Computer Science
and Scientific Computing

CURVES AND SURFACES FOR COMPUTER AIDED GEOMETRIC DESIGN

A Practical Guide

Fourth Edition

Gerald Farin

Curves and Surfaces for Computer-Aided Geometric Design

A Practical Guide

Fourth Edition

Gerald Farin
Department of Computer Science
Arizona State University
Tempe, Arizona

ACADEMIC PRESS
San Diego London Boston
New York Sydney Tokyo Toronto

LIMITED WARRANTY AND DISCLAIMER OF LIABILITY

ACADEMIC PRESS, INC. ("AP") AND ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION OR PRODUCTION OF THE ACCOMPANYING CODE ("THE PRODUCT") CANNOT AND DO NOT WARRANT THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THE PRODUCT. THE PRODUCT IS SOLD "AS IS" WITHOUT WARRANTY OF ANY KIND (EXCEPT AS HEREAFTER DESCRIBED), EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OF PERFORMANCE OR ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. AP WARRANTS ONLY THAT THE MAGNETIC DISKETTE(S) ON WHICH THE CODE IS RECORDED IS FREE FROM DEFECTS IN MATERIAL AND FAULTY WORKMANSHIP UNDER THE NORMAL USE AND SERVICE FOR A PERIOD OF NINETY (90) DAYS FROM THE DATE THE PRODUCT IS DELIVERED. THE PURCHASER'S SOLE AND EXCLUSIVE REMEDY IN THE EVENT OF A DEFECT IS EXPRESSLY LIMITED TO EITHER REPLACEMENT OF THE DISKETTE(S) OR REFUND OF THE PURCHASE PRICE, AT AP'S SOLE DISCRETION.

IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT, WARRANTY OR TORT (INCLUDING NEGLIGENCE), WILL AP OR ANYONE WHO HAS BEEN INVOLVED IN THE CREATION OR PRODUCTION OF THE PRODUCT BE LIABLE TO PURCHASER FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT OR ANY MODIFICATIONS THEREOF, OR DUE TO THE CONTENTS OF THE CODE, EVEN IF AP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

Any request for replacement of a defective diskette must be postage prepaid and must be accompanied by the original defective diskette, your mailing address and telephone number, and proof of date of purchase and purchase price. Send such requests, stating the nature of the problem, to Academic Press Customer Service, 6277 Sea Harbor Drive, Orlando, FL 32887, 1-800-321-5068. AP shall have no obligation to refund the purchase price or to replace a diskette based on claims of defects in the nature or operation of the Product.

Some states do not allow limitation on how long an implied warranty lasts, nor exclusions or limitations of incidental or consequential damage, so the above limitations and exclusions may not apply to you. This Warranty gives you specific legal rights, and you may also have other rights which vary from jurisdiction to jurisdiction.

THE RE-EXPORT OF UNITED STATES ORIGIN SOFTWARE IS SUBJECT TO THE UNITED STATES LAWS UNDER THE EXPORT ADMINISTRATION ACT OF 1969 AS AMENDED. ANY FURTHER SALE OF THE PRODUCT SHALL BE IN COMPLIANCE WITH THE UNITED STATES DEPARTMENT OF COMMERCE ADMINISTRATION REGULATIONS. COMPLIANCE WITH SUCH REGULATIONS IS YOUR RESPONSIBILITY AND NOT THE RESPONSIBILITY OF AP.

This is a volume in
COMPUTER SCIENCE AND SCIENTIFIC COMPUTING

Werner Rheinbolt, editor

This book is printed on acid free paper. ♾

ACADEMIC PRESS, INC.
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA
1300 Boylston Street, Chestnut Hill, MA 02167, USA
http://www.apnet.com

Academic Press Limited
24–28 Oval Road, London NW1 7DX, UK
http://www.hbuk.co.uk/ap/

Chapter 1 was written by P. Bézier.
Chapters 11 and 22 were written by W. Boehm.

Library of Congress Cataloging-in-Publication Data

Farin, Gerald E.
Curves and surfaces for computer aided geometric design : a practical guide / Gerald Farin.—4th ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-12-249054-1
1. Computer graphics. 2. Computer-aided design. I. Title. II. Series.
T385.F37 1996
006.6′01′516352—dc20 96-27090
CIP

Printed in the United States of America
96 97 98 99 00 MP 9 8 7 6 5 4 3 2 1

Contents

Preface

In the late 1950s, hardware became available that allowed the machining of 3D shapes out of blocks of wood or steel.[1] These shapes could then be used as stamps and dies for products such as the hood of a car. The bottleneck in this production method was soon found to be the lack of adequate software. In order to machine a shape using a computer, it became necessary to produce a computer-compatible description of that shape. The most promising description method was soon identified to be in terms of parametric surfaces. An example of this approach is provided by Plates I and III: Plate I shows the actual hood of a car; Plate III shows how it is represented internally as a collection of parametric surfaces.

The theory of parametric surfaces was well understood in differential geometry. Their potential for the representation of surfaces in a Computer Aided Design (CAD) environment was not known at all, however. The exploration of the use of parametric curves and surfaces can be viewed as the origin of Computer Aided Geometric Design (CAGD).

The major breakthroughs in CAGD were undoubtedly the theory of Bézier surfaces and Coons patches, later combined with B-spline methods. Bézier curves and surfaces were independently developed by P. de Casteljau at Citröen and by P. Bézier at Renault. De Casteljau's development, slightly earlier than Bézier's, was never published, and so the whole theory of polynomial curves and surfaces in Bernstein form now bears Bézier's name. CAGD became a discipline in its own right after the 1974 conference at the University of Utah (see Barnhill and Riesenfeld [31]).

This book presents a unified treatment of the main ideas of CAGD. During the last years, there has been a trend towards more geometric insight into curve and surface schemes; I have followed this trend by basing most concepts on simple geometric algorithms. For instance, a student will be able to construct Bézier curves with hardly any knowledge of the concept of a parametric curve. Later, when parametric curves are discussed in the context of differential geometry, one can apply differential geometry ideas to the concrete curves that were developed before.

The theory of Bézier curves (and rational Bézier curves) plays a central role in this book. They are numerically the most stable among all polynomial bases currently used in CAD systems, as was shown by Farouki and Rajan [196]. Thus Bézier curves are the ideal geometric standard for the representation of piecewise polynomial curves.

[1] A process that is now called CAM for *Computer Aided Manufacturing*.

Also, Bézier curves lend themselves easily to a geometric understanding of many CAGD phenomena and may, for instance, be used to derive the theory of rational and nonrational B-spline curves.

While this book offers a comprehensive treatment of the basic methods in curve and surface design, it is not meant to provide solutions to application-oriented problems that arise in practice. In particular, no algorithms are included to handle intersection, rendering, or offset problems. At present, no unified approach exists for these "geometry processing" problems. However, the material presented here should enable the reader to read the advanced literature on the topics; on offsets: [169], [182], [183], [282], [286], [289], [306], [420], [481]; on intersections: [28], [148], [150], [218], [223], [245], [265], [287], [290], [316], [325], [344], [380], [404], [423], [453], [455] [457]; on rendering: [1], [95], [205], [219], [326], [469].

Also, this is not a text on solid modeling. That branch of geometric modeling is concerned with the representation of objects that are enclosed by an assembly of surfaces, mostly very elementary ones such as planes, cylinders, or tori. As solid modeling systems are becoming fully accepted, they are incorporating the freeform curves and surfaces described in this book. The literature includes: [98], [186], [194], [276], [343], [354], [416], [487].

I have taught the material presented here in the form of both conference tutorials and university courses, typically at the intermediate level. The exercises are in three categories: simpler questions at the beginning of each Exercises section, harder questions marked by asterisks, and programming exercises marked by "P." Many of these programming exercises use data provided on the enclosed disk. Students should thus get a better feeling for "real" situations. In teaching this material, it is essential that students have access to computing and graphics facilities; practical experience greatly helps the understanding and appreciation of what might otherwise remain dry theory.

When I use this book as a text for a one-semester CAGD class at the lower graduate/upper undergraduate level, I typically cover the following chapters: the first half of Chapter 3, Chapters 4, 5, 6, 8, 9, 15, and 16. Material from other chapters is sprinkled in as needed.

The C programs on the disk are my implementations of some (but not all) of the most important methods described here. The programs were tested for many examples, but they are not meant to be "industrial strength." In general, no checks are made for consistency or correctness of input data. Also, modularity was valued higher than efficiency. The programs are in C, but with non-C users in mind—in particular, all modules should be easily translatable into FORTRAN.

This book would not have been possible without the stimulating environment provided by the CAGD group at Arizona State University (and formerly at the University of Utah), founded by Robert E. Barnhill. The book also greatly benefitted from numerous discussions I had with experts such as A. Nasri, T. Foley, Q. Fu, H. Hagen, J. Hoschek, G. Nielson, R. Patterson, and A. Worsey. I would also like to express my appreciation for the funding provided by the National Science Foundation

and the Department of Energy.[2] Special thanks go to D. C. Hansford for the numerous helpful suggestions concerning the mathematical side of the material, and also to W. Boehm, who was a critical and constructive consultant during the development of this book.

