersection of Two Pencils

Consider what happens when two pencils of lines intersect, line by line. Figure 18.6 shows two pencils

$$\begin{aligned} \mathbf{L}_0[t] &= \mathbf{L}_{00}(1-t) + \mathbf{L}_{01}t \\ \mathbf{L}_1[t] &= \mathbf{L}_{10}(1-t) + \mathbf{L}_{11}t, \end{aligned} \tag{18.25}$$

and the points at which they intersect for parameter values $t = 0, .25, .5, .75, 1$. It is clear that those five sample points are not collinear, and in fact as t varies continuously from 0 to 1, the two pencils intersect in a smooth curve as shown. This curve turns out to be a conic section, which can be expressed as a rational Bézier curve $\mathbf{P}[t]$ as follows.

$$\begin{aligned} \mathbf{P}[t] &= \mathbf{L}_0[t] \times \mathbf{L}_1[t] \\ &= (\mathbf{L}_{00}(1-t) + \mathbf{L}_{01}t) \times (\mathbf{L}_{10}(1-t) + \mathbf{L}_{11}t) \\ &= (\mathbf{L}_{00} \times \mathbf{L}_{10})(1-t)^2 + \frac{1}{2}(\mathbf{L}_{00} \times \mathbf{L}_{11} + \mathbf{L}_{01} \times \mathbf{L}_{10})2(1-t)t + (\mathbf{L}_{01} \times \mathbf{L}_{11})t^2, \end{aligned} \tag{18.26}$$

which expresses $\mathbf{P}$[t] as a quadratic rational Bézier curve whose control points are:

$$\begin{aligned} \mathbf{P}_0 &= \mathbf{L}_{00} \times \mathbf{L}_{10} \\ \mathbf{P}_1 &= \frac{1}{2}(\mathbf{L}_{00} \times \mathbf{L}_{11} + \mathbf{L}_{01} \times \mathbf{L}_{10}) \\ \mathbf{P}_2 &= \mathbf{L}_{01} \times \mathbf{L}_{11}. \end{aligned} \tag{18.27}$$

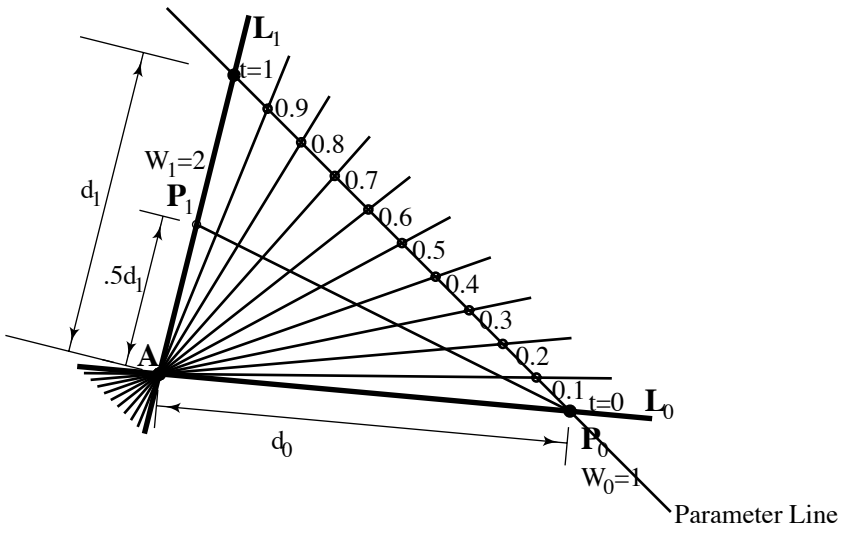


Figure 18.5: Pencil of Lines

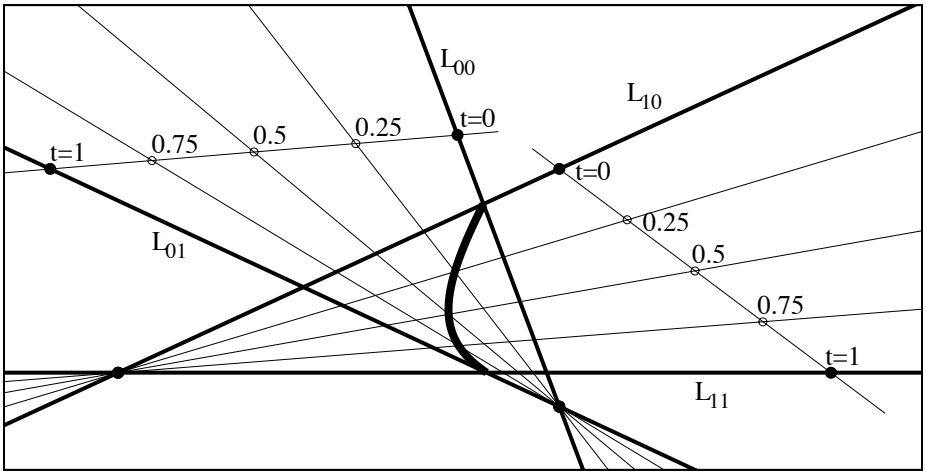


Figure 18.6: Intersection of Two Pencils of Lines

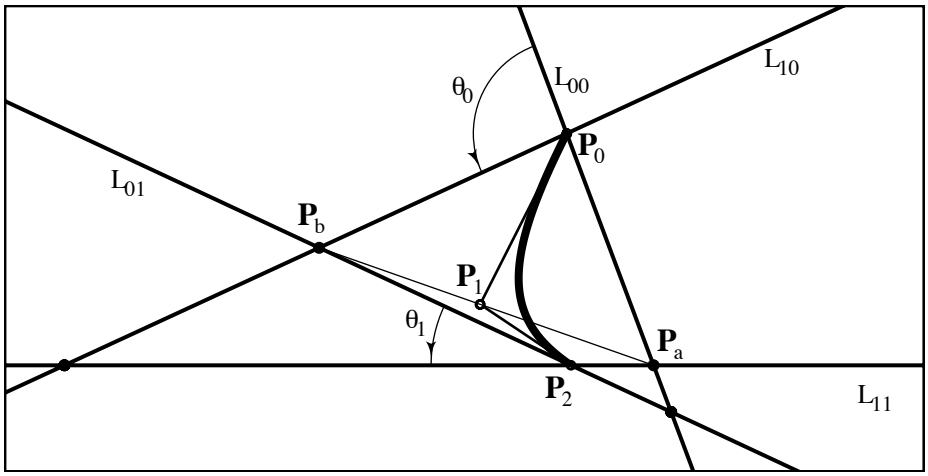


Figure 18.7: Rational Quadratic Curve

The geometric relation between the control points and weights are shown in Figure 18.7. Control point $\mathbf{P}_0$ lies at the intersection of $\mathbf{L}_{00}$ and $\mathbf{L}_{10}$ and has a weight of $d_{00}d_{10}\sin(\theta_0)$, where $d_{ij} = \sqrt{a_{ij}^2 + b_{ij}^2}$. Control point $\mathbf{P}_2$ lies at the intersection of $\mathbf{L}_{01}$ and $\mathbf{L}_{11}$ and has a weight of $d_{01}d_{11}\sin(\theta_1)$. Control point $\mathbf{P}_1$ and its weight is computed using the auxiliary points $\mathbf{P}_a$ and $\mathbf{P}_b$ which are the intersection of $\mathbf{L}_{00}$ and $\mathbf{L}_{11}$, and $\mathbf{L}_{01}$ and $\mathbf{L}_{10}$, respectively.

The two pencils of lines provide a useful intermediate representation of a conic section, from which the implicit equation and parametric equations can be derived with equal ease. The implicit equation can be found by eliminating the pencil parameter t from the two pencil equations

$$\begin{aligned} L_{00}(1-t) + L_{10}t &= 0 \\ L_{10}(1-t) + L_{11}t &= 0 \end{aligned} \tag{18.28}$$

where

$$L_{ij} = \mathbf{L}_{ij} \cdot \mathbf{P} = a_{ij}X + b_{ij}Y + c_{ij}W. \tag{18.29}$$

In matrix form,

$$\begin{bmatrix} L_{00} & L_{01} \\ L_{10} & L_{11} \end{bmatrix} \begin{Bmatrix} 1-t \\ t \end{Bmatrix} = 0 \tag{18.30}$$

from which the implicit equation of the intersection locus is

$$L_{00}L_{11} - L_{01}L_{10} = 0. \tag{18.31}$$

18.2.3 Pencils on Quadratic Curves

An arbitrary rational quadratic Bézier curve

$$\mathbf{P}[t] = \mathbf{P}_0(1-t)^2 + \mathbf{P}_1 2(1-t)t + \mathbf{P}_2 t^2 \tag{18.32}$$

can be represented as the intersection of two pencils. Consider a moving line $\mathbf{L}[t]$ which goes through a certain fixed point $\mathbf{P}[k]$ on the curve and follows the moving point $\mathbf{P}[t]$. This moving line

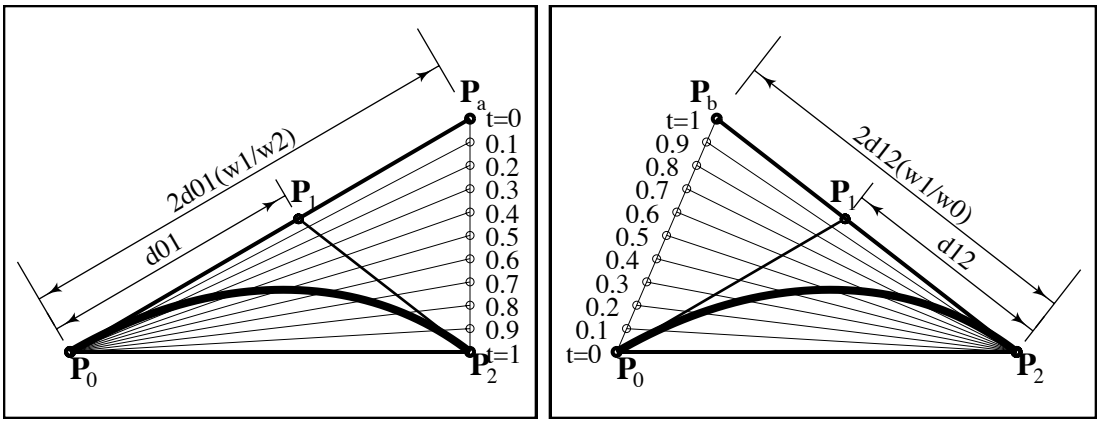


Figure 18.8: Quadratic Bézier Curve

is expressed as follows.

$$\begin{aligned}
\mathbf{L}[t] &= \mathbf{P}[k] \times \mathbf{P}[t] \\
&= (\mathbf{P}_0(1-k)^2 + \mathbf{P}_1 2(1-k)k + \mathbf{P}_2 k^2) \times (\mathbf{P}_0(1-t)^2 + \mathbf{P}_1 2(1-t)t + \mathbf{P}_2 t^2) \\
&= (t-k) \left\{ \begin{array}{cc} 1-k & k \end{array} \right\} \left[\begin{array}{cc} 2\mathbf{P}_0 \times \mathbf{P}_1 & \mathbf{P}_0 \times \mathbf{P}_2 \\ \mathbf{P}_0 \times \mathbf{P}_2 & 2\mathbf{P}_1 \times \mathbf{P}_2 \end{array} \right] \left\{ \begin{array}{c} 1-t \\ t \end{array} \right\}, \qquad (18.33)
\end{aligned}$$

from which this $\mathbf{L}[t]$ is always equivalent to a linear moving line (*i.e.* a pencil). Thus, two pencils can be found by choosing arbitrary two parameter values k_0 and k_1 and calculating $\mathbf{P}[k_i] \times \mathbf{P}[t]$. The simplest example is $k_0 = 0$, $k_1 = 1$. In this case,

$$\begin{aligned}
\mathbf{P}[0] \times \mathbf{P}[t] &= t\left[\mathbf{P}_0 \times (2\mathbf{P}_1(1-t) + \mathbf{P}_2 t)\right] \\
\mathbf{P}[1] \times \mathbf{P}[t] &= (1-t)\left[\mathbf{P}_2 \times (\mathbf{P}_0(1-t) + 2\mathbf{P}_1 t)\right], \qquad (18.34)
\end{aligned}$$

from which the two pencils intersecting at the moving point $\mathbf{P}[t]$ are

$$\begin{aligned}
\mathbf{L}_0 &= \mathbf{P}_0 \times (2\mathbf{P}_1(1-t) + \mathbf{P}_2 t) \\
\mathbf{L}_1 &= \mathbf{P}_2 \times (\mathbf{P}_0(1-t) + 2\mathbf{P}_1 t). \qquad (18.35)
\end{aligned}$$

Figure 18.8 shows the geometric relationship. The parameter lines of pencils $\mathbf{L}_0[t]$ and $\mathbf{L}_1[t]$ are $\mathbf{P}_a\mathbf{P}_2$ and $\mathbf{P}_0\mathbf{P}_b$, respectively, where

$$\begin{aligned}
\overline{\mathbf{P}_0\mathbf{P}_a} &= \frac{2w_1}{w_2}\overline{\mathbf{P}_0\mathbf{P}_1}, \\
\overline{\mathbf{P}_2\mathbf{P}_b} &= \frac{2w_1}{w_0}\overline{\mathbf{P}_2\mathbf{P}_1}. \qquad (18.36)
\end{aligned}$$

These parameter lines have the property of a *nomogram*. Consider a point $\mathbf{P}_p$ on the curve. If the line $\mathbf{P}_0\mathbf{P}_p$ intersects the parameter line $\mathbf{P}_a\mathbf{P}_2$ at the point $\mathbf{P}_q$, then the parameter value t at $\mathbf{P}_p$ can be found as

$$t = \frac{\overline{\mathbf{P}_a\mathbf{P}_q}}{\overline{\mathbf{P}_a\mathbf{P}_2}}. \qquad (18.37)$$

18.3 Moving Lines

18.3.1 Bernstein Form

In equation (18.12), if the polynomials $a[t]$, $b[t]$, and $c[t]$ are of degree n, one way to define the moving line $\mathbf{L}[t]$ is to use $n+1$ *control lines* $\mathbf{L}_i$ where

$$\mathbf{L}[t] = \sum_{i=0}^{n} \mathbf{L}_i B_i^n[t]. \tag{18.38}$$

Since this is the dual of a Bézier curve, it has many nice properties. For example, $\mathbf{L}[t]$ moves from $\mathbf{L}_0$ to $\mathbf{L}_n$ when t is changing from 0 to 1. The line $\mathbf{L}[\tau]$ for any parameter value τ can be calculated using the de Casteljau algorithm. Denote

$$\mathbf{L}_i^{<k>}[\tau] = (1-\tau)\mathbf{L}_i^{<k-1>}[\tau] + \tau\mathbf{L}_{i+1}^{<k-1>}[\tau] \tag{18.39}$$

where

$$\mathbf{L}_i^{<0>}[\tau] = \mathbf{L}_i. \tag{18.40}$$

Then, $\mathbf{L}_0^{<n>}[\tau]$ is the required line $\mathbf{L}[\tau]$. This algorithm also *subdivides* the moving line in the same way as for a Bézier curve.