I am also grateful to the following people for suggesting improvements over the previous editions: S. Abi-Ezzi, N. Beebe, W. Boehm, R. E. Barnhill, E. Clapp, P. J. Davis, B. Hamann, D. Jung, F. Kimura, T.-W. Kim, S. Mann, G. Nielson, A. Swimmer, K. Voegele, W. Waggenspack, H. Wolters, M. Wozny, G. Wu, and Y. Yamaguchi.

Gerald Farin
Tempe, Ariz.

[2]Grants DCR-8502858 and DE-FG02-87ER25041, respectively.

Chapter 1

P. Bézier: How a Simple System Was Born

In order to solve CAD/CAM mathematical problems, many solutions have been offered, each being adapted to specific matters. Most of the systems were invented by mathematicians, but UNISURF, at least initially, was developed by mechanical engineers from the automotive industry. They were familiar with parts mainly described by lines and circles; fillets and other blending auxiliary surfaces were scantily defined, their final shape being left to the skill and experience of patternmakers and die-setters.

Circa 1960, designers of stamped parts such as car-body panels used French curves and sweeps, but in fact the final standard was the "master model," the shape of which, for many valid reasons, could not coincide with the curves traced on the drawing board. This problem resulted in discussions, arguments, haggling, retouches, expenses, and delay.

Obviously, no significant improvement could be expected as long as no method was devised that could provide an accurate, complete, and indisputable definition of freeform shapes.

Computing and numerical control (NC) had made great progress at that time, and it was certain that only numbers, transmitted from the drawing office to tool drawing office, manufacture, patternshop and inspection, could provide an answer. Drawings would of course remain necessary, but they would only be explanatory, their accuracy having no importance. Numbers would be the single, final definition.

Certainly, no system could be devised without the help of mathematics—yet designers, who would be in charge of operating such a system, had a good knowledge of geometry, especially descriptive geometry, but no basic training in algebra or analysis.

It should be noted that in France very little was known at that time about the work performed in the American aircraft industry. The papers of James Ferguson

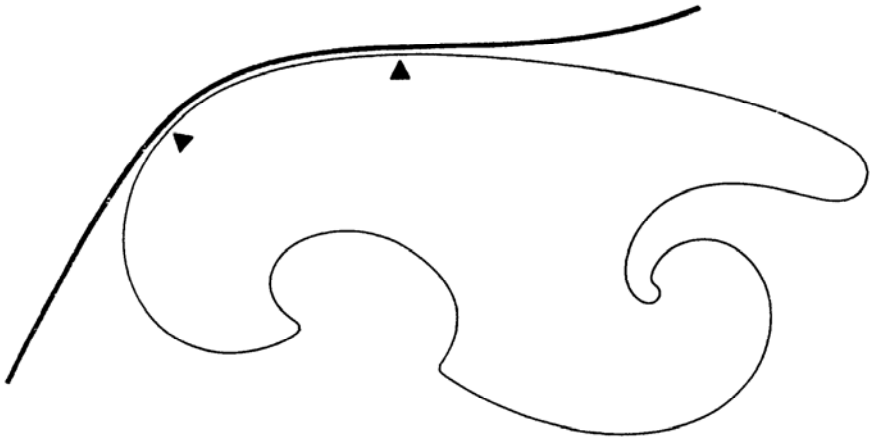

Figure 1.1: An arc of a hand-drawn curve is approximated by a part of a template.

were little disseminated before 1964; Citroën was secretive about the results obtained by Paul de Casteljau, and the famous technical report MAC-TR-41 (by S. A. Coons) did not appear until 1967. The works of W. Gordon and R. Riesenfeld were printed in 1974.

At the beginning, the concept of UNISURF was oriented toward geometry rather than analysis, but with the idea that every datum should be exclusively expressed by numbers.

For instance, an arc of a curve could be represented (Figure 1.1) by the coordinates, cartesian of course, of its limit points, i.e., A and B, together with their curvilinear abscissas, related by a grid traced on the edge.

The shape of the middle line of a sweep is a cube, if its cross-section is constant, its matter is homogeneous, and the effect of friction on the tracing cloth is neglected. However, it is difficult to take into account the length between endpoints. Moreover, the curves employed for software for NC machine tools, i.e., 2D milling machines, were lines, circles, and sometimes parabolas. Hence, a spline shape should be divided and subdivided into small arcs of circles placed end to end.

In order to transform an arc of circle into a portion of an ellipse, one could imagine (Figure 1.2) a square frame containing two sets of strings, whose intersections would be located on an arc of a circle. If the frame sides are hinged, flexing the hinges transforms the square into a diamond (Figure 1.3). The circle becomes an arc of an ellipse, which would be entirely defined as soon as the coordinates of points A, B, and C were known. If the hinged sides of the frame were replaced by pantographs (Figure 1.4), the diamond would become a parallelogram, and the arc of an ellipse is still defined by the coordinates of the three points A, B, and C (Figure 1.5).

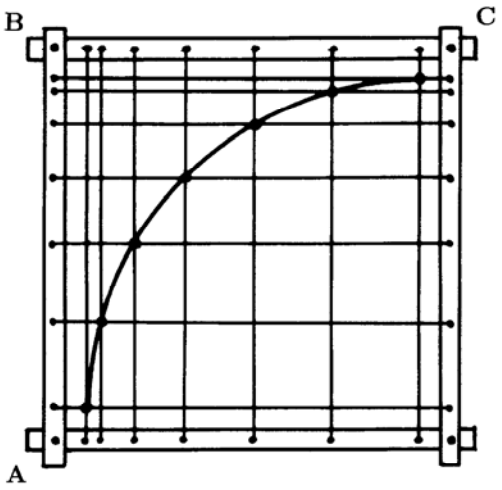

Figure 1.2: A circular arc is obtained by connecting the points in this rectangular grid.

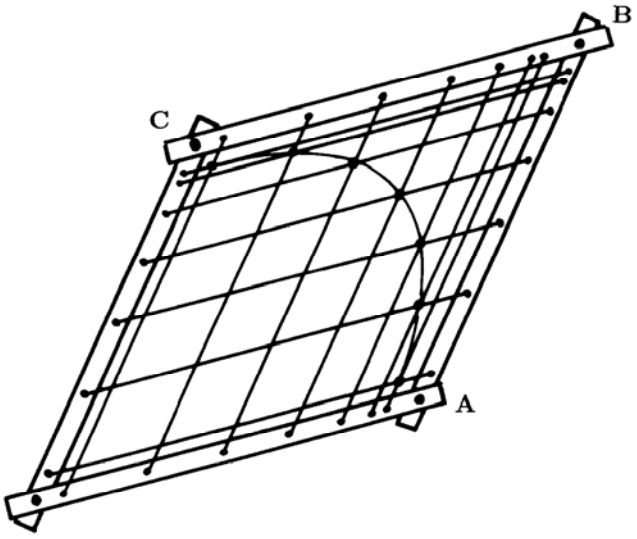

Figure 1.3: If the frame from the previous figure is sheared, an arc of an ellipse is obtained.

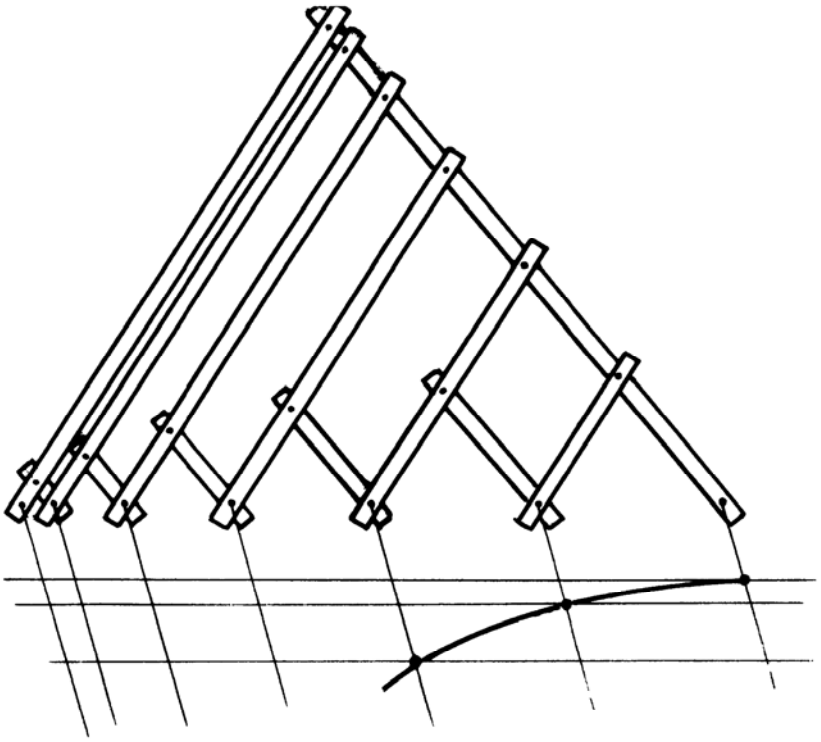

Figure 1.4: Pantograph construction of an arc of an ellipse.

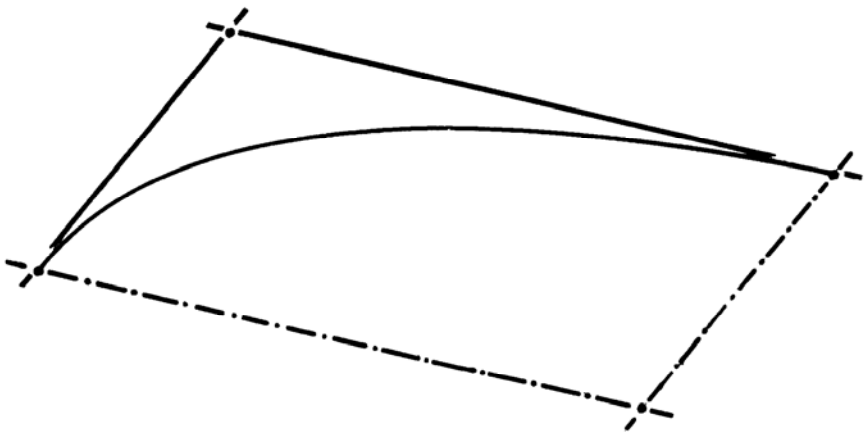

Figure 1.5: A "control polygon" for an arc of an ellipse.

Of course, this idea was not realistic, but it was easily replaced by the computation of coordinates of successive points of the curve. Harmonic functions were available with the help of analog computers, which were widely used at that time and gave excellent results.

However, employing only arcs of ellipses limited by conjugate diameters was far too restrictive, and a more flexible definition was required.

Another idea came from the practice of a speaker projecting, with a flashlight, a small cross or arrow onto a screen displaying a figure printed on a slide. Replacing the arrow with a curve and recording the exact location and orientation of the torch (Figure 1.6) would define the image of the curve projected on the wall of the drawing office. One could even imagine having a variety of slides, each of which would bear a specific curve: circles, parabola, astroid, etc.

Of course, this was not a realistic idea, because the focal plane of the zoom would seldom be square to the axis—an optician's nightmare! But the principle could be translated, via projective geometry and matrix computation, into cartesian coordinates.

At that time, designers defined the shape of a car body by cross-sections located 100 mm apart, and sometimes less. The advantage was that, from a drawing, one could derive templates for adjusting a clay model, a master, or a stamping tool. The drawback was that a stylist does not define a shape by cross-sections, but rather by so-called "character lines," which are seldom plane curves. Hence, a good system should be capable of manipulating and directly defining "space curves" or "freeform curves."

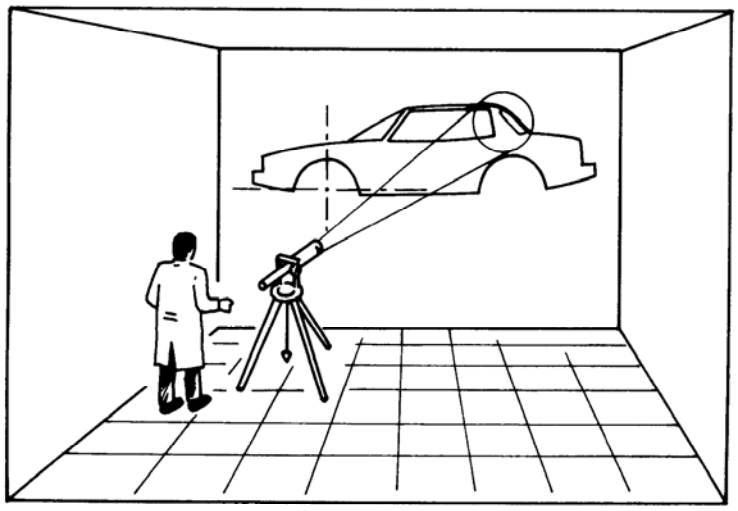

Figure 1.6: A projector producing a "template curve" on the drawing of an object.

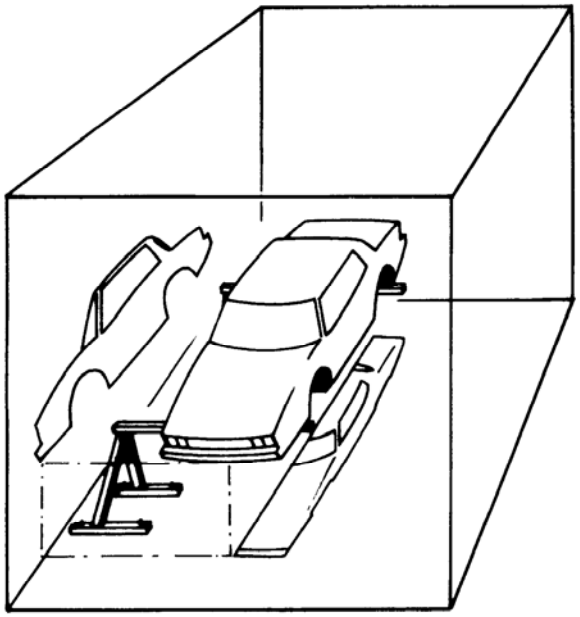

Figure 1.7: Two imaginary projections of a car.

Of course, one could imagine working alternately (Figure 1.7) on two projections of a space curve, but it is very unlikely that a stylist would accept such a solution.

Theoretically, at least, a space curve could be expressed by a sweep having a circular section, constrained by springs or counterweights (Figure 1.8), but this would prove quite impractical.

Would it not be best to revert to the basic idea of a frame? But instead of being inscribed in a square, the curve would be located in a cube (Figure 1.9) that could become any parallelepiped (Figure 1.10) by a linear transformation that is easy to compute. The first idea was to choose a basic curve that would be the intersection of two circular cylinders; the parallelepiped would be defined (Figure 1.10) by points O, X, Y, and Z, but it is more practical to put the basic vectors end to end so as to obtain a polygon OMNB (Figure 1.10), which directly defines the endpoint B and its tangent NB. Of course, points O, M, N, and B need not be coplanar and can define a space curve.

Polygons with three legs can define quite large a variety of curves (see Figure 3.4 in Section 3.3). To increase that variety, however, we can imagine to make use of cubes and hypercubes of any order (Figure 1.11) and the relevant polygons (Figure 1.13) (see Figure 3.4 in Section 3.3).

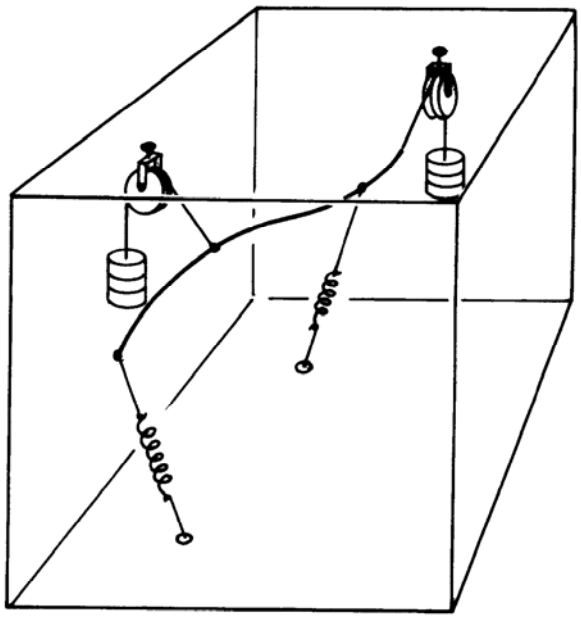

Figure 1.8: A curve held by springs.

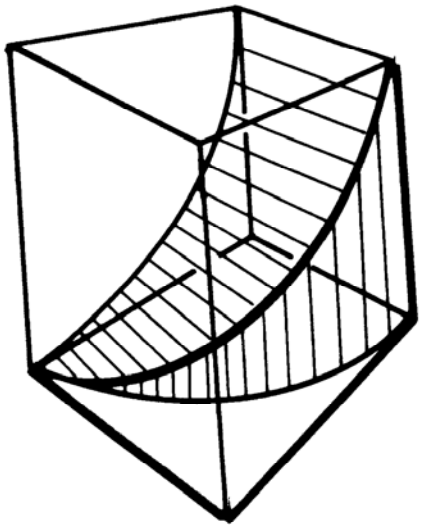

Figure 1.9: A curve defined inside a cube.