18.3.2 Moving Line which Follows Two Moving Points

Consider two moving points $\mathbf{P}[t]$ and $\mathbf{Q}[t]$ of degree m and n respectively. The moving line $\mathbf{L}[t]$ that follows the two moving points is

$$\mathbf{L}[t] = \mathbf{P}[t] \times \mathbf{Q}[t], \tag{18.41}$$

and its degree is generally $m+n$. If $\mathbf{P}[t]$ and $\mathbf{Q}[t]$ are Bézier curves

$$\begin{aligned} \mathbf{P}[t] &= \sum_{i=0}^{m} B_i^m[t]\mathbf{P}_i \\ \mathbf{Q}[t] &= \sum_{j=0}^{n} B_j^n[t]\mathbf{Q}_j, \end{aligned} \tag{18.42}$$

then the control lines $\mathbf{L}_k$ of the moving line are expressed as follows:

$$\mathbf{L}[t] = \sum_{k=0}^{m+n} B_k^{m+n}[t]\mathbf{L}_k \tag{18.43}$$

$$\mathbf{L}_k = \frac{1}{\binom{m+n}{k}} \sum_{i+j=k} \binom{m}{i}\binom{n}{j} \mathbf{P}_i \times \mathbf{Q}_j. \tag{18.44}$$

For example, if a moving line $\mathbf{L}[t]$ follows a degree one moving point

$$\mathbf{P}[t] = \mathbf{P}_0(1-t) + \mathbf{P}_1 t \tag{18.45}$$

and a degree two moving point

$$\mathbf{Q}[t] = \mathbf{Q}_0(1-t)^2 + \mathbf{Q}_1 2(1-t)t + \mathbf{Q}_2 t^2, \tag{18.46}$$

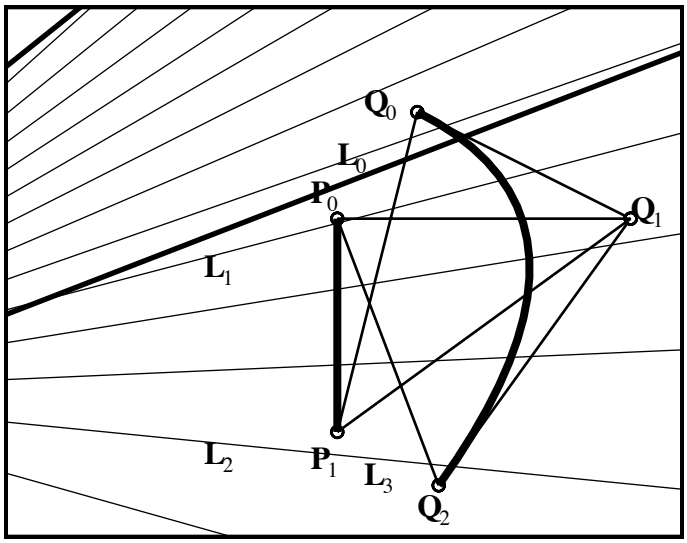


Figure 18.9: Cubic Moving Line which Follow Linear and Quadratic Moving Points

then $\mathbf{L}[t]$ is a cubic moving line with control lines

$$\begin{aligned}
\mathbf{L}_0 &= \mathbf{P}_0 \times \mathbf{Q}_0 \\
\mathbf{L}_1 &= \frac{1}{3}(2\mathbf{P}_0 \times \mathbf{Q}_1 + \mathbf{P}_1 \times \mathbf{Q}_0) \\
\mathbf{L}_2 &= \frac{1}{3}(\mathbf{P}_0 \times \mathbf{Q}_2 + 2\mathbf{P}_1 \times \mathbf{Q}_1) \\
\mathbf{L}_3 &= \mathbf{P}_1 \times \mathbf{Q}_2.
\end{aligned} \tag{18.47}$$

The relationship between the control lines and the control points is shown in Figure 18.9.

18.3.3 Intersection of Two Moving Lines

Two moving lines $\mathbf{L}_0[t]$ and $\mathbf{L}_1[t]$ of degree m and n respectively, intersect at a moving point

$$\mathbf{P}[t] = \mathbf{L}_0[t] \times \mathbf{L}_1[t], \tag{18.48}$$

whose degree is generally $m+n$. This is the dual of what we discussed in section 18.3.2. Therefore, the relationship between the control lines of $\mathbf{L}_0[t]$ and $\mathbf{L}_1[t]$ and the control points of $\mathbf{P}[t]$ is

$$\mathbf{P}_k = \frac{1}{\binom{m+n}{k}} \sum_{i+j=k} \binom{m}{i} \binom{n}{j} \mathbf{L}_{0i} \times \mathbf{L}_{1j}. \tag{18.49}$$

Figure 18.10 shows an example of cubic moving point $\mathbf{P}[t]$ which is the intersection of a pencil $\mathbf{L}_0[t]$ and a quadratic moving line $\mathbf{L}_1[t]$. Notice that each control point of $\mathbf{P}[t]$ can be calculated

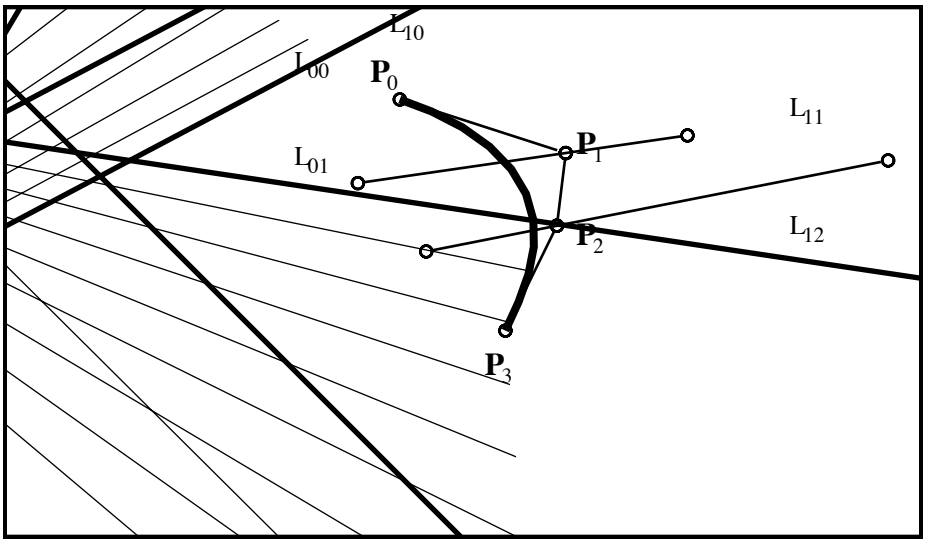


Figure 18.10: Intersection of Linear and Quadratic Moving Lines

from intersection(s) of the control lines as follows:

$$\begin{aligned}
\mathbf{P}_0 &= \mathbf{L}_{00} \times \mathbf{L}_{10} \\
\mathbf{P}_1 &= \frac{1}{3}(2\mathbf{L}_{00} \times \mathbf{L}_{11} + \mathbf{L}_{01} \times \mathbf{L}_{10}) \\
\mathbf{P}_2 &= \frac{1}{3}(\mathbf{L}_{00} \times \mathbf{L}_{12} + 2\mathbf{L}_{01} \times \mathbf{L}_{11}) \\
\mathbf{P}_3 &= \mathbf{L}_{01} \times \mathbf{L}_{12}.
\end{aligned} \tag{18.50}$$

18.3.4 Base Points

Any parameter value t for which $\mathbf{P}[t] = (0,0,0)$ or $\mathbf{L}[t] = (0,0,0)$ is called a *base point*. Any curve or moving line which has a base point can be replaced by an equivalent curve or moving line of degree one less. For example, if for some parameter value $t = \tau$, $\mathbf{P}[\tau] = (0,0,0)$, then $t - \tau$ would be a factor of $X[t]$, $Y[t]$, and $W[t]$ and we could divide out that factor to get

$$\mathbf{P}[t] = (t-\tau)(\frac{X[t]}{t-\tau}, \frac{Y[t]}{t-\tau}, \frac{W[t]}{t-\tau}). \tag{18.51}$$

In equation (18.41), assume $\mathbf{P}$ and $\mathbf{Q}$ have no base points themselves. Then, the degree of $\mathbf{L}[t]$ is generally $m+n$. The degree will be less than $m+n$ only if $\mathbf{L}$ has a base point. This can only happen if there exists a value τ such that $\mathbf{P}[\tau] \times \mathbf{Q}[\tau] = (0,0,0)$, which can only happen if at one of the $m \times n$ points at which $\mathbf{P}$ and $\mathbf{Q}$ intersect, both curves have the same parameter value τ. Thus, the degree of $\mathbf{L}$ is $m+n-r$ where r is the number of base points. The same thing happens in equation (18.48).

18.3.5 Axial Moving Lines

Moving lines of degree $n > 1$ generally have no axis, that is, no single point about which the line rotates. The special case in which a moving line does have an axis will be referred to as an *axial moving line*. A degree n axial moving line $\mathbf{L}[t]$ can be expressed

$$\mathbf{L}[t] = \mathbf{P}_A \times \mathbf{P}[t] \tag{18.52}$$

where $\mathbf{P}_A$ is the axis and a $\mathbf{P}[t]$ is a degree n curve. An axial moving line can be thought of as following a degree 0 moving point (*i.e.* a fixed point) and a degree n moving point (*i.e.* curve). The relationship between the control lines of the axial moving line and the control points of the curve is

$$\mathbf{L}_i = \mathbf{P}_A \times \mathbf{P}_i. \tag{18.53}$$

Notice that all the control lines $\mathbf{L}_i$ of the axial pencil go through the axis.

18.4 Curve Representation with Two Moving Lines

18.4.1 Axial Moving Line on a Curve

Consider an axial moving line $\mathbf{L}[t]$ which follows a degree n Bézier curve $\mathbf{P}[t]$:

$$\mathbf{L}[t] = \mathbf{P}_A \times \mathbf{P}[t]. \tag{18.54}$$

Though the degree of this moving line is generally n, it can always be reduced to degree $n-1$ if the axis lies on the curve

$$\mathbf{P}_A = \mathbf{P}[\tau], \tag{18.55}$$

because then the moving line has a base point.

We have already confirmed this fact for quadratic Bézier curves in section 18.2.2. For cubic Béziers,

$$\begin{aligned}
\mathbf{L}[t] &= \mathbf{P}[\tau] \times \mathbf{P}[t] \\
&= (t-\tau) \left\{ \begin{array}{ccc} (1-\tau)^2 & (1-\tau)\tau & \tau^2 \end{array} \right\} \\
&\qquad \left[\begin{array}{ccc} 3\mathbf{P}_0 \times \mathbf{P}_1 & 3\mathbf{P}_0 \times \mathbf{P}_2 & \mathbf{P}_0 \times \mathbf{P}_3 \\ 3\mathbf{P}_0 \times \mathbf{P}_2 & \mathbf{P}_0 \times \mathbf{P}_3 + 9\mathbf{P}_1 \times \mathbf{P}_2 & 3\mathbf{P}_1 \times \mathbf{P}_3 \\ \mathbf{P}_0 \times \mathbf{P}_3 & 3\mathbf{P}_1 \times \mathbf{P}_3 & 3\mathbf{P}_2 \times \mathbf{P}_3 \end{array} \right] \left\{ \begin{array}{c} (1-t)^2 \\ (1-t)t \\ t^2 \end{array} \right\}
\end{aligned} \tag{18.56}$$

where $\mathbf{P}_i$ $(i = 0, 1, 2, 3)$ are the control points of the Bézier curve. In general,

$$\begin{aligned}
\mathbf{L}[t] &= (t-\tau) \left\{ \begin{array}{cccc} (1-\tau)^{n-1} & (1-\tau)^{n-2}\tau & \ldots & \tau^{n-1} \end{array} \right\} \\
&\qquad \left[\begin{array}{cccc} \mathbf{L}_{0,0} & \mathbf{L}_{0,1} & \ldots & \mathbf{L}_{0,n-1} \\ \mathbf{L}_{1,0} & \mathbf{L}_{1,1} & \ldots & \mathbf{L}_{1,n-1} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ \mathbf{L}_{n-1,0} & \mathbf{L}_{n-1,1} & \ldots & \mathbf{L}_{n-1,n-1} \end{array} \right] \left\{ \begin{array}{c} (1-t)^{n-1} \\ (1-t)^{n-2}t \\ \vdots \\ t^{n-1} \end{array} \right\}
\end{aligned} \tag{18.57}$$

where

$$\mathbf{L}_{i,j} = \sum_{l+m=i+j+1} \binom{n}{l} \binom{n}{m} \mathbf{P}_l \times \mathbf{P}_m \tag{18.58}$$

Notice that the determinant of the $n \times n$ matrix is equivalent to the Bezout's resultant which gives the implicit equation of the curve.

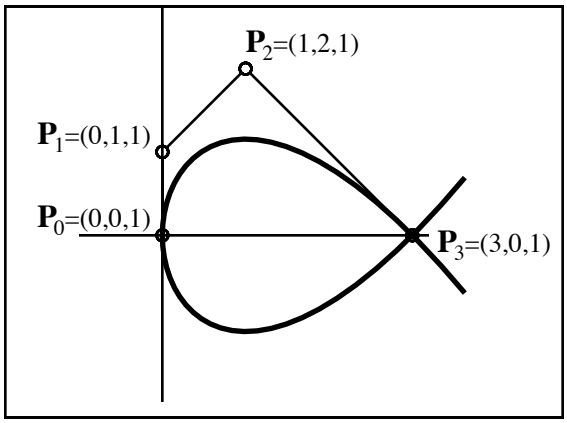


Figure 18.11: Cubic Bézier Curve

18.4.2 Axial Moving Line on a Double Point

Consider what is happening when the axis is on a double point of a degree n curve. Since the curve $\mathbf{P}[t]$ goes through the axis twice at certain parameter values $t = \tau_0, \tau_1$ (which may be imaginary), the moving line has two base points

$$\mathbf{L}[\tau_0] = \mathbf{L}[\tau_1] = 0. \tag{18.59}$$

Therefore, the degree of the axial moving line is reducible to $n-2$. In general, if the axis is on a point of multiplicity m, then the degree of the axial moving line which follows the degree n curve is $n-m$.

Figure 18.11 shows an example of a cubic curve. This cubic Bézier curve has a double point at $\mathbf{P}_3$. Thus the axial moving line with the axis $\mathbf{P}_3$ is a pencil:

$$\begin{aligned}
\mathbf{L}[t] &= \mathbf{P}_3 \times \mathbf{P}[t] \\
&= (3,0,1) \times \left\{(0,0,1)(1-t)^3 + (0,1,1)3(1-t)^2 t + (1,2,1)3(1-t)t^2 + (3,0,1)t^3\right\} \\
&= (t-1)(3t^2+3t, 3t+3, -9t^2-9t) \\
&= (t-1)(t+1)(3t, 3, -9t) \\
&= (t-1)(t+1)\left\{(0,3,0)(1-t) + (3,3,-9)t\right\}.
\end{aligned} \tag{18.60}$$

Two roots of $\mathbf{L}[t] = 0$ shows the type of the double point. In this case, it has two distinct real roots $t = \pm 1$, so the double point is a *crunode*. If the two roots are identical or imaginary, then the double point is a *cusp* or an *acnode*, respectively.

18.4.3 Cubic Curves

For any rational cubic Bézier curve, we can find a pencil and a quadratic moving line which intersects at the moving point. Let us calculate them for the example of Figure 18.11. First of all, make axial moving lines which follow the moving point by using equation (18.56)

$$\begin{aligned}
\mathbf{P}[\tau] \times \mathbf{P}[t] &= (t-\tau)\left\{ \begin{array}{ccc} (1-\tau)^2 & (1-\tau)\tau & \tau^2 \end{array} \right\} \\
&\qquad \left[\begin{array}{ccc} (-3,0,0) & (-6,3,0) & (0,3,0) \\ (-6,3,0) & (-9,12,-9) & (3,9,-9) \\ (0,3,0) & (3,9,-9) & (6,6,-18) \end{array} \right] \left\{ \begin{array}{c} (1-t)^2 \\ (1-t)t \\ t^2 \end{array} \right\}.
\end{aligned} \tag{18.61}$$

Since this moving line follows the moving point $\mathbf{P}[t]$ for any τ, the following equation is obtained

$$\begin{bmatrix} (-3,0,0) & (-6,3,0) & (0,3,0) \\ (-6,3,0) & (-9,12,-9) & (3,9,-9) \\ (0,3,0) & (3,9,-9) & (6,6,-18) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{18.62}$$

In this equation, each row of the matrix is a quadratic moving line that follows the moving point, and any linear combination of these three moving line also follow the moving point. Thus we can make linear operations between the three rows. Since the left most three elements of the matrix shows three lines which intersect at the same point $\mathbf{P}_0$, we can zero out the bottom left corner of the matrix

$$\begin{bmatrix} (-3,0,0) & (-6,3,0) & (0,3,0) \\ (-6,3,0) & (-9,12,-9) & (3,9,-9) \\ (0,0,0) & (0,3,0) & (3,3,-9) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{18.63}$$

Now the bottom moving line has a base point at $t = 0$ and it is reducible to a linear moving line

$$\begin{bmatrix} (0,3,0) & (3,3,-9) \end{bmatrix} \begin{Bmatrix} 1-t \\ t \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}. \tag{18.64}$$

As a result, this pencil and one of the three quadratic moving lines are the required two moving lines

$$\begin{aligned} \mathbf{L}_0[t] &= (0,3,0)(1-t) + (3,3,-9)t \\ \mathbf{L}_1[t] &= (-6,3,0)(1-t)^2 + (-9,12,-9)(1-t)t + (3,9,-9)t^2. \end{aligned} \tag{18.65}$$

Notice that the axis of the pencil always lies at the double point of the cubic curve. It can be calculated from the two control lines. In this example,

$$\begin{aligned} (0,3,0) \times (3,3,-9) &= (-27,0,-9) \\ &= -9(3,0,1), \end{aligned} \tag{18.66}$$

from which the double point is $(3,0)$.