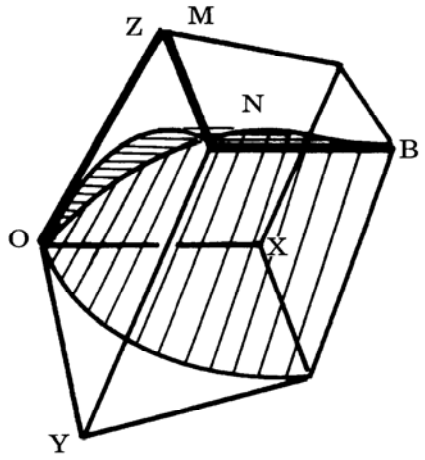

Figure 1.10: A curve defined inside a parallelepiped.

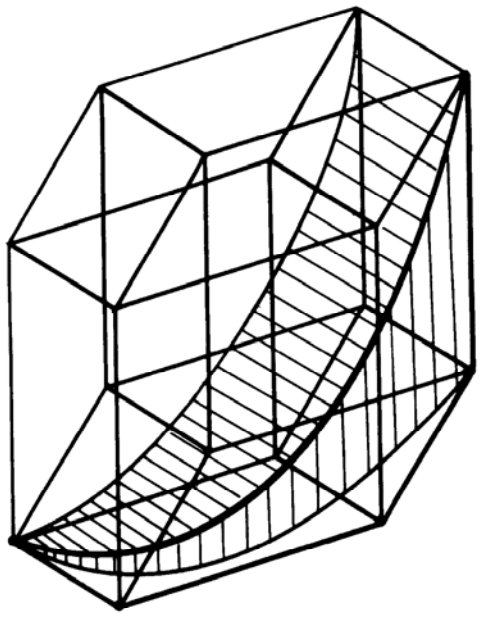

Figure 1.11: Higher order curves can be defined inside higher dimensional cubes.

At that moment, it became necessary to do away with harmonic functions and revert to polynomials. This was even more desirable because digital computers were gradually replacing analog computers. The polynomial functions were chosen according to the properties that were considered best: tangency, curvature, etc. Later it was discovered that they could be considered as sums of Bernstein's functions.

When it was suggested that these curves could replace sweeps and French curves, most stylists objected that they had invented their own templates and would not change. It was solemnly promised that their "secret" curves would be translated into secret listings and buried in the most secret part of the memory of the computer, and that only the stylists would have the key to the vaulted cellar. In fact, the standard curves were flexible enough and secret curves were soon forgotten. Designers and draftsmen easily understood the polygons and their relation to the shape of the corresponding curves.

In the traditional process of body engineering, a set of curves was carved in a 3D model, and interpolation between the curves was left to the experience of highly skilled patternmakers. However, in order to obtain a satisfactory numerical definition, the surface must be totally expressed with numbers.

At that time, around 1960, very little, if anything, had been published about biparametric patches. The basic idea of UNISURF came from a comparison with a process often used in foundries to obtain a core. Sand is compacted in a box (Figure 1.12), and the shape of the upper surface of the core is obtained by scraping off the surplus with a timber plank cut as a template. Of course, a shape obtained by such a method is relatively simple, because the shape of the plank is constant and that of the box edges is generally simple. To make the system more flexible, one might wish to change the shape of the template as it moves. In fact, this takes us back to a very old, and sometimes forgotten, definition of a surface: it is the locus of a curve that is

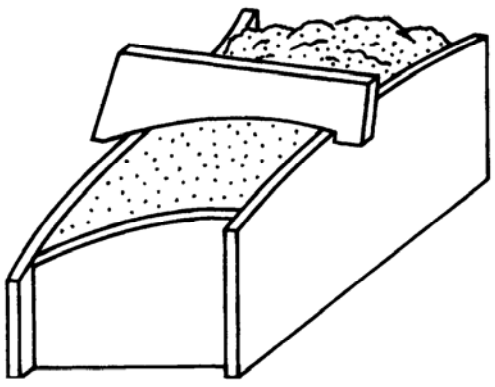

Figure 1.12: A surface is being obtained by scraping off excess material with wooden templates.

simultaneously moved and distorted. About 1970, a Dutch laboratory sculpted blocks of styrofoam with a flexible, electrically heated strip of steel, the shape of which was controlled by the flexion torque imposed on its extremities.

This process could not produce a large variety of shapes, but the principle could be translated into a mathematical solution. The guiding edges of the box are similar to the curves AB and CD of Figure 1.13, which can be considered as directrices of a surface defined by their characteristic polygon. If a curve such as EF is generatrix, defined by its own polygon, the ends of which run along lines AB and CD, and the intermediate vertices of the polygon are on curves GH and JK, then the surface ABDC is known as soon as the four polygons are defined. Connecting the corresponding vertices of the polygons defines the "characteristic net" of the patch, which plays the same role relative to the surface as the polygon of a curve. Hence, the cartesian coordinates of the points of the patch are computed according to the values of two parameters.

After this basic idea was expressed, a good many problems remained to be solved: choosing adequate functions, blending curves and patches, and dealing with degenerate patches, to name only a few. The solutions were a matter of relatively simple mathematics, the basic principle remaining untouched.

A system was thus progressively created. If we consider the way the initial idea evolved, we observe that the first solution—parallelogram, pantograph—is the result

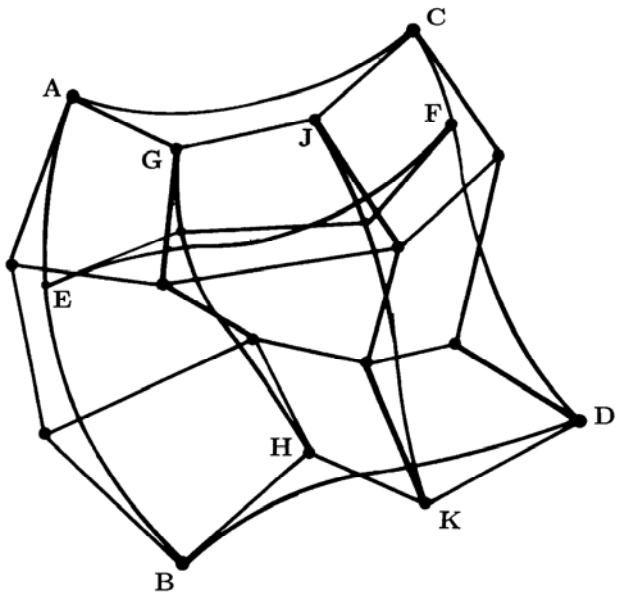

Figure 1.13: The characteristic net of a surface.

of an education oriented toward kinematics, the conception of mechanisms. Then geometry and optics appeared, which very likely came from army training, in which geometry, cosmography, and topography played an important part. Then reflection was oriented towards analysis, parametric spaces and finally, data processing, because a theory, as convenient as it may appear, must not impose too heavy a task on the computer and must be easily understood, at least in principle, by the operators.

Note that the various steps of this conception have a point in common: each idea must be related to the principle of a material system, simple and primitive though it may seem, on which a variable solution could be based.

An engineer defines what is to be done and how it can be done, not only describing the goal, but leading the way toward it.

Before looking any deeper into this subject, observe that elementary geometry played a major part. The subject should not gradually disappear from the training of a mechanical engineer. Each idea, each hypothesis, was expressed by a figure or a sketch, representing a mechanism. It would have been extremely difficult to build a purely mental image of a somewhat elaborate system without the help of pencil and paper. Let us consider, for instance, Figures 1.9 and 1.11; they are equivalent to Eqs. (4.7) and (15.6) in later chapters. These formulas, conveniently arranged, are best suited to express data to a computer. Most people, however, would better understand a simple figure than the equivalent algebraic expression.

Napoleon said: "A short sketch is better than a long report."

What parts are played by experience, by theory, and by imagination in the creation of a system? There is no definite answer. The importance of experience and of theoretical knowledge is not always clearly perceived. Imagination seems a gift, a godsend or the result of beneficial heredity. But is not imagination in fact the result of the maturation of knowledge gained during education and professional practice? Is it not born from facts apparently forgotten, stored in the dungeon of a distant part of memory, and suddenly remembered when circumstances call them back? Is not imagination partly based on the ability to connect notions which, at first sight, look quite unrelated, such as mechanics, electronics, optics, foundry, and data processing—to catch barely seen analogies—like Alice in Wonderland, to go "through the looking glass"?

Will psychologists someday be able to detect in humans a gift such as this that would be applicable to science and technology? Is it related to the sense of humor, which can detect unexpected relations between facts that look quite unconnected? Will we learn how to develop it? Or will it forever remain a gift, bestowed by pure chance on some people while others must rely on carefulness and rationality?

It is important that "sensible" people sometimes give free rein to imaginative people. "I succeeded," said Henry Ford, "because I let some fools try what wise people had advised me not to let them try."

Chapter 2

Introductory Material

2.1 Points and Vectors

When a designer or stylist works on an object, he or she does not think of that object in very mathematical terms. A point on the object would not be thought of as a triple of coordinates, but rather in functional terms: as a corner, the midpoint between two other points, and so on. The objective of this book, however, is to discuss objects that *are* defined in mathematical terms, the language that lends itself best to computer implementations. As a first step toward a mathematical description of an object, one therefore defines a *coordinate system* in which it will be described analytically.

The space in which we describe our object does not possess a preferred coordinate system—we have to define one ourselves. Many such systems could be picked (and some will certainly be more practical than others). But whichever one we choose, it should not affect any properties of the object itself. Our interest is in the object and not in its relationship to some arbitrary coordinate system. Therefore, the methods we develop must be independent of a particular choice of a coordinate system. We say that those methods must be *coordinate-free* or *coordinate-independent.*[1]

The concept of coordinate-free methods is stressed throughout this book. It motivates the strict distinction between points and vectors as discussed next. (For more details on this topic, see R. Goldman [230].)

We shall denote *points*, elements of three-dimensional euclidean (or point) space $\mathbb{E}^3$, by lowercase boldface letters such as **a**, **b**, etc. (The term "euclidean space" is used here because it is a relatively familiar term to most people. More correctly, we should have used the term "affine space.") A point identifies a location, often relative to other objects. Examples are the midpoint of a straight line segment or the center of gravity of a physical object.

[1] More mathematically, the geometry of this book is affine geometry. The objects that we will consider "live" in affine spaces, not in linear spaces.

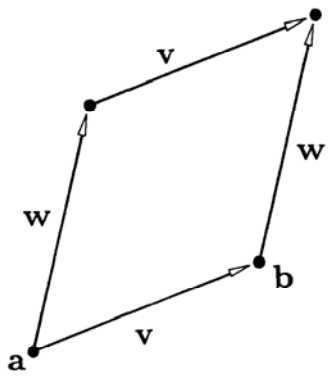

Figure 2.1: Points and vectors: vectors are not affected by translations.

The same notation (lowercase boldface) will be used for *vectors*, elements of three-dimensional linear (or vector) space $\mathbb{R}^3$. If we represent points or vectors by coordinates relative to some coordinate system, we shall adopt the convention of writing them as coordinate columns.

Although both points and vectors are described by triples of real numbers, we emphasize that there is a clear distinction between them: for any two points $\mathbf{a}$ and $\mathbf{b}$, there is a unique vector $\mathbf{v}$ that points from $\mathbf{a}$ to $\mathbf{b}$. It is computed by componentwise subtraction:

$$\mathbf{v} = \mathbf{b} - \mathbf{a}; \quad \mathbf{a}, \mathbf{b} \in \mathbb{E}^3, \quad \mathbf{v} \in \mathbb{R}^3.$$

On the other hand, given a vector $\mathbf{v}$, there are infinitely many pairs of points $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{v} = \mathbf{b} - \mathbf{a}$. For if $\mathbf{a}$, $\mathbf{b}$ is one such pair and if $\mathbf{w}$ is an arbitrary vector, then $\mathbf{a} + \mathbf{w}$, $\mathbf{b} + \mathbf{w}$ is another such pair since $\mathbf{v} = (\mathbf{b} + \mathbf{w}) - (\mathbf{a} + \mathbf{w})$ also. Figure 2.1 illustrates this fact.

Assigning the point $\mathbf{a} + \mathbf{w}$ to every point $\mathbf{a} \in \mathbb{E}^3$ is called a *translation*, and the above asserts that vectors are invariant under translations while points are not.

Elements of point space $\mathbb{E}^3$ can only be *subtracted* from each other—this operation yields a vector. They cannot be *added*—this operation is not defined for points. (It is defined for vectors.) Figure 2.2 gives an example.

However, addition-like operations are defined for points: they are *barycentric combinations*.[2] These are weighted sums of points where the weights sum to one:

$$\mathbf{b} = \sum_{j=0}^{n} \alpha_j \mathbf{b}_j; \qquad \begin{array}{l} \mathbf{b}_j \in \mathbb{E}^3 \\ \alpha_0 + \cdots + \alpha_n = 1 \end{array}. \tag{2.1}$$

[2]They are also called *affine combinations*.

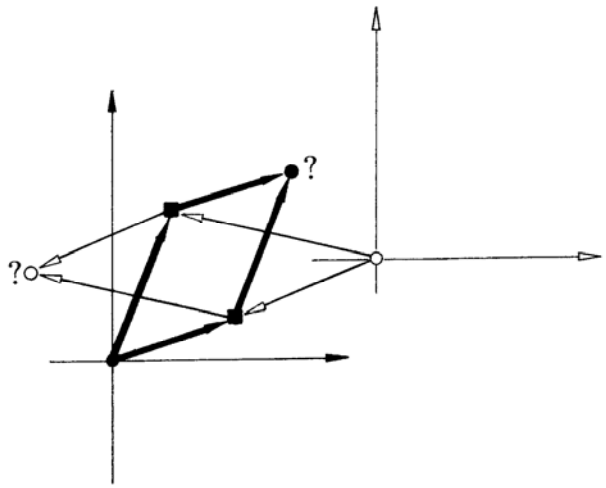

Figure 2.2: Addition of points: this is not a well-defined operation, since different coordinate systems would produce different "solutions." The two points to be "added" are marked by solid squares.

At first glance, this looks like an undefined summation of points, but we can rewrite (2.1) as

$$\mathbf{b} = \mathbf{b}_0 + \sum_{j=1}^{n} \alpha_j (\mathbf{b}_j - \mathbf{b}_0),$$

which is clearly the sum of a point and a vector.

An example of a barycentric combination is the centroid $\mathbf{g}$ of a triangle with vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, given by

$$\mathbf{g} = \frac{1}{3}\mathbf{a} + \frac{1}{3}\mathbf{b} + \frac{1}{3}\mathbf{c}.$$

The term "barycentric combination" is derived from "barycenter," meaning "center of gravity." The origin of this formulation is in physics: if the $\mathbf{b}_j$ are centers of gravity of objects with masses m_j, then their center of gravity $\mathbf{b}$ is located at $\mathbf{b} = \sum m_j \mathbf{b}_j / \sum m_j$ and has the combined mass $\sum m_j$. (If some of the m_j are negative, the notion of electric charges may provide a better analogy; see Coxeter [119], p. 214.) Since a common factor in the m_j is immaterial for the determination of the center of gravity, we may normalize them by requiring $\sum m_j = 1$.

An important special case of barycentric combinations are the *convex combinations*. These are barycentric combinations where the coefficients α_j, in addition to

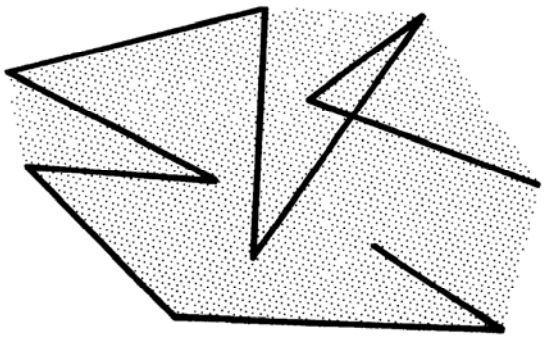

Figure 2.3: Convex hulls: a point set (a polygon) and its convex hull, shown shaded.

summing to one, are also nonnegative. A convex combination of points is always "inside" those points, which is an observation that leads to the definition of the *convex hull* of a point set: this is the set that is formed by all convex combinations of a point set. Figure 2.3 gives an example; see also Exercises. More intuitively, the convex hull of a set is formed as follows: for a 2D set, imagine a string that is loosely circumscribed around the set, with nails driven through the points in the set. Now pull the string tight—it will become the boundary of the convex hull.

The convex hull of a point set is a *convex set*. Such a set is characterized by the following: for any two points in the set, the straight line connecting them is also contained in the set. Examples are ellipses or parallelograms. It is an easy exercise to verify that affine maps (see next section) preserve convexity.

Let us return to barycentric combinations, which generate points from points. If we want to generate a *vector* from a set of points, we may write

$$\mathbf{v} = \sum_{j=0}^{n} \sigma_j \mathbf{p}_j,$$

where we have a new restriction on the coefficients: Now we must demand that the σ_j sum to zero.

If we are given an equation of the form

$$\mathbf{a} = \sum \beta_j \mathbf{b}_j,$$

and $\mathbf{a}$ is supposed to be a point, then we must be able to split the sum into three groups:

$$\mathbf{a} = \sum_{\sum \beta_j = 1} \beta_j \mathbf{b}_j + \sum_{\sum \beta_j = 0} \beta_j \mathbf{b}_j + \sum_{\text{remaining } \beta' s} \beta_j \mathbf{b}_j.$$

Then the $\mathbf{b}_j$ in the first sum are points, and those in the second sum may be interpreted as either points or vectors. The $\mathbf{b}_j$ in the third one are vectors. While the second and third sums may be empty, the first one must contain at least one term.