18.4.4 Quartic Curves

Any rational quartic Bézier curve can be expressed either with two quadratic moving lines, or with one linear and one cubic moving line. We can apply the same approach as in section 18.4.3.

Figure 18.12 shows a sample quartic Bézier curve, with the curve plotted beyond the traditional $[0,1]$ parameter interval. First of all, using equation (18.57), calculate four cubic moving lines which follow the curve by

$$\begin{bmatrix} (-4,0,0) & (-12,6,0) & (-8,12,0) & (0,3,0) \\ (-12,6,0) & (-32,36,-24) & (-16,50,-48) & (3,9,-9) \\ (-8,12,0) & (-16,50,-48) & (4,56,-104) & (12,6,-24) \\ (0,2,0) & (4,8,-8) & (12,6,-24) & (8,-4,-16) \end{bmatrix} \begin{Bmatrix} (1-t)^3 \\ (1-t)^2 t \\ (1-t)t^2 \\ t^3 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{18.67}$$

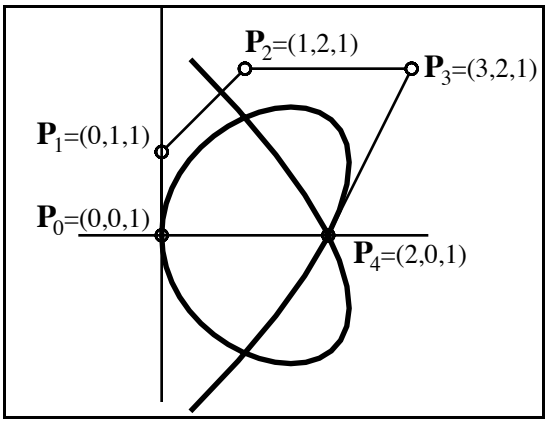


Figure 18.12: Quartic Bézier Curve

In this 4×4 matrix, the left most elements of the bottom two rows can be eliminated by applying row operations

$$\begin{bmatrix} (-4,0,0) & (-12,6,0) & (-8,12,0) & (0,3,0) \\ (-12,6,0) & (-32,36,-24) & (-16,50,-48) & (3,9,-9) \\ (0,0,0) & (0,2,0) & (4,4,-8) & (4,-2,-8) \\ (0,0,0) & (8,6,0) & (28,4,-24) & (20,-14,-40) \end{bmatrix} \begin{Bmatrix} (1-t)^3 \\ (1-t)^2 t \\ (1-t)t^2 \\ t^3 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}, \tag{18.68}$$

from which we can obtain two quadratic moving lines

$$\begin{bmatrix} (0,2,0) & (4,4,-8) & (4,-2,-8) \\ (8,6,0) & (28,4,-24) & (20,-14,-40) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{18.69}$$

that follow the curve, and hence which intersect at the curve.

Any linear combination of these two quadratic moving lines is also a quadratic moving line that follows the curve. If the moving line is axial, then the axis is a double point. Any rational quartic curve has three double points, which may possibly coalesce to form one triple point. In the case of three distinct double points, their locations can be obtained by following method. Let $\mathbf{L}_Q[t]$ be a linear combination of the two moving lines:

$$\begin{aligned} \mathbf{L}_Q[t] &= \begin{Bmatrix} 1-\tau & \tau \end{Bmatrix} \begin{bmatrix} (0,2,0) & (4,4,-8) & (4,-2,-8) \\ (8,6,0) & (28,4,-24) & (20,-14,-40) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \qquad (18.70) \\ &= (8\tau, 4\tau+2, 0)(1-t)^2 + (24\tau+4, 4, -16\tau-8)(1-t)t + (16\tau+4, -12\tau-2, -32\tau-8)t^2 \qquad (18.71) \end{aligned}$$

If it is axial, three control lines must intersect at one point. This condition is expressed with the following equation

$$(8\tau, 4\tau+2, 0) \times (24\tau+4, 4, -16\tau-8) \cdot (16\tau+4, -12\tau-2, -32\tau-8) = 0 \tag{18.72}$$

$$32\tau^3 - 32\tau^2 - 8\tau = 0$$

$$\tau = 0, \frac{1 \pm \sqrt{2}}{2}.$$

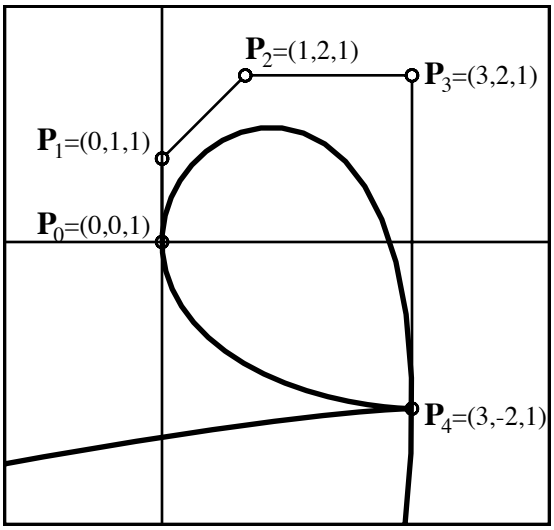


Figure 18.13: Quartic Bézier Curve with a Triple Point

Three double points can be obtained by calculating the axis of $\mathbf{L}_Q[t]$

$$(8\tau, 4\tau+2, 0) \times (24\tau+4, 4, -16\tau-8) = \begin{cases} -1(2,0,1) & (\tau = 0) \\ (-12 \mp 8\sqrt{2})(1, \mp\sqrt{2}, 1) & (\tau = \frac{1\pm\sqrt{2}}{2}), \end{cases} \tag{18.73}$$

from which the double points are $(2,0)$ and $(1, \pm\sqrt{2})$.

If there is a triple point on the curve, however, there is no pair of quadratic moving lines that can represent the curve. Figure 18.13 is an example quartic Bézier curve with a triple point at $\mathbf{P}_4$. We can obtain four cubic moving lines that follow the curve

$$\begin{bmatrix} (-4,0,0) & (-12,6,0) & (-8,12,0) & (2,3,0) \\ (-12,6,0) & (-32,36,-24) & (-14,51,-48) & (12,12,-12) \\ (-8,12,0) & (-14,51,-48) & (12,60,-108) & (24,12,-48) \\ (2,3,0) & (12,12,-12) & (24,12,-48) & (16,0,-48) \end{bmatrix} \begin{Bmatrix} (1-t)^3 \\ (1-t)^2 t \\ (1-t)t^2 \\ t^3 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}, \tag{18.74}$$

form which we can get two quadratic moving lines

$$\begin{bmatrix} (2,3,0) & (8,6,-12) & (8,0,-24) \\ (8,12,0) & (30,21,-48) & (28,0,-84) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{18.75}$$

These two quadratic moving lines (call them $\mathbf{L}_1$ and $\mathbf{L}_2$) are linearly independent, but their intersection does not express the curve because all control points obtained from $\mathbf{L}_1 \times \mathbf{L}_2$ are $(0,0,0)$. This means that both moving lines have exactly the same rotation, but their weights are different. In fact, each of them has a base point and both are identical to the same pencil. The pencil can be obtained by eliminating the bottom left element

$$\begin{bmatrix} (0,0,0) & (2,3,0) & (-4,0,12) \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{18.76}$$

As a result, the curve can be expressed as the intersection of this pencil and one of the cubic moving lines. The axis of the pencil is

$$\begin{aligned}(2,3,0)\times(-4,0,12) &= (36,-24,12)\\ &= 12(3,-2,1)\end{aligned} \tag{18.77}$$

and thus, the triple point is $(3,-2)$.

Notice that a quartic curve with a triple point cannot be expressed as the intersection of two quadratic moving lines. If it were possible, there would be three quadratic moving lines that are linearly independent. In that case, they could represent a cubic curve.

18.4.5 General Case

The methods for cubic and quartic curves can be extended to higher degree curves. For a degree n curve, there exists an n parameter family of moving lines whose degree is $n-1$. A basis for that family of moving lines is given by

$$\begin{bmatrix} \mathbf{L}_{0,0} & \mathbf{L}_{0,1} & \dots & \mathbf{L}_{0,n-1} \\ \mathbf{L}_{1,0} & \mathbf{L}_{1,1} & \dots & \mathbf{L}_{1,n-1} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ \mathbf{L}_{n-1,0} & \mathbf{L}_{n-1,1} & \dots & \mathbf{L}_{n-1,n-1} \end{bmatrix} \begin{Bmatrix} (1-t)^{n-1} \\ (1-t)^{n-2}t \\ \vdots \\ t^{n-1} \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \tag{18.78}$$

where

$$\mathbf{L}_{i,j} = \sum_{l+m=i+j+1} \binom{n}{l} \binom{n}{m} \mathbf{P}_l \times \mathbf{P}_m. \tag{18.79}$$

By calculating linear combinations of these rows of the $n \times n$ matrix, we can always zero out all but two elements in the left-most column, since all such elements are lines which pass through the point $\mathbf{P}_0$:

$$\begin{bmatrix} \mathbf{L}_{0,0} & \mathbf{L}_{0,1} & \dots & \mathbf{L}_{0,n-1} \\ \mathbf{L}_{1,0} & \mathbf{L}_{1,1} & \dots & \mathbf{L}_{1,n-1} \\ 0 & \mathbf{L}_{2,1}^{<1>} & \dots & \mathbf{L}_{2,n-1}^{<1>} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ 0 & \mathbf{L}_{n-1,1}^{<1>} & \dots & \mathbf{L}_{n-1,n-1}^{<1>} \end{bmatrix} \begin{Bmatrix} (1-t)^{n-1} \\ (1-t)^{n-2}t \\ \vdots \\ t^{n-1} \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}. \tag{18.80}$$

$\mathbf{L}_{i,j}^{<k>}$ denotes the element after k-th calculation. Now, we have $n-2$ moving lines whose degree is $n-1$:

$$\begin{bmatrix} \mathbf{L}_{2,1}^{<1>} & \mathbf{L}_{2,2}^{<1>} & \dots & \mathbf{L}_{2,n-1}^{<1>} \\ \mathbf{L}_{3,1}^{<1>} & \mathbf{L}_{3,2}^{<1>} & \dots & \mathbf{L}_{3,n-1}^{<1>} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ \mathbf{L}_{n-1,1}^{<1>} & \mathbf{L}_{n-1,2}^{<1>} & \dots & \mathbf{L}_{n-1,n-1}^{<1>} \end{bmatrix} \begin{Bmatrix} (1-t)^{n-2} \\ (1-t)^{n-3}t \\ \vdots \\ t^{n-2} \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}. \tag{18.81}$$

In this $(n-2)\times(n-1)$ matrix, it turns out that we can *again* zero out all but two elements of the left-most column, since magically, all lines $\mathbf{L}_{i,1}^{<1>}$ contain the point $\mathbf{P}_0$. This can be seen by evaluating the set of equations (18.81) at $t=0$:

$$\begin{Bmatrix} \mathbf{L}_{2,1}^{<1>} \\ \mathbf{L}_{3,1}^{<1>} \\ \vdots \\ \mathbf{L}_{n-1,1}^{<1>} \end{Bmatrix} \cdot \mathbf{P}[0] = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}. \tag{18.82}$$

This zeroing out process can be repeated until one or two rows remain.

If the degree is $n = 2m$, we can repeat the element zeroing process $m - 1$ times and obtain two degree m moving lines

$$\begin{bmatrix} \mathbf{L}_{n-2,m-1}^{<m-1>} & \mathbf{L}_{n-2,m}^{<m-1>} & \dots & \mathbf{L}_{n-2,n-1}^{<m-1>} \\ \mathbf{L}_{n-1,m-1}^{<m-1>} & \mathbf{L}_{n-1,m}^{<m-1>} & \dots & \mathbf{L}_{n-1,n-1}^{<m-1>} \end{bmatrix} \begin{Bmatrix} (1-t)^m \\ (1-t)^{m-1}t \\ \vdots \\ t^m \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}. \tag{18.83}$$

If the degree is $n = 2m + 1$, we can repeat the zero out process m times. In this case, only one element can be eliminated in the last step, and thus, the bottom two rows express degree $m+1$ and degree m moving lines

$$\begin{bmatrix} \mathbf{L}_{n-2,m-1}^{<m-1>} & \mathbf{L}_{n-2,m}^{<m-1>} & \dots & \mathbf{L}_{n-2,n-1}^{<m-1>} \\ 0 & \mathbf{L}_{n-1,m}^{<m>} & \dots & \mathbf{L}_{n-1,n-1}^{<m>} \end{bmatrix} \begin{Bmatrix} (1-t)^{m+1} \\ (1-t)^m t \\ \vdots \\ t^{m+1} \end{Bmatrix} \cdot \mathbf{P}[t] = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}. \tag{18.84}$$

If there is a triple or higher order multiple point, however, the bottom two moving lines may be reducible. In such case, a lower degree moving line can be obtained as discussed in section 18.4.4.

18.4.6 Implicitization

Conventionally, the implicit equation of a rational Bézier curve is calculated as the determinant of the $n \times n$ matrix in equation (18.78) [SAG84], which is called *Bezout's resultant*. It is also possible to implicitize using a pair of moving lines. An arbitrary point $\mathbf{P} = (x, y, 1)$ lies on the curve $\mathbf{P}[t]$ if and only if it lies on both moving lines in equation (18.83) (or (18.84)):

$$\begin{bmatrix} \mathbf{L}_{n-2,m-1}^{<m-1>} \cdot \mathbf{P} & \mathbf{L}_{n-2,m}^{<m-1>} \cdot \mathbf{P} & \dots & \mathbf{L}_{n-2,n-1}^{<m-1>} \cdot \mathbf{P} \\ \mathbf{L}_{n-1,m-1}^{<m-1>} \cdot \mathbf{P} & \mathbf{L}_{n-1,m}^{<m-1>} \cdot \mathbf{P} & \dots & \mathbf{L}_{n-1,n-1}^{<m-1>} \cdot \mathbf{P} \end{bmatrix} \begin{Bmatrix} (1-t)^m \\ (1-t)^{m-1}t \\ \vdots \\ t^m \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}. \tag{18.85}$$

This is equivalent to saying that, if $\mathbf{P} = (x, y, 1)$ lies on the curve, then there exists a value of t which satisfies both equations (18.85), which means that the resultant of the two equations is zero.

Let us calculate the implicit equation of the quartic curve in Figure 18.12. By calculating the dot product with $\mathbf{P}$ for each element, equation (18.69) is expressed as follows

$$\begin{bmatrix} 2y & 4x+4y-8 & 4x-2y-8 \\ 8x+6y & 28x+4y-24 & 20x-14y-40 \end{bmatrix} \begin{Bmatrix} (1-t)^2 \\ (1-t)t \\ t^2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}. \tag{18.86}$$

The implicit equation is the resultant of these two quadratic equations of t,

$$\begin{aligned} & \begin{vmatrix} \begin{vmatrix} 2y & 4x+4y-8 \\ 8x+6y & 28x+4y-24 \end{vmatrix} & \begin{vmatrix} 2y & 4x-2y-8 \\ 8x+6y & 20x-14y-40 \end{vmatrix} \\ \begin{vmatrix} 2y & 4x-2y-8 \\ 8x+6y & 20x-14y-40 \end{vmatrix} & \begin{vmatrix} 4x+4y-8 & 4x-2y-8 \\ 28x+4y-24 & 20x-14y-40 \end{vmatrix} \end{vmatrix} \\ = & \begin{vmatrix} -32x^2-16y^2+64x & -32x^2+32xy-16y^2+64x-32y \\ -32x^2+32xy-16y^2+64x-32y & -32x^2+64xy-48y^2-64y+128 \end{vmatrix} \\ = & \ 512(y^4+4x^3+2xy^2-16x^2-6y^2+16x). \end{aligned} \tag{18.87}$$

This method is generally faster than the conventional method. In most cases, the resultant of the two moving lines ends up in a $(\frac{n}{2})\times(\frac{n}{2})$ determinant, while you must calculate the $n\times n$ determinant in conventional method.

18.5 Tangent Moving Lines

18.5.1 Tangent Moving Lines and Envelope Curves

Consider the moving line which is tangent to a curve $\mathbf{P}[t]$ at each parameter value t. We call this the *tangent moving line* of $\mathbf{P}[t]$. The tangent moving line is computed

$$\text{Line}(\mathbf{P}[t]\times\mathbf{P}'[t]). \tag{18.88}$$

where $\mathbf{P}'[t]=(X'[t],Y'[t],W'[t])$. Thus, one would expect the tangent moving line to generally be degree $2m-1$ if $\mathbf{P}[t]$ is a degree m curve. However, it happens that $\mathbf{P}[t]\times\mathbf{P}'[t]$ always has a base point at $t=\infty$. This is most easily seen by using power basis polynomials

$$\mathbf{P}[t] = (\sum_{i=0}^{m} X_i t^i, \sum_{i=0}^{m} Y_i t^i, \sum_{i=0}^{m} W_i t^i); \tag{18.89}$$

$$\mathbf{P}'[t] = (\sum_{i=0}^{m-1} (i+1)X_{i+1} t^i, \sum_{i=0}^{m-1} (i+1)Y_{i+1} t^i, \sum_{i=0}^{m-1} (i+1)W_{i+1} t^i); \tag{18.90}$$

$$\mathbf{P}[\infty] = (X_m, Y_m, W_m); \tag{18.91}$$

$$\mathbf{P}'[\infty] = m(X_m, Y_m, W_m). \tag{18.92}$$

Thus, when equation (18.88) is expanded, the coefficient of t^{2m-1} is $(X_m,Y_m,W_m)\times m(X_m,Y_m,W_m)=(0,0,0)$. Therefore, the degree of the tangent moving line is at most $2m-2$.

The control lines of the tangent moving line can be obtained as follows. Both $\mathbf{P}[t]$ and $\mathbf{P}'[t]$ can be expressed with degree $m-1$ Bernstein polynomials:

$$\mathbf{P}[t] = \sum_{i=0}^{m} B_i^m[t]\mathbf{P}_i = (1-t)\sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_i + t\sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_{i+1} \tag{18.93}$$

$$\mathbf{P}'[t] = m\sum_{i=0}^{m-1} B_i^{m-1}[t](\mathbf{P}_{i+1}-\mathbf{P}_i) = m\left[-\sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_i + \sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_{i+1}\right]. \tag{18.94}$$

Thus,

$$\mathbf{P}[t]\times\mathbf{P}'[t] = m\left[\sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_i \times \sum_{i=0}^{m-1} B_i^{m-1}[t]\mathbf{P}_{i+1}\right]. \tag{18.95}$$

Any non-axial moving line $\mathbf{L}[t]$ can be represented as the set of lines which are tangent to a given parametric curve — its *envelope curve*. The envelope curve is given by the equation

$$\text{Point}(\mathbf{L}[t]\times\mathbf{L}'[t]) \tag{18.96}$$

where $\mathbf{L}'[t]=(a'[t],b'[t],c'[t])$ [Win23], p. 244. Thus, $\mathbf{L}[\tau]$ contains $\text{Point}(\mathbf{L}[\tau]\times\mathbf{L}'[\tau])$ and is tangent to its envelope curve at that point. Figure 18.14 shows an example of a cubic moving line and its envelope curve. Notice that equations (18.88) and (18.96) are the dual of each other. The degree of $\mathbf{L}[t]\times\mathbf{L}'[t]$ is at most $2m-2$ if $\mathbf{L}[t]$ is a degree m moving line. The control points can be calculated using the dual of equation (18.95).

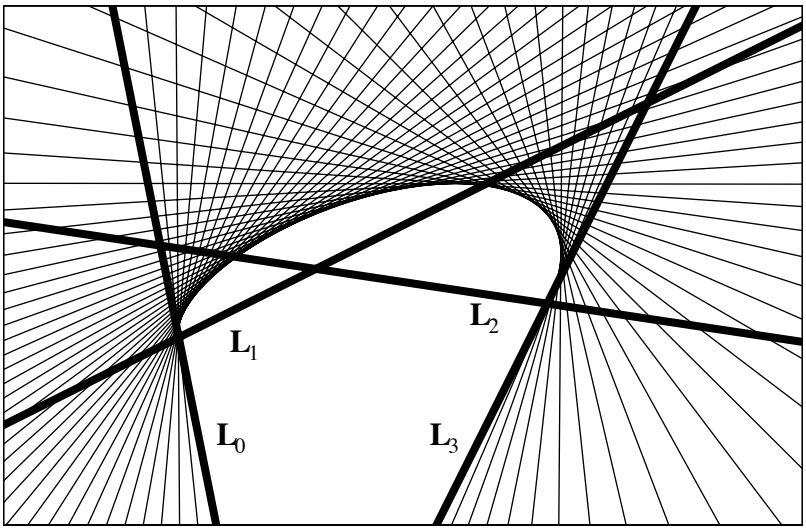


Figure 18.14: Envelope curve

18.5.2 Reciprocal Curves

We would expect that

$$\text{Point}\left((\mathbf{P}[t]\times\mathbf{P}'[t])\times(\mathbf{P}[t]\times\mathbf{P}'[t])'\right)=\text{Point}(\mathbf{P}[t]). \tag{18.97}$$

That is, if we compute the envelope curve of the tangent moving line of $\mathbf{P}[t]$, we would expect to end up with $\mathbf{P}[t]$. However, the expression on the left is degree $4m-6$ and the on the right is degree m. Thus, we conclude that the expression on the left must have $3m-6$ base points. How do we account for those base points?

In equation (18.88), $\text{Line}(\mathbf{P}[t]\times\mathbf{P}'[t])$ has a base point at $t=\tau$ if and only if $\mathbf{P}[\tau]=0$ or $\mathbf{P}'[\tau]=0$. The latter condition means that the curve $\mathbf{P}[t]$ has a *cusp* (or *stationary point* [Sal34], p.25) at $t=\tau$. Likewise, in equation (18.96), $\text{Point}(\mathbf{L}[t]\times\mathbf{L}'[t])$ has a base point at $t=\tau$ if and only if $\mathbf{L}[\tau]=0$ or $\mathbf{L}'[\tau]=0$. Notice that the latter condition is equivalent to saying that the envelope curve $\mathbf{L}[t]\times\mathbf{L}'[t]$ has an *inflection point* (or *stationary tangent* [Sal34], p.33) at $t=\tau$. Now, let m be the degree of the curve $\mathbf{P}[t]$, and let κ and ι be its number of cusps and inflection points respectively. Assume that the curve has no base point. Then, the degree of its tangent moving line, $\text{Line}(\mathbf{P}[t]\times\mathbf{P}'[t])$, is $2m-2-\kappa$, and the degree of the envelope curve of the moving line, $\text{Point}((\mathbf{P}[t]\times\mathbf{P}'[t])\times(\mathbf{P}[t]\times\mathbf{P}'[t])')$ (*i.e.* the original curve $\mathbf{P}[t]$), is $4m-6-2\kappa-\iota$. This fact concludes that any rational curve of degree m must have the following number of cusps and/or inflection points

$$2\kappa+\iota=3m-6. \tag{18.98}$$

Equation (18.98) is confirmed by one of *Plücker's six equations* [Sal34], p.65 for implicit curves

$$\iota=3m^2-6m-6\delta-8\kappa \tag{18.99}$$

where δ is the number of double points except cusps. For any rational curve of degree m, there are fixed number of double points (include cusps)

$$\delta+\kappa=\frac{1}{2}(m-1)(m-2). \tag{18.100}$$

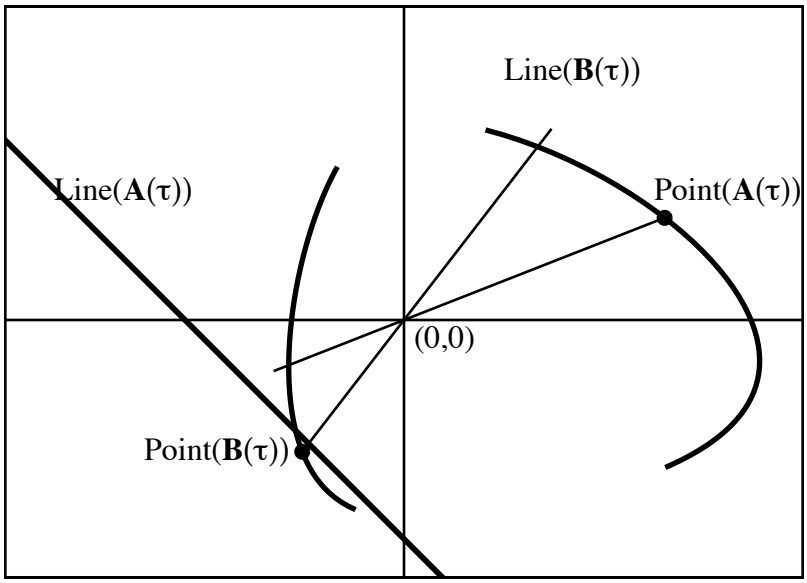


Figure 18.15: Dual and Reciprocal

By eliminating δ from equations (18.99) and (18.100), we can obtain equation (18.98).

Given $\mathbf{A}[t]$, Point($\mathbf{A}[\tau]$) is the dual of Line($\mathbf{A}[\tau]$) for every value of τ, as defined in Figure 18.2. Thus, we say that the moving point Point($\mathbf{A}[t]$) is dual with the moving line Line($\mathbf{A}[t]$). Consider a second moving point Point($\mathbf{B}[t]$) and its dual moving line Line($\mathbf{B}[t]$). If

$$\mathbf{B}[t] = k_1[t](\mathbf{A}[t] \times \mathbf{A}'[t]) \tag{18.101}$$

where $k_1[t]$ is a rational function, then there also exists a rational function $k_2[t]$ such that

$$\mathbf{A}[t] = k_2[t](\mathbf{B}[t] \times \mathbf{B}'[t]). \tag{18.102}$$

When this relationship exists, Point($\mathbf{A}[t]$) and Point($\mathbf{B}[t]$) are referred to as *reciprocal curves* [Sal34], p.54. The rational functions are related

$$\begin{aligned} k_2[t] &= \frac{1}{k_1^2[t]\mathbf{A}[t] \times \mathbf{A}'[t] \cdot \mathbf{A}''[t]} \\ k_1[t] &= \frac{1}{k_2^2[t]\mathbf{B}[t] \times \mathbf{B}'[t] \cdot \mathbf{B}''[t]} \end{aligned} \tag{18.103}$$

The geometric relationship between two reciprocal curves is shown in Figure 18.15. For any value of $t = \tau$, Point($\mathbf{A}[\tau]$) has a dual Line($\mathbf{A}[\tau]$) which is tangent to the curve Point($\mathbf{B}[t]$) at Point($\mathbf{B}[\tau]$). Likewise, Point($\mathbf{B}[\tau]$) has a dual Line($\mathbf{B}[\tau]$) which is tangent to the curve Point($\mathbf{A}[t]$) at Point($\mathbf{A}[\tau]$). If Point($\mathbf{A}[\tau]$) happens to be a cusp, then Point($\mathbf{B}[\tau]$) is an inflection point, and vice versa (see Figure 18.16). If Point($\mathbf{A}[\tau]$) = Point($\mathbf{A}[\beta]$) (that is, if there exists a self intersection or *crunode*), then Point($\mathbf{B}[\tau]$) and Point($\mathbf{B}[\beta]$) have the same tangent line (known as a *double tangent*). This is illustrated in Figure 18.17.

Table 18.1 summarizes the correspondences between a pair of reciprocal curves. The degree of a curve can be defined as the number of times (properly counting real, complex, infinite, and tangent

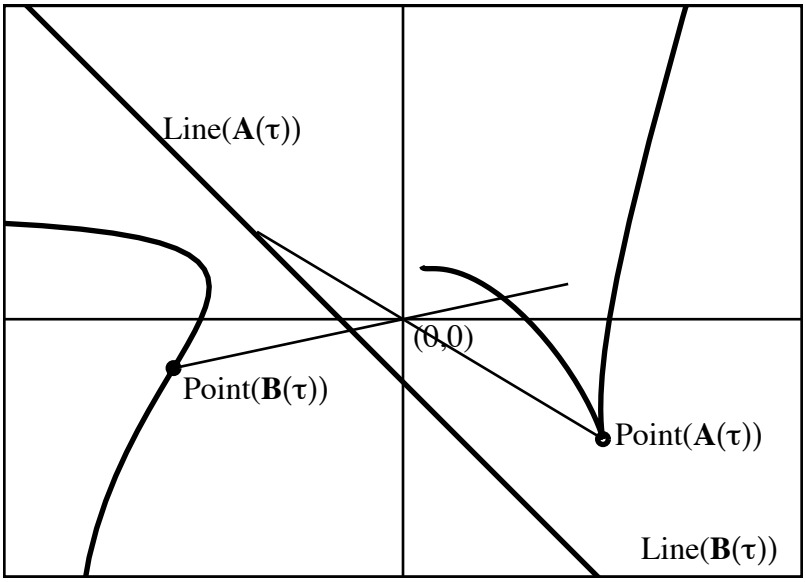


Figure 18.16: Cusp ⇔ Inflection Point

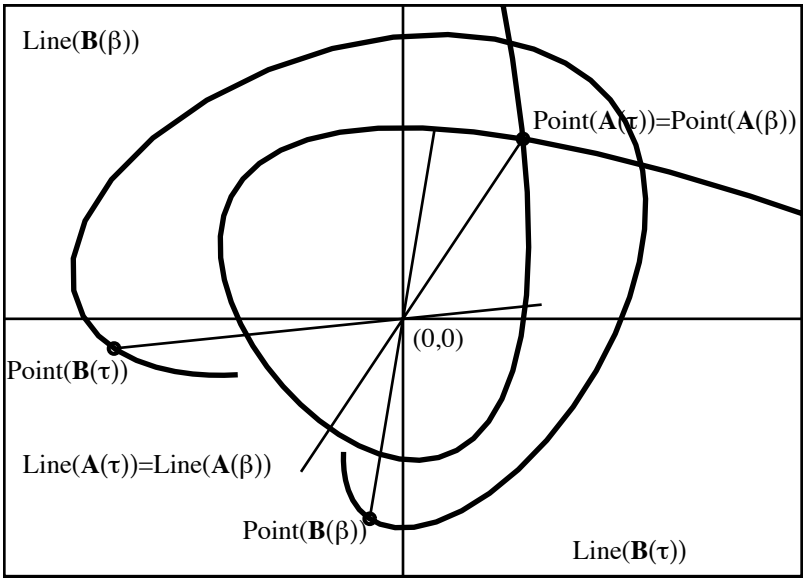


Figure 18.17: Crunode ⇔ Double Tangent

Table 18.1: Reciprocal Curve

curve 1	curve 2
degree	class
class	degree
double point	double tangent
double tangent	double point
cusp	inflection point
inflection point	cusp

intersections) any line intersects the curve. The *class* of a curve is the number of lines passing through a given point which are tangent to the curve.