The interplay between points and vectors is unusual at first. Later, it will turn out to be of invaluable theoretical and practical help. For example, we can perform quick *type checking* when we derive formulas. If the point coefficients fail to add up to one or zero—depending on the context—we know that something has gone wrong. In a more formal way, T. DeRose has developed the concept of "geometric programming," a graphics language that automatically performs type checks [145], [146]. R. Goldman's article [230] treats the validity of point/vector operations in more detail.

2.2 Affine Maps

Most of the transformations that are used to position or scale an object in a computer graphics or CAD environment are *affine* maps. (More complicated, so-called projective maps are discussed in Chapter 13.) The term "affine map" is due to L. Euler; affine maps were first studied systematically by F. Moebius [361].

The fundamental operation for points is the barycentric combination. We will thus base the definition of an affine map on the notion of barycentric combinations. *A map Φ that maps $\mathbb{E}^3$ into itself is called an affine map if it leaves barycentric combinations invariant.* So if

$$\mathbf{x} = \sum \alpha_j \mathbf{a}_j; \quad \sum \alpha_j = 1, \quad \mathbf{x}, \mathbf{a}_j \in \mathbb{E}^3$$

and Φ is an affine map, then also

$$\Phi\mathbf{x} = \sum \alpha_j \Phi\mathbf{a}_j; \quad \Phi\mathbf{x}, \Phi\mathbf{a}_j \in \mathbb{E}^3. \tag{2.2}$$

This definition looks fairly abstract, yet has a simple interpretation. The expression $\mathbf{x} = \sum \alpha_j \mathbf{a}_j$ specifies how we have to weight the points $\mathbf{a}_j$ so that their weighted average is $\mathbf{x}$. This relation is still valid if we apply an affine map to all points $\mathbf{a}_j$ and to $\mathbf{x}$. As an example, the midpoint of a straight line segment will be mapped to the midpoint of the affine image of that straight line segment. Also, the centroid of a number of points will be mapped to the centroid of the image points.

Let us now be more specific. In a given coordinate system, a point $\mathbf{x}$ is represented by a coordinate triple, which we also denote by $\mathbf{x}$. An affine map now takes on the familiar form

$$\Phi\mathbf{x} = A\mathbf{x} + \mathbf{v}, \tag{2.3}$$

where A is a 3×3 matrix and $\mathbf{v}$ is a vector from $\mathbb{R}^3$.

A simple computation verifies that (2.3) does in fact describe an affine map, i.e., that barycentric combinations are preserved by maps of that form. For the following, recall that $\sum \alpha_j = 1$:

$$\begin{aligned}\Phi\left(\sum \alpha_j \mathbf{a}_j\right) &= A\left(\sum \alpha_j \mathbf{a}_j\right) + \mathbf{v} \\ &= \sum \alpha_j A\mathbf{a}_j + \sum \alpha_j \mathbf{v} \\ &= \sum \alpha_j \left(A\mathbf{a}_j + \mathbf{v}\right) \\ &= \sum \alpha_j \Phi \mathbf{a}_j,\end{aligned}$$

which concludes our proof. It also shows that the inverse of our initial statement is true as well: every map of the form (2.3) represents an affine map.

Some examples of affine maps:

The identity. It is given by $\mathbf{v} = \mathbf{0}$, the zero vector, and by $A = I$, the identity matrix.

A translation. It is given by $A = I$, and a *translation vector* $\mathbf{v}$.

A scaling. It is given by $\mathbf{v} = \mathbf{0}$ and by a diagonal matrix A. The diagonal entries define by how much each component of the preimage $\mathbf{x}$ is to be scaled.

A rotation. If we rotate around the z-axis, then $\mathbf{v} = \mathbf{0}$ and

$$A = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

A shear. An example is given by $\mathbf{v} = \mathbf{0}$ and

$$A = \begin{bmatrix} 1 & a & b \\ 0 & 1 & c \\ 0 & 0 & 1 \end{bmatrix}.$$

This family of shears maps the (x, y)-plane onto itself.

A parallel projection. All of $\mathbb{E}^3$ is projected onto the (x, y)-plane if we set

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and $\mathbf{v} = \mathbf{0}$. Note that A may also be viewed as a scaling matrix.

Figure 2.4: A shear: this affine map is used in font design in order to generate slanted fonts. Left: original letter; right: slanted letter.

We give one example of an affine map that is important in the area of *font design*. A given letter is subjected to a 2D shear and thus transforms into a slanted letter. Figure 2.4 gives an example; see also Section 8.5.

An important special case of affine maps are the *euclidean maps*, also called *rigid body motions*. They are characterized by orthonormal matrices A that are defined by the property $A^{\mathrm{T}}A = I$. Euclidean maps leave lengths and angles unchanged; the most important examples are rotations and translations.

Affine maps can be combined, and a complicated map may be decomposed into a sequence of simpler maps. Every affine map can be composed of translations, rotations, shears, and scalings.

The *rank* of A has an important geometric interpretation: if $\mathrm{rank}(A) = 3$, then the affine map Φ maps three-dimensional objects to three-dimensional objects. If the rank is less than three, Φ is a parallel projection onto a plane (rank $= 2$) or even onto a straight line (rank $= 1$).

An affine map of $\mathbb{E}^2$ to $\mathbb{E}^2$ is uniquely determined by a (nondegenerate) triangle and its image. Thus any two triangles determine an affine map of the plane onto itself. In $\mathbb{E}^3$, an affine map is uniquely defined by a (nondegenerate) tetrahedron and its image.

More important facts about affine maps are discussed in the following section.

2.3 Linear Interpolation

Let $\mathbf{a}$, $\mathbf{b}$ be two distinct points in $\mathbb{E}^3$. The set of all points $\mathbf{x} \in \mathbb{E}^3$ of the form

$$\mathbf{x} = \mathbf{x}(t) = (1 - t)\mathbf{a} + t\mathbf{b}; \quad t \in \mathbb{R} \tag{2.4}$$

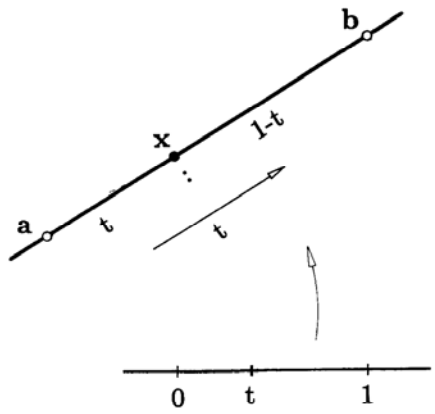

Figure 2.5: Linear interpolation: two points **a**, **b** define a straight line through them. The point **x** divides the straight line segment between **a** and **b** in the ratio $t : 1 - t$.

is called the *straight line* through **a** and **b**. Any three (or more) points on a straight line are said to be *collinear*.

For $t = 0$ the straight line passes through **a**, and for $t = 1$ it passes through **b**. For $0 \leq t \leq 1$, the point **x** is between **a** and **b**, while for all other values of t it is outside; see Figure 2.5.

Equation (2.4) represents **x** as a barycentric combination of two points in $\mathbb{E}^3$. The same barycentric combination holds for the three points $0, t, 1$ in $\boldsymbol{E}^1$: $t = (1 - t) \cdot 0 + t \cdot 1$. So t is related to 0 and 1 by the same barycentric combination that relates **x** to **a** and **b**. However, by the definition of affine maps, the three points **a**, **x**, **b** in three-space are an affine map of the three points $0, t, 1$ in one-space! *Thus linear interpolation is an affine map of the real line onto a straight line in* $\mathbb{E}^3$.[3]

It is now almost a tautology when we state: *Linear interpolation is affinely invariant.* Written as a formula: if Φ is an affine map of $\mathbb{E}^3$ onto itself, and (2.4) holds, then also

$$\Phi\mathbf{x} = \Phi\big((1 - t)\mathbf{a} + t\mathbf{b}\big) = (1 - t)\Phi\mathbf{a} + t\Phi\mathbf{b}. \tag{2.5}$$

Closely related to linear interpolation is the concept of *barycentric coordinates*, due to Moebius [361]. Let **a**, **x**, **b** be three collinear points in $\mathbb{E}^3$:

$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b}; \qquad \alpha + \beta = 1. \tag{2.6}$$

Then α and β are called *barycentric coordinates* of **x** with respect to **a** and **b**. Note that by our previous definitions, **x** is a barycentric combination of **a** and **b**.

[3] Strictly speaking, we should therefore use the term "affine interpolation" instead of "linear interpolation." We use "linear interpolation" because its use is so widespread.

The connection between barycentric coordinates and linear interpolation is obvious: we have $\alpha = 1 - t$ and $\beta = t$. This shows, by the way, that barycentric coordinates do not always have to be positive: For $t \notin [0, 1]$, either α or β is negative. For any three collinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$, the barycentric coordinates of $\mathbf{b}$ with respect to $\mathbf{a}$ and $\mathbf{c}$ are given by

$$\alpha = \frac{\text{vol}_1(\mathbf{b}, \mathbf{c})}{\text{vol}_1(\mathbf{a}, \mathbf{c})},$$
$$\beta = \frac{\text{vol}_1(\mathbf{a}, \mathbf{b})}{\text{vol}_1(\mathbf{a}, \mathbf{c})},$$

where vol_1 denotes the one-dimensional volume, which is the signed distance between two points. Barycentric coordinates are not only defined on a straight line, but also on a plane. Section 2.6 has more details.