Tangent moving lines and reciprocal curves provide a powerful tool for analyzing the singularities of rational curves. For example, you can obtain the inflection points of a curve by finding the cusps of its reciprocal curve. The lines tangent to a curve at two distinct real points are found as the crunodes of the reciprocal curve. Equation (18.95) provides an easy way to compute reciprocal curves in Bernstein form.

18.5.3 Tangent Directions

The *hodograph* $\mathbf{P}'[t]$ of a curve $\mathbf{P}[t]$ is its parametric first derivative. The vector with tail at the origin and tip at $\mathbf{P}'[t]$ indicates the magnitude and direction of the derivative of $\mathbf{P}[t]$. It is well-known that the hodograph of a degree n polynomial Bézier curve can be expressed as a degree $n-1$ polynomial Bézier curve. However the hodograph of a degree n rational curve is much more complicated, being a rational function of degree $2n$. [SW87] proposed a *scaled hodograph*, which shows only the derivative direction, and is a degree $2n-2$ polynomial Bźier curve. We can derive an equivalent result using tangent moving lines.

Let $\mathbf{L}_T[t]$ be the tangent moving line of a degree n curve $\mathbf{P}[t]$, and let

$$\mathbf{L}_{Ti} = \text{Line}(a_i, b_i, c_i) \quad (i = 0, 1, \ldots, 2n-2) \tag{18.104}$$

be its control lines. If we force all c_i to be 0

$$\mathbf{L}_H[t] = \sum_{i=0}^{2n-2} \mathbf{L}_{Hi} \tag{18.105}$$

$$\mathbf{L}_{Hi} = \text{Line}(a_i, b_i, 0), \tag{18.106}$$

then this moving line $\mathbf{L}_H[t]$ has an axis at the origin. Notice that $\mathbf{L}_H[t]$ follows the degree $2n-2$ polynomial curve

$$\mathbf{P}_H[t] = \sum_{i=0}^{2n-2} \mathbf{P}_{Hi} \tag{18.107}$$

$$\mathbf{P}_{Hi} = \text{Point}(b_i, -a_i, 1), \tag{18.108}$$

because

$$\text{Point}(0, 0, 1) \times \text{Point}(b_i, -a_i, 1) = \text{Line}(a_i, b_i, 0). \tag{18.109}$$

Since two moving lines $\mathbf{L}_T[t]$ and $\mathbf{L}_H[t]$ has the same direction at any t, $\mathbf{L}_H[t]$ gives the tangent direction of the curve $\mathbf{P}[t]$. Thus, the curve $\mathbf{P}_H[t]$ is a scaled hodograph of $\mathbf{P}[t]$.

This method is available for any rational curve $\mathbf{P}[t]$, however, the magnitude of the derivative is not correct except for polynomial curves.

Chapter 19

Genus and Parametrization of Planar Algebraic Curves

19.1 Genus and Parametrization

We have seen that every parametric curve can be expressed in implicit form. The reverse is not generally true. The condition under which an implicit curve can be parametrized using rational polynomials is that its *genus* must be zero. Basically, the genus of a curve is given by the formula $g = \frac{(n-1)(n-2)}{2} - d$ where g is the genus, n is the degree, and d is the number of double points. There are some subtleties involved in this equation, but we will not concern ourselves with them. They deal with more complicated multiple points.

We see immediately that all curves of degree one and two have genus zero and thus can be parametrized using rational polynomials. Curves of degree three must have one double point in order to qualify.

It is also the case that an irreducible curve of degree n can have at most $(n-1)(n-2)/2$ double points. Thus, a rational degree n curve has the largest possible number of double points for a curve of its degree. An irreducible curve is one whose implicit equation $f(x, y) = 0$ cannot be factored. For example, the degree two curve $xy + x + y + 1$ can be factored into $(x + 1)(y + 1)$ and is thus actually two straight lines. Note that the point at which those two lines intersect is a double point of the curve, even though an irreducible conic cannot have any double points.

Another example of a reducible curve is given by the quartic which is factored into

$$(x^2 + y^2 - 1)(x^2 + y^2 - 4)$$

which is two concentric circles. This leads to another characteristic of rational curves - you can sketch an entire rational curve without removing your pencil from the paper (with the possible exception of a finite number of acnodes). For this reason, the classical algebraic geometry literature sometimes refers to rational curves as "unicursal" curves.

It should also be noted that an algebraic curve which consists of more than one component is not necessarily reducible. For example, many non-rational cubic curves consist of an oval and a branch which does not touch the oval, and yet the equation does not factor.

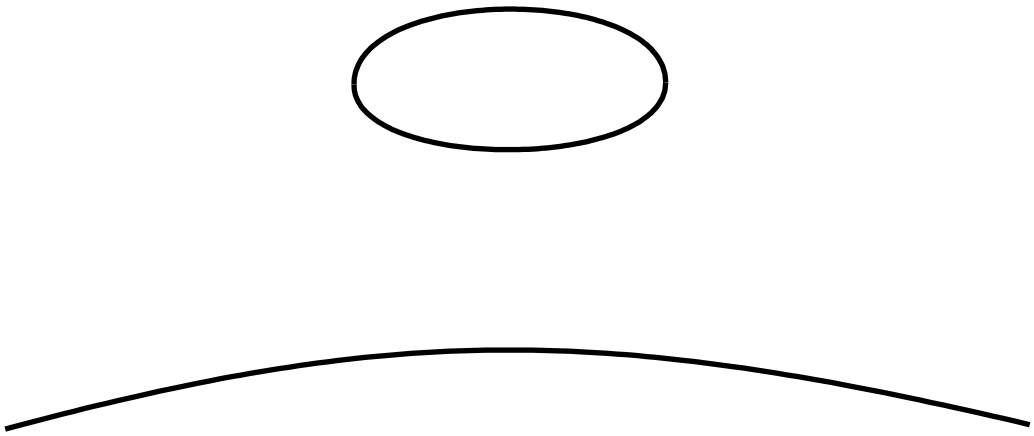


Figure 19.1: Irreducible Cubic Curve

19.2 Detecting Double Points

How does one verify the existence of a double point? A computational method is to verify that any straight line through the alleged double point hits the curve at least twice at that point. For example, any curve whose equation has no constant or linear terms has a double point at the origin. Consider the cubic algebraic curve:

$$x^3 - 2xy^2 + xy + 3y^2 = 0.$$

To determine how many times a general line through the origin hits this curve, we define such a line parametrically so that the point $t = 0$ on the line corresponds to the origin:

$$x = at, \qquad y = bt.$$

The intersection of the line and the curve yields

$$(a^3 - 2ab^2)t^3 + (ab + 3b^2)t^2 = 0.$$

Since this equation has at least a double root at $t = 0$ for any values of a and b (which control the slope of the line), we conclude that the curve has a double point at the origin.

There are basically three types of double points: the crunode (or simply *node*, or self intersection), the cusp, and the acnode (or isolated node). An example of a cubic curve with a crunode at the origin is the curve in Figure 19.2 whose implicit equation is

$$x^3 + 9x^2 - 12y^2 = 0.$$

You can verify by implicitization that this curve can be expressed as a Bézier curve with control points $(3, 3)$, $(-13, -15)$, $(-13, 15)$, $(3, -3)$.

An example of a cubic curve with a cusp at the origin is given by

$$x^3 - 3y^2 = 0$$

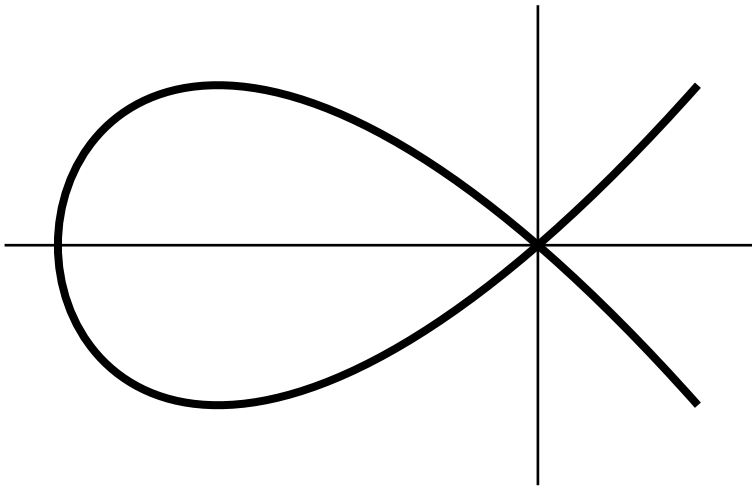


Figure 19.2: Crunode: $x^3 + 9x^2 - 12y^2 = 0$

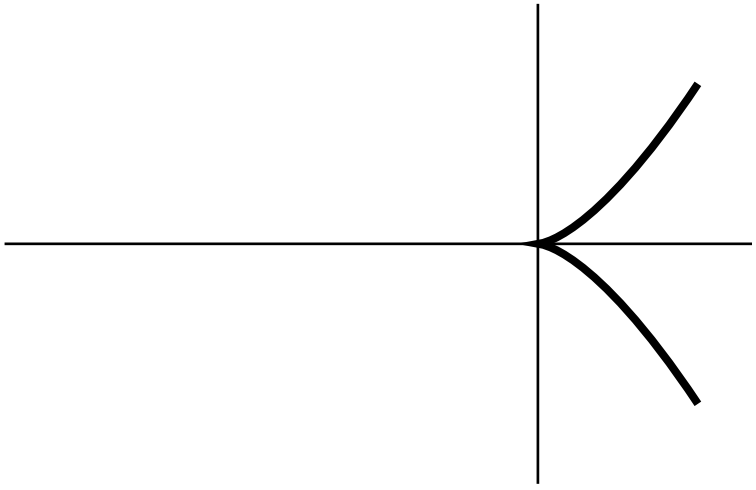


Figure 19.3: Cusp: $x^3 - 3y^2 = 0$

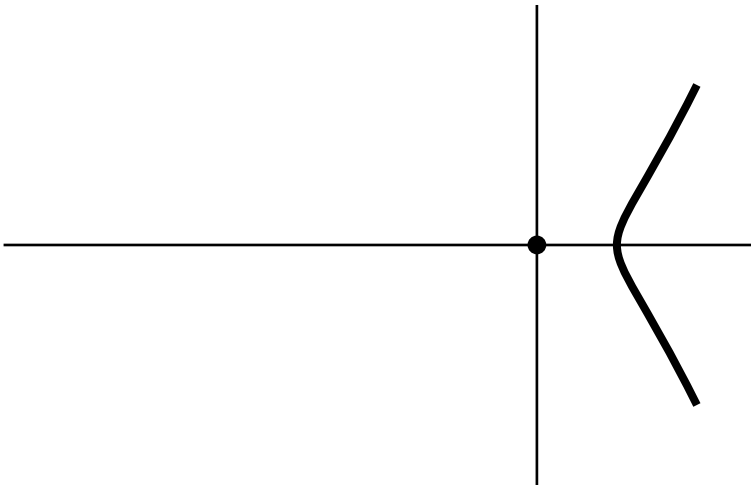


Figure 19.4: Acnode: $x^3 - 3x^2 - 3y^2 = 0$

which is illustrated in Figure 19.2.

An example of a curve with an acnode at the origin is shown in Figure 19.2, given by the equation

$$x^3 - 3x^2 - 3y^2 = 0.$$

We can classify a double point as being a crunode, acnode or cusp by examining the tangent lines to the curve at the double point. A crunode has two distinct real tangent lines, a cusp has two identical tangent lines, and an acnode has two complex tangent lines. A tangent line through a double point hits the curve three times at the double point. We can find the slope of the tangent lines by computing values of a and b for which the intersection of the line and the curve has a triple root at $t = 0$. In the previous example, this occurs if $ab + 3b^2 = 0$ from which $a = 1$, $b = 0$ and $a = -3$, $b = 1$ are the two independent solutions. These represent two real, distinct lines, so the double point in a crunode.

The exercise that we just performed only lets us verify that a given point is singular (a singular point is a double point, triple point, or multiple point in general). To determine the location of singular points, we can use a tool known as the *discriminant* (discussed later).

19.3 Implicit Curve Intersections

We have seen how the intersections of two Bezier curves can be computed, and also how the intersection points of a parametric and an implicit curve can be found. What about two implicit curves?

One direct method for computing the intersection points of two implicit curves is to take the resultant of the curves with respect to x or y. The X-resultant is computed by treating the implicit equations as polynomials in x whose coefficients are polynomials in y. The X-resultant eliminates x from the two equations and produces a polynomial in y whose roots are the y coordinates of the intersection points.

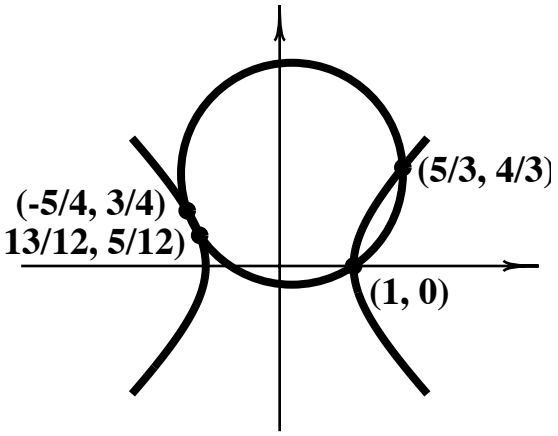


Figure 19.5: Circle and Hyperbola

We illustrate with a circle $6x^2 + 6y^2 - 2x - 15y - 4 = 0$ and a hyperbola $x^2 - y^2 - 1 = 0$.

The x-resultant of these two implicit equations is $144y^4 - 360y^3 + 269y^2 - 60y$ whose roots are $y = 0$, $y = 4/3$, $y = 3/4$, and $y = 5/12$. These are the y-coordinates of the points of intersection of the two curves.

We can use the y-resultant to find the x-coordinates of the points of intersection. The y-resultant is $144x^4 - 48x^3 - 461x^2 + 40x + 325$ which has roots $x = 1$, $x = 5/3$, $x = -5/4$, and $x = -13/12$.

We now find ourselves in the interesting situation of knowing the x and y components of the points of intersection, but we don't know which x goes with which y! One way to determine that is simply to evaluate each curve equation with every x and every y to see which (x, y) pairs satisfy both curve equations simultaneously. A more clever way is to use Euclid's algorithm. In fact, Euclid's algorithm spares us the trouble of computing both the x-resultant and the y-resultant.

Suppose we had only computed the y-resultant and we wanted to find the y-coordinate of the point of intersection whose x-coordinate is 5/3. That is to say, we want to find a point $(\frac{5}{3}, y)$ which satisfies both curve equations. We substitute $x = 5/3$ into the circle equation to get $16/6 - y^2 = 0$ and into the hyperbola equation to get $6y^2 - 15y + 28/3 = 0$. We now simply want to find a value of y which satisfies both of these equations. Euclid's algorithm tells us that the GCD of these two $3y - 4 = 0$, and thus one point of intersection is $(\frac{5}{3}, \frac{4}{3})$.

19.4 Discriminants

The discriminant of a univariate polynomial is the resultant of the polynomial and its first derivative. If the discriminant is zero, the polynomial has a double root. (Why?) We all encountered discriminants as early as 7th grade in connection with the quadratic equation. For a degree two polynomial $at^2 + bt + c$, the value $b^2 - 4ac$ is referred to as the discriminant, although if we actually compute the resultant of $at^2 + bt + c$ and its derivative $2at + b$, we find that the discriminant is actually $a^2(b^2 - 4ac)$. However, since it is understood that $a \neq 0$, the discriminant can only vanish if $b^2 - 4ac = 0$.