Another important concept in this context is that of *ratios*. The ratio of three collinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ is defined by

$$\text{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{\text{vol}_1(\mathbf{a}, \mathbf{b})}{\text{vol}_1(\mathbf{b}, \mathbf{c})}. \tag{2.7}$$

If α and β are barycentric coordinates of $\mathbf{b}$ with respect to $\mathbf{a}$ and $\mathbf{c}$, it follows that

$$\text{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{\beta}{\alpha}. \tag{2.8}$$

The barycentric coordinates of a point do not change under affine maps, and neither does their quotient. Thus the ratio of three collinear points is not affected by affine transformations. So if (2.8) holds, then also

$$\text{ratio}(\Phi\mathbf{a}, \Phi\mathbf{b}, \Phi\mathbf{c}) = \frac{\beta}{\alpha}, \tag{2.9}$$

where Φ is an affine map. This property may be used to *compute* ratios efficiently. Instead of using square roots to compute the distances between points $\mathbf{a}$, $\mathbf{x}$, and $\mathbf{b}$, we would project them onto one of the coordinate axes and then use simple differences of their x- or y-coordinates.[4] This method works since parallel projection is an affine map!

Equation (2.9) states that *affine maps are ratio preserving*. This property may be used to define affine maps. Every map that takes straight lines to straight lines and is ratio preserving is an affine map.

The concept of ratio preservation may be used to derive another useful property of linear interpolation. We have defined the straight line segment $[\mathbf{a}, \mathbf{b}]$ to be the affine image of the *unit interval* $[0, 1]$, but we can also view that straight line segment as the affine image of any interval $[a, b]$.

[4] But be sure to avoid projection onto the x-axis if the three points are parallel to the y-axis!

The interval $[a, b]$ may itself be obtained by an affine map from the interval $[0, 1]$ or vice versa. With $t \in [0, 1]$ and $u \in [a, b]$, that map is given by $t = (u-a)/(b-a)$. The interpolated point on the straight line is now given by both

$$\mathbf{x}(t) = (1 - t)\mathbf{a} + t\mathbf{b}$$

and

$$\mathbf{x}(u) = \frac{b-u}{b-a}\mathbf{a} + \frac{u-a}{b-a}\mathbf{b}. \tag{2.10}$$

Since a, u, b and $0, t, 1$ are in the same ratio as the triple $\mathbf{a}, \mathbf{x}, \mathbf{b}$, we have shown that *linear interpolation is invariant under affine domain transformations.* By affine domain transformation, we simply mean an affine map of the real line onto itself. The parameter t is sometimes called a *local parameter* of the interval $[a, b]$.

A concluding remark: we have demonstrated the interplay between the two concepts of linear interpolation and ratios. In this book, we will often describe methods by saying that points have to be collinear and must be in a given ratio. This is the geometric (descriptive) equivalent of the algebraic (algorithmic) statement that one of the three points may be obtained by linear interpolation from the other two.

2.4 Piecewise Linear Interpolation

Let $\mathbf{b}_0, \ldots, \mathbf{b}_n \in \mathbb{E}^3$ form a *polygon* $\mathbf{B}$. A polygon consists of a sequence of straight line segments, each interpolating to a pair of points $\mathbf{b}_i, \mathbf{b}_{i+1}$. It is therefore also called the *piecewise linear interpolant* $\mathcal{PL}$ to the points $\mathbf{b}_i$. If the points $\mathbf{b}_i$ lie on a curve $\mathbf{c}$, then $\mathbf{B}$ is said to be a piecewise linear interpolant to $\mathbf{c}$, and we write

$$\mathbf{B} = \mathcal{PL}\mathbf{c}. \tag{2.11}$$

One of the important properties of piecewise linear interpolation is *affine invariance*. If the curve $\mathbf{c}$ is mapped onto a curve $\Phi\mathbf{c}$ by an affine map Φ, then the piecewise linear interpolant to $\Phi\mathbf{c}$ is the affine map of the original piecewise linear interpolant:

$$\mathcal{PL}\,\Phi\mathbf{c} = \Phi\ \mathcal{PL}\,\mathbf{c}. \tag{2.12}$$

Another property is the *variation diminishing property*. Consider a continuous curve $\mathbf{c}$, a piecewise linear interpolant $\mathcal{PL}\mathbf{c}$, and an arbitrary plane. Let cross $\mathbf{c}$ be the number of crossings that the curve $\mathbf{c}$ has with this plane, and let cross $\mathcal{PL}\mathbf{c}$ be the number of crossings that the piecewise linear interpolant has with this plane. (Special cases may arise; see Section 2.9.) Then we always have

$$\text{cross}\ \mathcal{PL}\,\mathbf{c} \leq \text{cross}\,\mathbf{c}. \tag{2.13}$$

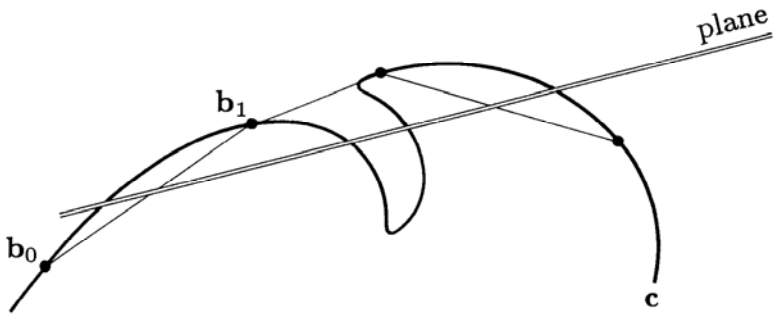

Figure 2.6: The variation diminishing property: a piecewise linear interpolant to a curve has no more intersections with any plane than the curve itself.

This property follows from a simple observation: consider two points $\mathbf{b}_i$, $\mathbf{b}_{i+1}$. The straight line segment through them can cross a given plane at one point at most, while the curve segment from $\mathbf{c}$ that connects them may cross the same plane in many arbitrary points. The variation diminishing property is illustrated in Figure 2.6.

2.5 Menelaos' Theorem

We use the concept of piecewise linear interpolation to prove one of the most important geometric theorems for the theory of CAGD: Menelaos' theorem. This theorem can be used for the proof of many constructive algorithms, and its importance was already realized by de Casteljau [134].

Referring to Figure 2.7, let

$$\begin{aligned}
\mathbf{a}_t &= (1-t)\mathbf{p}_1 + t\mathbf{p}_2, \\
\mathbf{a}_s &= (1-s)\mathbf{p}_1 + s\mathbf{p}_2, \\
\mathbf{b}_t &= (1-t)\mathbf{p}_2 + t\mathbf{p}_3, \\
\mathbf{b}_s &= (1-s)\mathbf{p}_2 + s\mathbf{p}_3.
\end{aligned}$$

Let $\mathbf{c}$ be the intersection of the straight lines $\mathbf{a}_t\mathbf{b}_t$ and $\mathbf{a}_s\mathbf{b}_s$. Then

$$\text{ratio}(\mathbf{a}_t, \mathbf{c}, \mathbf{b}_t) = \frac{s}{1-s} \quad \text{and} \quad \text{ratio}(\mathbf{a}_s, \mathbf{c}, \mathbf{b}_s) = \frac{t}{1-t}. \tag{2.14}$$

For a proof, we simply show that $\mathbf{c}$ satisfies the two equations

$$\mathbf{c} = (1-s)\mathbf{a}_t + s\mathbf{b}_t \quad \text{and} \quad \mathbf{c} = (1-t)\dot{\mathbf{a}}_s + t\mathbf{b}_s,$$

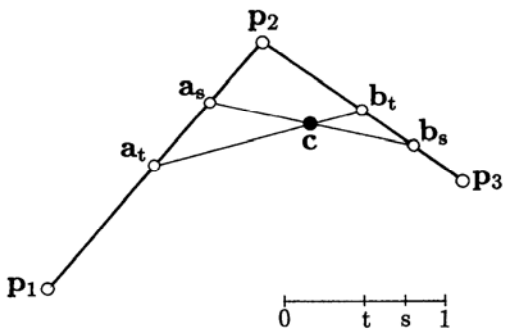

Figure 2.7: Menelaos' theorem: the point **c** may be obtained from linear interpolation at t or at s.

which is straightforward. Notice also that the four collinear points $\mathbf{p}_1$, $\mathbf{a}_t$, $\mathbf{a}_s$, $\mathbf{p}_2$ as well as the four collinear points $\mathbf{p}_2$, $\mathbf{b}_t$, $\mathbf{b}_s$, $\mathbf{p}_3$ are affine maps of the four points 0, t, s, 1 on the real line.