It is also possible to compute the discriminant of an implicit curve by taking the resultant of

the implicit equation and its partial derivative with respect to one of the variables. To do this, we treat the implicit equation as a polynomial in y whose coefficients are polynomials in x (or, vice versa). The resultant will then be a polynomial in x with constant coefficients. The roots of that polynomial will correspond to the x coordinates of the vertical tangents and of any double points.

Graphically, the discriminant can be thought of as the silhouette of the curve.

19.5 Parametrizing Unicursal Curves

There are several ways in which a parametrization may be imposed on a curve of genus zero. For a conic, we may establish a one-one correspondence between points on the curve and a family of lines through a point on the curve. This is most easily illustrated by translating the curve so that it passes through the origin, such as does the curve

$$x^2 - 2x + y^2 = 0$$

which is a unit circle centered at $(0, 1)$. We next make the substitution $y = tx$ and solve for x as a function of t:

$$x^2(1 + t^2) - 2x = 0$$

$$x = \frac{2}{1 + t^2}$$

$$y = tx = \frac{2t}{1 + t^2}$$

Notice that $y = tx$ is a family of lines through the origin. A one parameter family of lines (or, of any implicitly defined curves) is known as a *pencil* of lines or curves. The variable line $y = tx$ intersects the curve once at the origin, and at exactly one other point (because of Bezout's theorem). Thus, we have established a one-one correspondence between points on the curve and values t which correspond to lines containing that point and the origin.

An example of a circle parametrized in this manner is shown in Figure 19.6.

The same trick can be played with a genus zero cubic curve which has been translated so that its double point lies on the origin. if this happens, its implicit equation involves terms of degree two and three only. For example, consider the curve

$$(x^3 + 2x^2y + 3xy^2 + 4y^3) + (5x^2 + 6xy + 7y^2) = 0$$

which has a double point at the origin. Again make the substitution $y = tx$ to obtain

$$x^3(1 + 2t + 3t^2 + 4t^3) + x^2(5 + 6t + 7t^2) = 0.$$

From which

$$x = -\frac{5 + 6t + 7t^2}{1 + 2t + 3t^2 + 4t^3}, \quad y = tx = -\frac{5t + 6t^2 + 7t^3}{1 + 2t + 3t^2 + 4t^3}$$

An example of a cubic curve parametrized in this manner is shown in Figure 19.7.

In general, a cubic curve does not have a double point, and if it does, that double point is not generally at the origin. Thus, before one can parametrize a cubic curve, one must first compute the location of its double point (or, determine that rational parametrization is not possible if there is no double point).

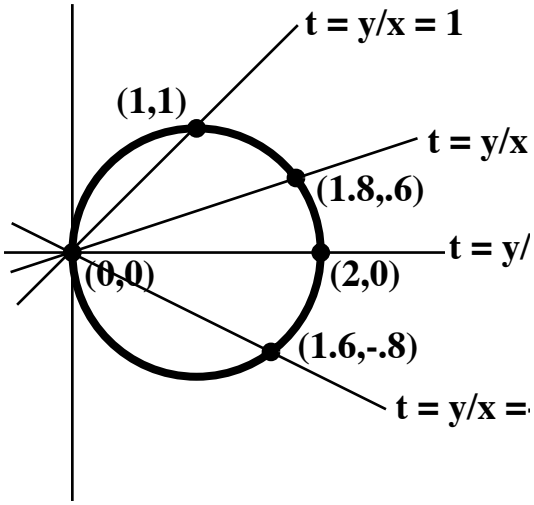


Figure 19.6: Parametrizing a Circle

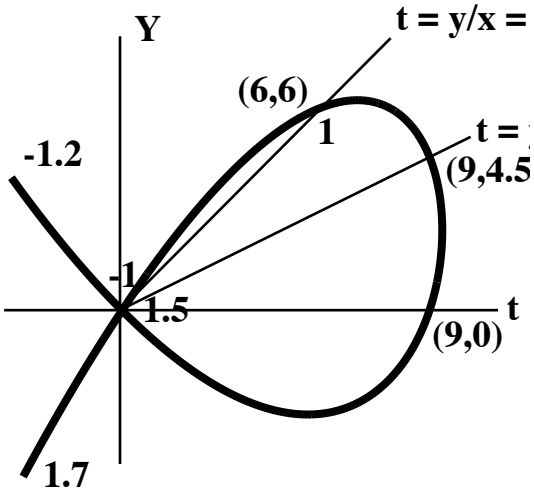


Figure 19.7: Parametrizing a Cubic Curve

Double points satisfy the equations $f(x,y) = f_x(x,y) = f_y(x,y) = 0$. Consider the cubic curve

$$f(x,y) = -21 + 46\,x - 13\,x^2 + x^3 + 25\,y - 23\,x\,y + 3\,x^2\,y - 9\,y^2 + 3\,x\,y^2 + y^3$$

for which

$$f_x(x,y) = 46 - 26\,x + 3\,x^2 - 23\,y + 6\,x\,y + 3\,y^2$$

and

$$f_y(x,y) = 25 - 23\,x + 3\,x^2 - 18\,y + 6\,x\,y + 3\,y^2$$

From section 19.3, we compute the x coordinates of the intersections of $f_x = 0$ and $f_y = 0$ by taking the resultant of f_x and f_y with respect to y:

$$\text{Resultant}(\mathrm{f_x}, \mathrm{f_y}, \mathrm{y}) = 174 - 159\,\mathrm{x} + 36\,\mathrm{x}^2$$

whose roots are $x = 2$ and $x = \frac{29}{12}$. Likewise the y coordinates of the intersections of $f_x = 0$ and $f_y = 0$ are found by taking the resultant of f_x and f_y with respect to x:

$$\text{Resultant}(\mathrm{f_x}, \mathrm{f_y}, \mathrm{x}) = 297 - 207\,\mathrm{y} + 36\,\mathrm{y}^2$$

whose roots are $y = 3$ and $y = \frac{11}{4}$. From these clues, we find that the only values of (x,y) which satisfy $f(x,y) = f_x(x,y) = f_y(x,y) = 0$ are $(x,y) = (2,3)$, which is therefore the double point.

This curve can be parametrized by translating the implicit curve so that the double point lies at the origin. This is done by making the substitution $x = x + 2$, $y = y + 3$, yielding

$$2\,x^2 + x^3 + 7\,x\,y + 3\,x^2\,y + 6\,y^2 + 3\,x\,y^2 + y^3$$

Parametrization is then performed using the method discussed earlier in this section,

$$x = -\frac{6t^2 + 7t + 2}{t^3 + 3t^2 + 3t + 1}; \quad y = -\frac{6t^3 + 7t^2 + 2t}{t^3 + 3t^2 + 3t + 1}$$

and the parametrized curve is translated back so that the doubled point is again at $(2,3)$:

$$x = -\frac{6t^2 + 7t + 2}{t^3 + 3t^2 + 3t + 1} + 2 = \frac{2t^3 - t}{t^3 + 3t^2 + 3t + 1};$$

$$y = -\frac{6t^3 + 7t^2 + 2t}{t^3 + 3t^2 + 3t + 1} + 3 = \frac{-3t^3 + 2t^2 + 7t + 3}{t^3 + 3t^2 + 3t + 1}$$

19.6 Undetermined Coefficients

An implicit curve equation has $(n+1)(n+2)/2$ terms. Since the equation $f(x,y) = 0$ can be scaled without changing the curve, we can freely assign any of the non-zero coefficients to be 1 (or, any other value that we choose) and we are left with $(n^2 + 3n)/2$ coefficients. This means that we can generally specify $(n^2 + 3n)/2$ points through which an implicit curve will pass. One way this is done is by using the method of undetermined coefficients.

Consider the general conic given by

$$c_{20}x^2 + c_{11}xy + c_{02}y^2 + c_{10}x + c_{01}y + c_{00} = 0$$

and five points (x_i, y_i) which we wish the curve to interpolate. The coefficients c_{ij} can be solved from the system of linear equations:

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{bmatrix} \begin{Bmatrix} {}^{20} \\ c_{11} \\ c_{02} \\ c_{10} \\ c_{01} \\ c_{00} \end{Bmatrix} = 0.$$

Of course, this method can be applied to any special equation. For example, the general equation of a circle is $(x-a)^2 + (y-b)^2 - r^2 = 0$. A circle through three points can be found by applying the method of undetermined coefficients to the equation $Ax + By + C = x^2 + y^2$ to compute A, B, C. Then, we find $a = A/2$, $b = B/2$, and $r = \sqrt{C + a^2 + b^2}$.

Chapter 20

POLYNOMIAL APPROXIMATION OF RATIONAL CURVES

A degree r plane rational Bézier curve

$$\begin{aligned}\mathbf{R}(t) &= \frac{\sum_{i=0}^{r} B_i^r(t) w_i \mathbf{R}_i}{\sum_{i=0}^{r} B_i^r(t) w_i} \\ &= (x_R(t), y_R(t)) \qquad (\mathbf{R}_i = (X_i, Y_i))\end{aligned} \tag{20.1}$$

can be expressed as a degree n plane polynomial Bézier curve $\mathbf{H}(t)$ with a *moving control point* $\mathbf{M}(t)$:

$$\begin{aligned}\mathbf{H}(t) &\equiv \mathbf{R}(t) \\ &= \sum_{i=0, i\neq m}^{n} B_i^n(t)\mathbf{P}_i + \mathbf{M}(t) B_m^n(t) \\ &= (x_H(t), y_H(t)) \quad (0 < m < n),\end{aligned} \tag{20.2}$$

where

$$\mathbf{M}(t) = \frac{\sum_{i=0}^{r} B_i^r(t) w_i \mathbf{M}_i}{\sum_{i=0}^{r} B_i^r(t) w_i} \tag{20.3}$$

is a degree r plane rational Bézier curve. The curve $\mathbf{H}(t)$ is called a *Hybrid Curve* [SK91]. Figure 20.1.a shows a rational cubic curve, and Figure 20.1.b shows that same curve represented as a degree two hybrid curve. Figure 20.1.c shows how to evaluate a point $\mathbf{H}(t)$ on a hybrid curve — you first evaluate the moving control point at $\mathbf{M}(t)$. The polynomial Bézier curve can then be evaluated, treating point $\mathbf{M}(t)$ as a stationary control point ($\mathbf{P}_1$ in this case). Figure 20.1.d shows the curve in Figure 20.1.a represented as a degree four hybrid curve.

As derived in [SK91], the $\mathbf{P}_i$ and $\mathbf{M}_i$ can be computed:

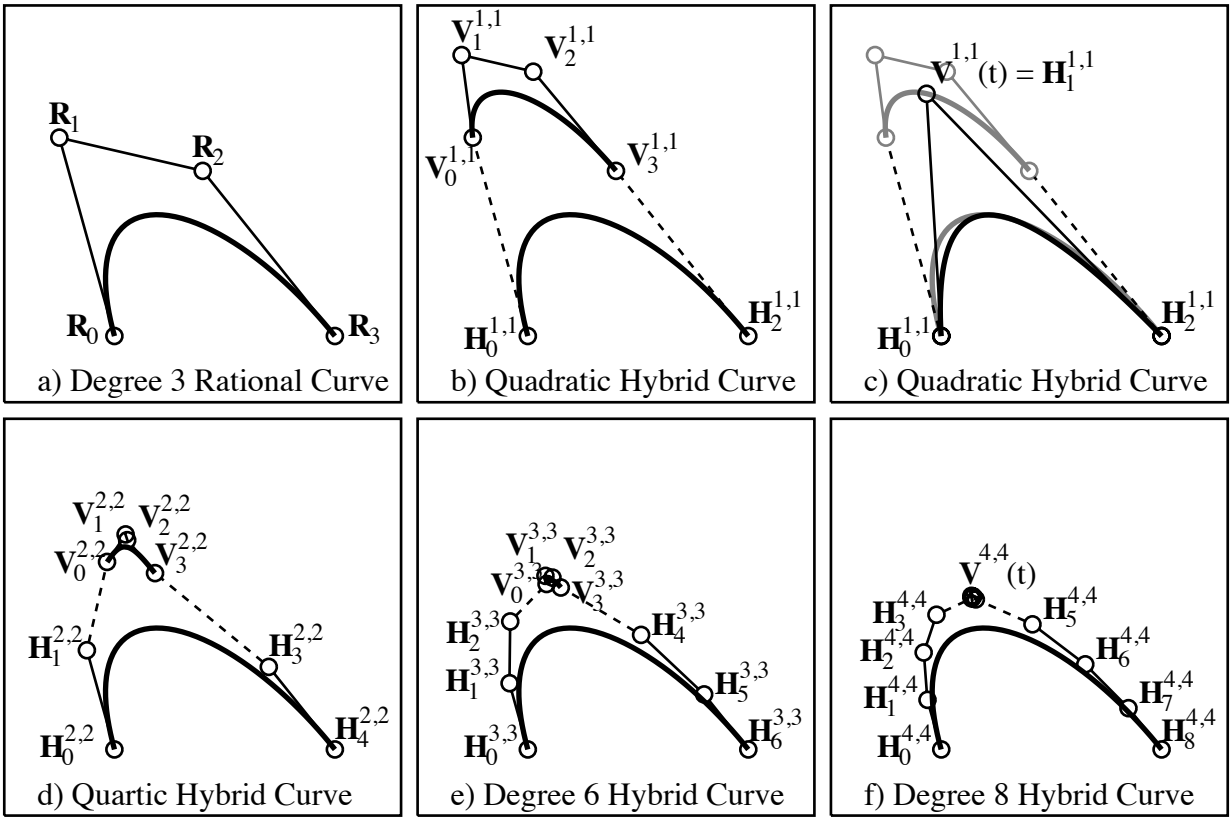


Figure 20.1: Hybrid curves

For $i = 0, \ldots, m-1$:

$$\begin{aligned} \mathbf{P}_0 &= \mathbf{R}_0, \\ \mathbf{P}_i &= \mathbf{R}_0 + \frac{\sum_{j=1}^{\min(r,i)} \binom{r}{j}\binom{n}{i-j} w_j[\mathbf{R}_j - \mathbf{P}_{i-j}]}{w_0\binom{n}{i}}. \end{aligned} \tag{20.4}$$

For $i = n+r, \ldots, m+1+r$:

$$\begin{aligned} \mathbf{P}_n &= \mathbf{R}_r, \\ \mathbf{P}_{i-r} &= \mathbf{R}_r + \frac{\sum_{j=\max(0,i-n)}^{r-1} \binom{r}{j}\binom{n}{i-j} w_j[\mathbf{R}_j - \mathbf{P}_{i-j}]}{w_r\binom{n}{i-r}}. \end{aligned} \tag{20.5}$$

For $i = 0, \ldots, r$:

$$\mathbf{M}_i = \sum_{\substack{j = \max(0, i+m-n) \\ j \neq i}}^{\min(i+m,r)} \frac{\binom{r}{j}\binom{n}{i+m-j} w_j[\mathbf{R}_j - \mathbf{P}_{i+m-j}]}{w_i\binom{n}{m}\binom{r}{i}} + \mathbf{R}_i. \tag{20.6}$$

The moving control point is always a rational curve of the same degree as the original rational curve. Also, the control point weights of the moving control point curve is the same as those of the original rational curve.

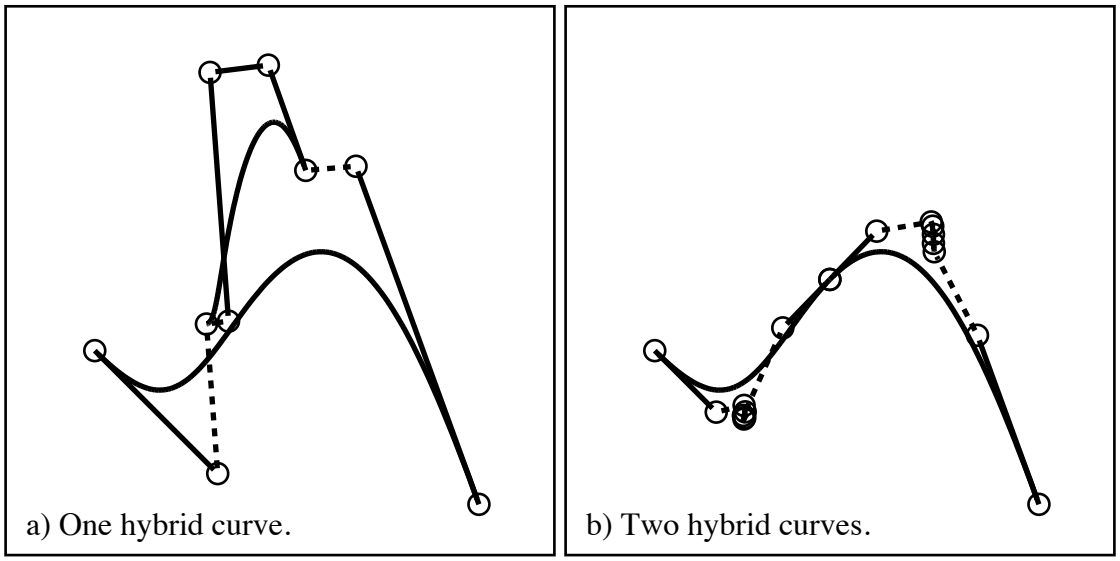


Figure 20.2: Hybrid curves after subdivision

If we simply replace the moving control point $\mathbf{M}(t)$ with a stationary control point $\mathbf{P}_m$, the hybrid representation $\mathbf{H}(t)$ of the rational curve $\mathbf{R}(t)$ leads directly to a polynomial approximation $\mathbf{P}(t)$ (as in equation 20.10). An intelligent choice of point $\mathbf{P}_m$ is the midpoint of the x/y min-max

box which bounds the moving control point. Denote the width of such a bounding box as $2|\triangle_x|$, the heighth $2|\triangle_y|$, and the diagonal $2||\triangle||$. If n is an even number, $m = n/2$, and $||\mathbf{M}(t) - \mathbf{P}_m|| \leq ||\triangle||$, then

$$||\mathbf{P}(t) - \mathbf{H}(t)|| \leq ||\triangle|| \left(\begin{array}{c} n \\ \frac{n}{2} \end{array} \right) \frac{1}{2^n}, \tag{20.7}$$

$$\begin{aligned} |x_P(t) - x_H(t)| &\leq B_{xPH} \\ &= |\triangle_x| \left(\begin{array}{c} n \\ \frac{n}{2} \end{array} \right) \frac{1}{2^n}, \end{aligned} \tag{20.8}$$

$$\begin{aligned} |y_P(t) - y_H(t)| &\leq B_{yPH} \\ &= |\triangle_y| \left(\begin{array}{c} n \\ \frac{n}{2} \end{array} \right) \frac{1}{2^n}. \end{aligned} \tag{20.9}$$

Note that $||\triangle||$ for the quartic hybrid curve is smaller than $||\triangle||$ for the quadratic hybrid curve. This is often, though not always, the case, depending on the radius of convergence. Conditions under which convergence does occur is discussed in [WS93]. However, as illustrated in Figure 20.2, each half of a subdivided rational curve always has a smaller value of $||\triangle||$ than does the original rational curve.

20.1 PLANAR AREAS

Shape design often calls for the computation of integral values such as the cross-sectional area of ducts or the center of gravity of a ship's cross-section. For plane regions bounded by polynomial Bézier curves, closed form solutions exist. This chapter reviews those solutions and also presents a numerical algorithm for the case of where the bounding curves are in *rational* Bézier form. The earliest such algorithm for polynomial curves is probably due to Faux and Pratt [FP79]. General algorithms have been developed by Liu, Nowacki, and Lu [Liu87, NLL90].

Most algorithms for polynomial curves, such as evaluation, subdivision and degree elevation, extend readily to rational curves, while other algorithms, such as those involving derivatives and integrals, do not. Ultimately, the problem of computing areas, centroids, or volumes of revolution for regions bounded by rational Bézier curves reduces to the problem of integrating a rational function (that is, a polynomial divided by another polynomial) for which there exist basically two approaches: the method of partial fractions, and numerical integration.

Recall that partial fractions is a basic technique from calculus in which a rational function of t is written as the sum of a polynomial and fractions of the form

$$\frac{A}{(at+b)^k} \quad \text{or} \quad \frac{At+B}{(at^2+bt+c)^k},$$

where A, B, a, b and c are constants and k is a positive integer. This method leads to straightforward closed-form integration formulae for degree two rational functions, and significantly more complicated formulae for degree three and four cases. Closed-form integration formulae do not generally exist for degree greater than four.

Numerical integration [BF89], using methods such as the trapezoid rule, Simpson's rule, etc., is the technique most often applied in practice. But for rational functions, determining an error bound is expensive, since it involves finding a bound on high order derivatives of a rational function.

20.2 Integrals Involving Plane Bézier Curves

This section reviews general algorithms for computing integral values on plane Bézier curves, as given by D. Liu and H. Nowacki [Liu87, NLL90].

Let $\mathbf{P}_i = (x_i, y_i) \in R^2, i = 0, 1, \ldots, n$, then

$$\begin{aligned} \mathbf{P}(t) &= \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i \\ &= (x_P(t), y_P(t)) \quad (\mathbf{P}_i = (x_i, y_i)) \end{aligned} \tag{20.10}$$

is a degree n plane Bézier curve.

First, define the following:

$$\begin{aligned} C_{i,j}^{(n)} &= \int_0^1 B_i^n(t) B_j^{n-1}(t) dt, \\ D_{i,j}^{(n)} &= C_{i,j-1}^{(n)} - C_{i,j}^{(n)} \\ &= \begin{cases} -\frac{1}{2n} & (i,j) = (0,0) \\ \frac{1}{2n} & (i,j) = (n,n) \\ \dfrac{(j-i)\binom{n}{i}\binom{n}{j}}{(i+j)(2n-i-j)\binom{2n}{i+j}} & 0 \le i,j \le n, (i,j) \ne (0,0), (n,n). \end{cases} \end{aligned} \tag{20.11}$$

$$\begin{aligned} C_{i,j,k}^{(n)} &= \int_0^1 B_i^n(t) B_j^n(t) B_k^{n-1}(t) dt, \\ D_{i,j,k}^{(n)} &= C_{i,j,k-1}^{(n)} - C_{i,j,k}^{(n)} \\ &= \begin{cases} -\frac{1}{3n} & (i,j,k) = (0,0,0) \\ \frac{1}{3n} & (i,j,k) = (n,n,n) \\ \dfrac{(2k-i-j)\binom{n}{i}\binom{n}{j}\binom{n}{k}}{(i+j+k)(3n-i-j-k)\binom{3n}{i+j+k}} & 0 \le i,j,k \le n, (i,j,k) \ne (0,0,0), (n,n,n). \end{cases} \end{aligned} \tag{20.12}$$

The **area under a curve** given by equation 20.10 over the interval $[x_0, x_n]$ (see Figure 20.3) is:

$$\begin{aligned} Area &= \int_{x_0}^{x_n} y_P dx_P \\ &= n \sum_{i,j=0}^{n} D_{i,j}^{(n)} y_i x_j, \qquad (\frac{dx_P(t)}{dt} \ne 0, \quad 0 \le t \le 1). \end{aligned} \tag{20.13}$$

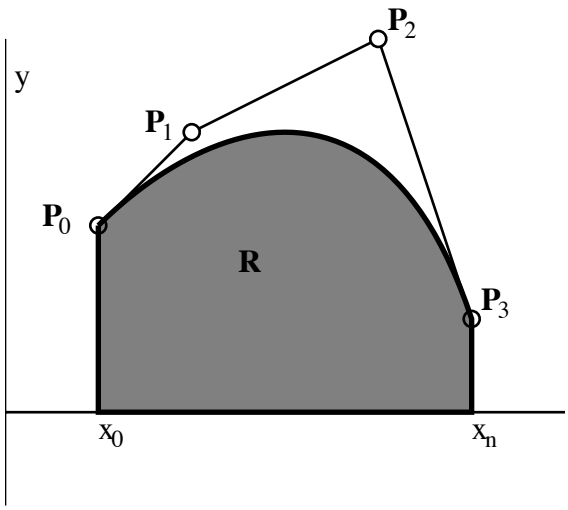


Figure 20.3: Area under cubic Bézier curve

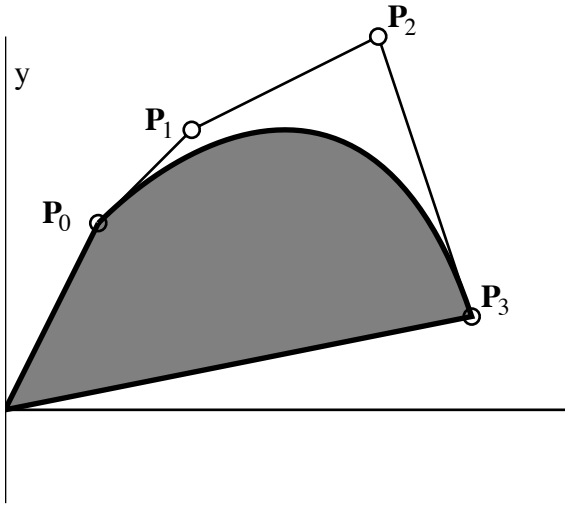


Figure 20.4: Area of cubic Bézier sector

The **area contained in the sector** formed by the origin and a curve given by equation 20.10 (see Figure 20.4) is:

$$\begin{aligned} \textit{Directed Area} &= \frac{1}{2}\int_0^1 \mathbf{P}(t) \wedge \frac{d\mathbf{P}(t)}{dt} dt \\ &= n \sum_{\substack{i,j=0 \\ i<j}}^{n} D_{i,j}^{(n)} (\mathbf{P}_i \wedge \mathbf{P}_j), \qquad (\mathbf{P}(t) \wedge \frac{d\mathbf{P}(t)}{dt} \neq 0, \quad 0 \leq t \leq 1), \end{aligned} \tag{20.14}$$

where symbol $\wedge$ denotes a wedge product, and

$$\mathbf{P}_i \wedge \mathbf{P}_j = \begin{vmatrix} x_i & y_i \\ x_j & y_j \end{vmatrix}. \tag{20.15}$$

The **first moments of area** of R (in Figure 20.3) about the x- and y-axes are:

$$\begin{aligned} I_x &= \int_{x_0}^{x_n} \frac{1}{2} y_P^2 dx_P \\ &= \frac{n}{2} \sum_{i,j,k=0}^{n} D_{i,j,k}^{(n)} y_i y_j x_k. \end{aligned} \tag{20.16}$$

$$\begin{aligned} I_y &= \int_{x_0}^{x_n} x_P y_P dx_P \\ &= n \sum_{i,j,k=0}^{n} D_{i,j,k}^{(n)} x_i y_j x_k. \end{aligned} \tag{20.17}$$

The **volume** of a solid of revolution generated by revolution of region R (in Figure 20.3) about the x-axis is:

$$\begin{aligned} \textit{Volume} &= \int_{x_0}^{x_n} \pi y_P^2 dx_P \\ &= 2\pi I_x. \end{aligned} \tag{20.18}$$

20.3 Error Bounds on Integration

Computing areas, etc., of rational Bézier curves can be done by first approximating the rational curve with a polynomial curve (as discussed in chapter 20), then applying the closed form equations for polynomial curves. This section discusses how to determine error bounds on the resulting integration values.

From [Flo92], bounds for $\mathbf{P}'(t)$, $\mathbf{R}'(t)$, $\mathbf{P}(t)$, and $\mathbf{R}(t)$ are

$$\begin{aligned}
|x'_P(t)| &\leq B_{x'P} \\
&= n \cdot max_{0\leq i\leq n-1}|\triangle x_i|, && (20.19)\\
|y'_P(t)| &\leq B_{y'P} \\
&= n \cdot max_{0\leq i\leq n-1}|\triangle y_i|, && (20.20)\\
|x'_R(t)| &\leq B_{x'R} \\
&= r \cdot \left(\frac{max_{0\leq i\leq r}w_i}{min_{0\leq i\leq r}w_i}\right)^2 \cdot max_{0\leq i\leq r-1}|\triangle X_i|, && (20.21)\\
|y'_R(t)| &\leq B_{y'R} \\
&= r \cdot \left(\frac{max_{0\leq i\leq r}w_i}{min_{0\leq i\leq r}w_i}\right)^2 \cdot max_{0\leq i\leq r-1}|\triangle Y_i|, && (20.22)\\
|x_P(t)| &\leq B_{xP} \\
&= max_{0\leq i\leq n}|x_i|, && (20.23)\\
|y_P(t)| &\leq B_{yP} \\
&= max_{0\leq i\leq n}|y_i|, && (20.24)\\
|x_R(t)| &\leq B_{xR} \\
&= max_{0\leq i\leq r}|X_i|, && (20.25)\\
|y_R(t)| &\leq B_{yR} \\
&= max_{0\leq i\leq r}|Y_i|. && (20.26)
\end{aligned}$$

Of special interest is the error bound $\left|\int_{x_0}^{x_n} y_R dx_R - \int_{x_0}^{x_n} y_P dx_P\right|$ which expresses the error in applying integration formulae directly to polynomial approximations of rational curves:

$$\begin{aligned}
\int_{x_0}^{x_n} y_R dx_R - \int_{x_0}^{x_n} y_P dx_P &= \int_0^1 y_R x'_R dt - \int_0^1 y_P x'_P dt \\
&= \int_0^1 y_H x'_H dt - \int_0^1 y_P x'_P dt \\
&= \int_0^1 (y_H x'_H - y_H x'_P + y_H x'_P - y_P x'_P) dt \\
&= \int_0^1 y_H d(x_H - x_P) + \int_0^1 (y_H - y_P) x'_P dt \\
&= -\int_0^1 (x_H - x_P) y'_H dt + \int_0^1 (y_H - y_P) x'_P dt,
\end{aligned}$$

and hence

$$\begin{aligned}
\left|\int_{x_0}^{x_n} y_R dx_R - \int_{x_0}^{x_n} y_P dx_P\right| &\leq \int_0^1 |x_H - x_P| \cdot |y'_H| dt + \int_0^1 |y_H - y_P| \cdot |x'_P| dt \\
&\leq B_{xPH} \cdot B_{y'R} + B_{yPH} \cdot B_{x'P}. && (20.27)
\end{aligned}$$

Likewise,

$$\begin{aligned}
\left|\int_0^1 \mathbf{R}\wedge\frac{d\mathbf{R}}{dt}dt-\int_0^1 \mathbf{P}\wedge\frac{d\mathbf{P}}{dt}dt\right| &= \left|\int_0^1 \mathbf{H}\wedge\frac{d\mathbf{H}}{dt}dt-\int_0^1 \mathbf{P}\wedge\frac{d\mathbf{P}}{dt}dt\right| \\
&= \left|\int_0^1 \left\{\mathbf{H}\wedge(\frac{d\mathbf{H}}{dt}-\frac{d\mathbf{P}}{dt})\right\}dt+\int_0^1\left\{(\mathbf{H}-\mathbf{P})\wedge\frac{d\mathbf{P}}{dt}\right\}dt\right| \\
&= \left|\int_0^1 \begin{vmatrix} x_H & y_H \\ x'_H-x'_P & y'_H-y'_P\end{vmatrix}dt+\int_0^1\begin{vmatrix} x_H-x_P & y_H-y_P \\ x'_P & y'_P\end{vmatrix}dt\right| \\
&= \left|\int_0^1 x_H(y'_H-y'_P)dt-\int_0^1 y_H(x'_H-x'_P)dt+\int_0^1(x_H-x_P)y'_Pdt\right. \\
&\quad \left.-\int_0^1(y_H-y_P)x'_Pdt\right| \\
&= \left|-\int_0^1(y_H-y_P)x'_Hdt+\int_0^1(x_H-x_P)y'_Hdt+\int_0^1(x_H-x_P)y'_Pdt\right. \\
&\quad \left.-\int_0^1(y_H-y_P)x'_Pdt\right| \\
&= \left|\int_0^1(x_H-x_P)(y'_H+y'_P)dt-\int_0^1(y_H-y_P)(x'_H+x'_P)dt\right| \\
&\leq \int_0^1|x_H-x_P|(|y'_H|+|y'_P|)dt+\int_0^1|y_H-y_P|(|x'_H|+|x'_P|)dt \\
&\leq B_{xPH}\cdot(B_{y'R}+B_{y'P})+B_{yPH}\cdot(B_{x'R}+B_{x'P}). \qquad (20.28)
\end{aligned}$$

$$\begin{aligned}
\left|\int_{x_0}^{x_n} y_R^2dx_R-\int_{x_0}^{x_n}y_P^2dx_P\right| &= \left|\int_0^1 y_H^2x'_Hdt-\int_0^1 y_P^2x'_Pdt\right| \\
&= \left|\int_0^1 y_H^2(x'_H-x'_P)dt+\int_0^1 y_H(y_H-y_P)x'_Pdt+\right. \\
&\quad \left.\int_0^1 y_P(y_H-y_P)x'_Pdt\right| \\
&= \left|-2\int_0^1(x_H-x_P)y_Hy'_Hdt+\int_0^1(y_H-y_P)(y_H+y_P)x'_Pdt\right| \\
&\leq 2\int_0^1|x_H-x_P||y_H||y'_H|dt+\int_0^1|y_H-y_P|(|y_H|+|y_P|)|x'_P|dt \\
&\leq 2(B_{xPH}\cdot B_{y'R}\cdot B_{yR})+B_{yPH}\cdot B_{x'P}\cdot(B_{yR}+B_{yP}). \qquad (20.29)
\end{aligned}$$

$$\begin{aligned}
\left|\int_{x_0}^{x_n} x_R y_R dx_R - \int_{x_0}^{x_n} x_P y_P dx_P\right| &= \left|\int_0^1 x_H y_H x'_H dt - \int_0^1 x_P y_P x'_P dt\right| \\
&= \left|\int_0^1 x_H y_H (x'_H - x'_P) dt + \int_0^1 x_H (y_H - y_P) x'_P dt + \right. \\
&\quad \left. \int_0^1 (x_H - x_P) y_P x'_P dt\right| \\
&= \left|-\int_0^1 (x_H - x_P)(x'_H y_H + x_H y'_H) dt + \int_0^1 (y_H - y_P) x_H x'_P dt + \right. \\
&\quad \left. \int_0^1 (x_H - x_P) y_P x'_P dt\right| \\
&= \left|\int_0^1 (x_H - x_P)(-x'_H y_H - x_H y'_H + x'_P y_P) dt + \right. \\
&\quad \left. \int_0^1 (y_H - y_P) x_H x'_P dt\right| \\
&\leq \int_0^1 |x_H - x_P|(|x'_H||y_H| + |x_H||y'_H| + |x'_P||y_P|) dt + \\
&\quad \int_0^1 |y_H - y_P||x_H||x'_P| dt \\
&\leq B_{xPH} \cdot (B_{x'R} \cdot B_{yR} + B_{y'R} \cdot B_{xR} + B_{x'P} \cdot B_{yP}) + \\
&\quad B_{yPH} \cdot B_{x'P} \cdot B_{xR}. \qquad (20.30)
\end{aligned}$$

20.4 Examples

Since the integration error is a function of the hybrid curve bounding box size, that error can be controlled by recursively splitting $\mathbf{R}(t)$ in half until the bounding boxes are small enough to satisfy the ultimate integration error limit. Then, the exact integration formulae for polynomial curves can be used.

We have implemented this algorithm and compared it against two standard methods of numerical integration: the trapezoid rule, and Simpson's rule. It turns out that the trapezoid rule can be easily modified to return an error bound when dealing with rational Bézier curves. Imagine we are computing the area between the x axis and a rational Bézier curve. Then the top of each trapezoid is a line segment which approximates a segment of the rational Bézier curve. The convex hull of the control points of that curve segment then serves as a bound on how accurately a given trapezoid approximates the differential area.

The traditional way to determine an error bound for Simpson's rule requires us to obtain a bound on the fourth derivative of the integrand $x'(t)y(t)$ (for areas) or $\pi x'(t)y^2(t)$ (for volumes). This is possible, but for a degree r rational Bézier curve, the fourth derivative of the area (volume) integrand is a rational function of degree $48r$ ($64r$). Instead of using this robust error bound in our implementation of Simpson's rule, we used the heuristic of comparing the answer obtained using m intervals with the answer obtained using $2m$ intervals. This, of course, provides only a general indication of the error, not a guarantee of precision.

Figures 20.5 and 20.6 show two test cases in which the shaded area and volume of revolution about the x axis were computed. Figure 20.6 is actually a semi-circle expressed as a cubic rational

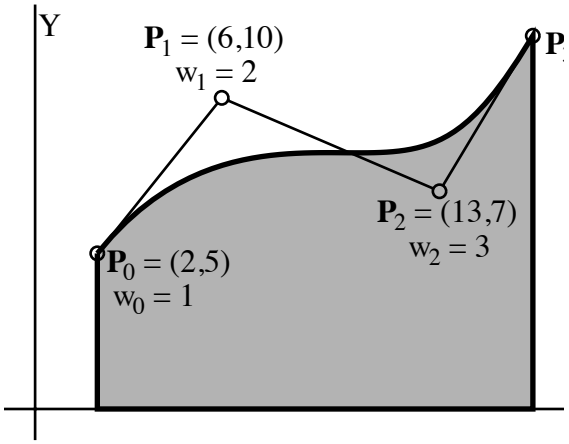


Figure 20.5: Example 1

Table 20.1: Execution times for Figure 5 (10^{-3} seconds)

Method	Tolerance = 10^{-3}		Tolerance = 10^{-6}	
	Area	Volume	Area	Volume
Degree 8 Hybrid	1	6	4	12
Degree 6 Hybrid	2	5	5	14
Simpson	4	15	7	61
Trapezoid	6	210	40	1560

Bézier curve.

Tables 20.4 and 20.4 list the execution time in milliseconds. Tests were run on a workstation rated at 65 Specmarks. The exact area of the semi-circle is $\pi/8 = .39269908\ldots$, and the computed area using hybrid curves is 0.39269906 using a tolerance of 10^{-6}, and 0.39272477 using a tolerance of 10^{-3}.

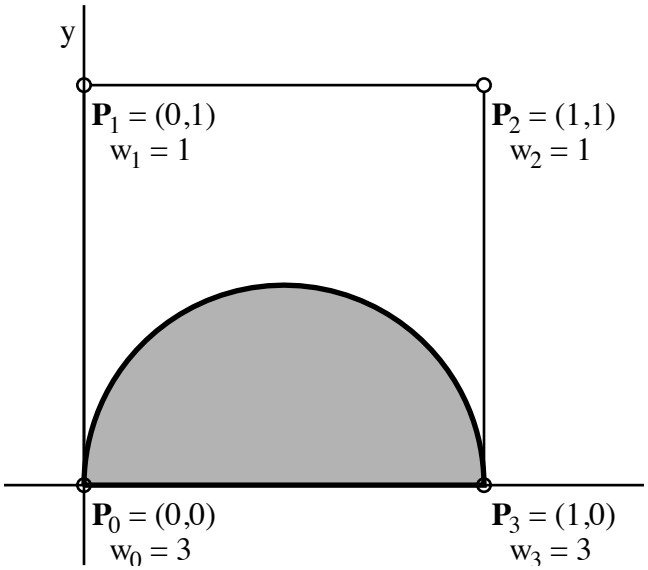


Figure 20.6: Example 2

Table 20.2: Execution times for Figure 6 (10^{-3} seconds)

Method	Tolerance = 10^{-3}		Tolerance = 10^{-6}	
	Area	Volume	Area	Volume
Degree 8 Hybrid	0.7	1.8	1.9	3.4
Degree 6 Hybrid	0.7	1.0	2.2	3.2
Simpson	0.2	0.5	1.0	1.0
Trapezoid	0.7	1.0	23	34

Bibliography

[AL94] W. Adams and P. Loustaunau. *An introduction to Gröbner bases*. American Mathematical Society, Providence, R.I., 1994.

[B$\acute{}$74] Pierre Bézier. Mathematical and practical possibilities of unisurf. In Robert E. Barnhill and Richard F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974.

[B$\acute{}$78] Pierre Bézier. General distortion of an ensemble of biparametric surfaces. *Computer-Aided Design*, 10:116–120, 1978.

[B$\ddot{}$81] Wolfgang Böhm. Generating the Bézier points of B-spline curves and surfaces. *Computer-Aided Design*, 13:355–356, 1981.

[B$\acute{}$86] Pierre Bézier. *The Mathematical Basis of the UNISURF CAD System*. Butterworths, London, 1986.

[Bal74] Alan A. Ball. Consurf. part 1: Introduction to the conic lofting tile. *Computer-Aided Design*, 6:243–249, 1974.

[Bal81] D. H. Ballard. Strip trees: a hierarchical representation for curves. *Communications of the ACM*, 24:310–321, 1981.

[BF89] R. L. Burden and J. D. Faires. *Numerical Analysis*. PWS-KENT Publishing Company, Boston, 1989.

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes Nach Einem Nulldimensionalen Polynomideal*. PhD thesis, Universitat Innsbruck, Austria, 1965.

[Buc85] B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Co., Netherlands, 1985.

[BW93] T. Becker and V. Weispfenning. *Gröbner Bases: A computational approach to commutative algebra*. Springer-Verlag, 1993.

[CLO92] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms*. Springer-Verlag, 1992.

[CR74] Edwin Catmull and R. Rom. A class of local interpolating splines. In Robert E. Barnhill and Richard F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974.

[Dav63] Philip J. Davis. *Interpolation and Approximation*. Dover, New York, 1963.

[dB78] Carl de Boor. *A Practical Guide to Splines*. Springer–Verlag, New York, 1978.

[Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, Boston, 1990.

[Flo92] Michael S. Floater. Derivatives of rational Bézier curves. *Computer Aided Geometric Design*, 9:161–174, 1992.

[FP79] Iver D. Faux and Michael J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, Chichester, 1979.

[FR87] Rida T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.

[FR88] Rida T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.

[GSA84] Ronald N. Goldman, Thomas W. Sederberg, and David C. Anderson. Vector elimination: A technique for the implicitization, inversion, and intersection of planar parametric rational polynomial curves. *Computer Aided Geometric Design*, 1:327–356, 1984.

[GZ84] Wang Guo-Zhao. The subdivision method for finding the intersection between two Bézier curves or surfaces. *Zhejiang University Journal special issue on computational geometry*, pages 108–119, 1984.

[Han78a] E. Hansen. A globally convergent interval method for computing and bounding real roots. *BIT*, 18:415–424, 1978.

[Han78b] E. Hansen. Interval forms of Newton's method. *Computing*, 20:153–163, 1978.

[HL93] J Hoschek and D Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, 1993.

[Hos83] Josef Hoschek. Dual Bézier curves and surfaces. In Robert E. Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer Aided Geometric Design*. North-Holland, 1983.

[Hos87] Josef Hoschek. Approximate conversion of spline curves. *Computer Aided Geometric Design*, 4:59–66, 1987.

[Hos88] Josef Hoschek. Intrinsic parameterization for approximation. *Computer Aided Geometric Design*, 5:27–31, 1988.

[KM83] P. A. Koparkar and S. P. Mudur. A new class of algorithms for the processing of parametric curves. *Computer-Aided Design*, 15:41–45, 1983.

[Kur80] A.G. Kurosh. *Higher Algebra*. MIR, 1980.

[Lac88] M. A. Lachance. Chebyshev economization for parametric surfaces. *Computer Aided Geometric Design*, 5:195–208, 1988.

[Liu87] D. Liu. Algorithms for computing area and volume bounded by Bézier curves and surfaces. *Mathematica Numerica Sinica*, 9:327–336, 1987.

[LM87] Tom Lyche and Knut Mørken. Knot removal for parametric B–spline curves and surfaces. *Computer Aided Geometric Design*, 4:217–230, 1987.

[LR80] Jeffery M. Lane and Richard F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. RAMI*, 2:35–46, 1980.

[MK84] S. P. Mudur and P. A. Koparkar. Interval methods for processing geometric objects. *IEEE Computer Graphics and Applications*, 2:7–17, 1984.

[Moo66] R. E. Moore. *Interval Analysis*. Prentice–Hall, Englewood Cliffs, NJ, 1966.

[Moo79] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[NLL90] Horst Nowacki, D. Liu, and X. Lu. Fairing Bézier curves with constraints. *Computer Aided Geometric Design*, 7:43–55, 1990.

[NSK90] Tomoyuki Nishita, Thomas W. Sederberg, and Masanori Kakimoto. Ray tracing trimmed rational surface patches. *Computer Graphics*, 24:337–345, 1990.

[Ove68] A. W. Overhauser. Analytic definition of curves and surfaces by parabolic blending. Technical report, Ford Motor Company, May 1968.

[Pat89] N. M. Patrikalakis. Approximate conversion of rational splines. *Computer Aided Geometric Design*, 6:155–165, 1989.

[Ram87] Lyle Ramshaw. Blossoming: A connect-the-dots approach to splines. Technical Report 19, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, June 1987.

[Ram89a] Lyle Ramshaw. Béziers and B-splines as multiaffine maps. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume 40 of *ASI Series F*, pages 757–776. NATO, Springer-Verlag, 1989.

[Ram89b] Lyle Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6:323–358, 1989.

[Rok75] J. Rokne. Reducing the degree of an interval polynomial. *Computing*, 14:5–14, 1975.

[Sab87] Malcolm A. Sabin. Envelope curves and surfaces. In Ralph R. Martin, editor, *The Mathematics of Surfaces II*, pages 413–418. Oxford University Press, 1987.

[SAG84] Thomas W. Sederberg, David C. Anderson, and Ronald N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing*, 28:72–84, 1984.

[Sal34] George Salmon. *Higher Plane Curves*. G.E.Stechert & Co., 1934.

[Sed89] Thomas W. Sederberg. Algorithm for algebraic curve intersection. *Computer-Aided Design*, 21:547–554, 1989.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, New York, 1982.

[SK91] Thomas W. Sederberg and Masanori Kakimoto. Approximating rational curves using polynomial curves. In Gerald Farin, editor, *NURBS for Curve and Surface Design*, pages 149–158, Philadelphia, 1991. SIAM.

[SN90] Thomas W. Sederberg and Tomoyuki Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22:538–549, 1990.

[Som51] D. M. Y. Sommerville. *Analytical Geometry of Three Dimensions*. Cambridge University Press, 1951.

[SP86a] Thomas W. Sederberg and Scott R. Parry. A comparison of curve-curve intersection algorithms. *Computer-Aided Design*, 18:58–63, 1986.

[SP86b] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20:151–160, 1986.

[Spe94] Melvin R. Spencer. *Polynomial Real Root Finding in Bernstein Form*. PhD thesis, Brigham Young University, 1994.

[SW87] Thomas W. Sederberg and Xuguang Wang. Rational hodographs. *Computer Aided Geometric Design*, 4:333–335, 1987.

[SWZ89] Thomas W. Sederberg, Scott C. White, and Alan K. Zundel. Fat arcs: a bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[SZSS98] T W Sederberg, J Zheng, D Sewell, and M Sabin. Non-uniform recursive subdivision surfaces. *Proceedings of SIGGRAPH 98*, pages 387–394, July 1998. ISBN 0-89791-999-8.

[Tim80] Harry G. Timmer. Alternative representation for parametric cubic curves and surfaces. *Computer-Aided Design*, 12:25–28, 1980.

[Win23] R. M. Winger. *An Introduction to Projective Geometry*. D. C. Heath and Company, 1923.

[WS93] Guo-Jin Wang and Thomas W. Sederberg. Convergence conditions for hermite approximation of rational functions. *submitted*, 1993.

[WW88] Mark A. Watkins and Andrew J. Worsey. Degree reduction of Bézier curves. *Computer-Aided Design*, 20:398–405, 1988.