Equation (2.14) is a "CAGD version" of the original Menelaos' theorem, which may be stated as (see Coxeter [119]):

$$\text{ratio}(\mathbf{b}_s, \mathbf{b}_t, \mathbf{p}_2) \cdot \text{ratio}(\mathbf{p}_2, \mathbf{a}_t, \mathbf{a}_s) \cdot \text{ratio}(\mathbf{a}_s, \mathbf{c}, \mathbf{b}_s) = -1. \tag{2.15}$$

The proof of (2.15) is a direct consequence of (2.14). Note the ordering of points in the second ratio! Menelaos' theorem is closely related to Ceva's, which is given in Section 2.6.

2.6 Barycentric Coordinates in the Plane

Barycentric coordinates were discussed in Section 2.3, where they were used in connection with straight lines. Now we will use them as coordinate systems when dealing with the plane. Planar barycentric coordinates are at the origin of affine geometry—they were first introduced by F. Moebius in 1827; see his collected works [361].

Consider a triangle with vertices **a**, **b**, **c** and a fourth point **p**, all in $\mathbb{E}^2$. It is always possible to write **p** as a barycentric combination of **a**, **b**, **c**:

$$\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}. \tag{2.16}$$

A reminder: if (2.16) is to be a barycentric combination (and hence geometrically meaningful), we require that

$$u + v + w = 1. \tag{2.17}$$

The coefficients $\mathbf{u} := (u, v, w)$ are called *barycentric coordinates* of $\mathbf{p}$ with respect to $\mathbf{a}, \mathbf{b}, \mathbf{c}$. We will often drop the distinction between the barycentric coordinates of a point and the point itself; we then speak of "the point $\mathbf{u}$."

If the four points $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and $\mathbf{p}$ are given, we can always determine $\mathbf{p}$'s barycentric coordinates u, v, w: Eqs. (2.16) and (2.17) can be viewed as a linear system of three equations [recall that (2.16) is shorthand for two scalar equations] in three unknowns u, v, w. The solution is obtained by an application of Cramer's rule:

$$u = \frac{\text{area}(\mathbf{p}, \mathbf{b}, \mathbf{c})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})}, \quad v = \frac{\text{area}(\mathbf{a}, \mathbf{p}, \mathbf{c})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})}, \quad w = \frac{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{p})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})}. \tag{2.18}$$

Actually, Cramer's rule makes use of determinants; they are related to areas by the identity

$$\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} \begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{vmatrix}. \tag{2.19}$$

We note that in order for (2.18) to be well defined, we must have $\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \neq 0$, which means that $\mathbf{a}, \mathbf{b}, \mathbf{c}$ must not lie on a straight line.

Because of their connection with barycentric combinations, barycentric coordinates are *affinely invariant*: let $\mathbf{p}$ have barycentric coordinates u, v, w with respect to $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Now map all four points to another set of four points by an affine map Φ. Then $\Phi\mathbf{p}$ has the same barycentric coordinates u, v, w with respect to $\Phi\mathbf{a}, \Phi\mathbf{b}, \Phi\mathbf{c}$.

Figure 2.8 illustrates more of the geometric properties of barycentric coordinates. An immediate consequence of Figure 2.8 is known as *Ceva's theorem:*

$$\text{ratio}(\mathbf{a}, \mathbf{p}_c, \mathbf{b}) \cdot \text{ratio}(\mathbf{b}, \mathbf{p}_a, \mathbf{c}) \cdot \text{ratio}(\mathbf{c}, \mathbf{p}_b, \mathbf{a}) = 1.$$

More details on this and related theorems can be found in most geometry books, e.g., Gans [224] or Berger [46], or Boehm and Prautzsch [76].

Any three noncollinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ define a barycentric coordinate system in the plane. The points inside the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ have positive barycentric coordinates, while the remaining ones have (some) negative barycentric coordinates. Figure 2.9 shows more.

We may use barycentric coordinates to define *bivariate linear interpolation*. Suppose we are given three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \mathbb{E}^3$. Then any point of the form

$$\mathbf{p} = \mathbf{p}(\mathbf{u}) = \mathbf{p}(u, v, w) = u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3 \tag{2.20}$$

with $u + v + w = 1$ lies in the plane spanned by $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. This map from $\mathbb{E}^2$ to $\mathbb{E}^3$ is called *linear interpolation*. Since $u + v + w = 1$, we may interpret u, v, w as barycentric coordinates of $\mathbf{p}$ relative to $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. We may also interpret u, v, w as barycentric coordinates of a point in $\mathbb{E}^2$ relative to some triangle $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{E}^2$. Then (2.20) may be interpreted as a map of the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{E}^2$ onto the triangle $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \mathbb{E}^3$. We call the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ the *domain triangle*. Note that the actual

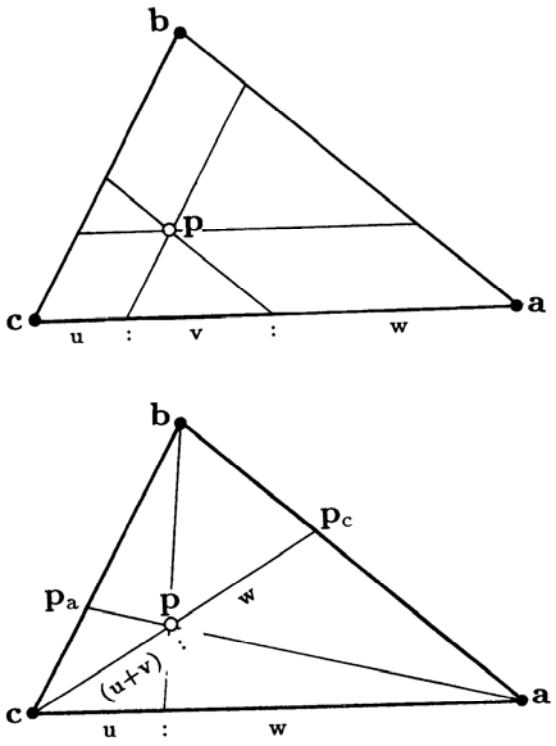

Figure 2.8: Barycentric coordinates: let $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$. The two figures show some of the ratios generated by certain straight lines through $\mathbf{p}$.

location or shape of the domain triangle is totally irrelevant to the definition of linear interpolation. (Of course, we must demand that it be nondegenerate.) Since we can interpret u, v, w as barycentric coordinates in both two and three dimensions, it follows that linear interpolation (2.20) is an affine map.

Barycentric coordinates are not restricted to one and two dimensions; they are defined for spaces of higher dimensions as well. For example, in three-space, any nondegenerate tetrahedron with vertices $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ may be used to write any point $\mathbf{p}$ as $\mathbf{p} = u_1\mathbf{p}_1 + u_2\mathbf{p}_2 + u_3\mathbf{p}_3 + u_4\mathbf{p}_4$.

2.7 Tessellations and Triangulations

When dealing with sequences of straight line segments, we were in the context of piecewise linear interpolation. We may also consider more than one triangle, thus introducing bivariate piecewise linear interpolation. While straight line segments are

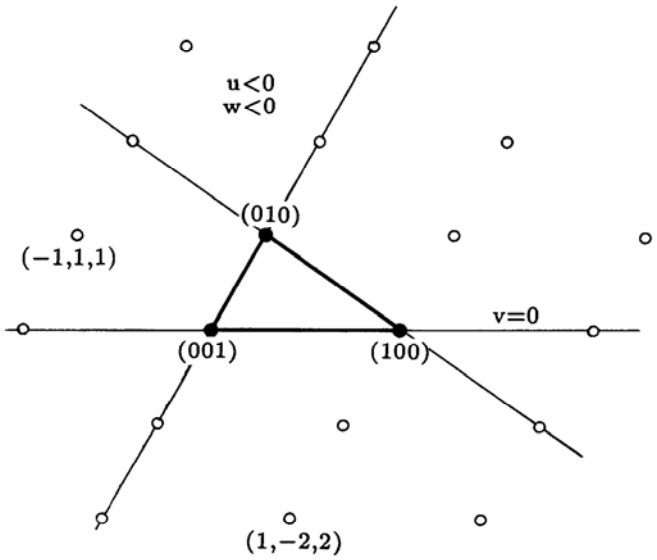

Figure 2.9: Barycentric coordinates: a triangle defines a coordinate system in the plane.

combined into polygons in a straightforward way, the corresponding concepts for triangles are not so obvious; they are the subject of this section.

We will first introduce the concept of a *Dirichlet tessellation*; this will lead to an efficient way to deal with triangles. So consider a collection of points $\mathbf{p}_i$ in the plane. We are going to construct influence regions around each point in the following way: Suppose each point is a transmitter for a cellular phone network. As a car moves through the points $\mathbf{p}_i$, its phone should always be using the closest transmitter. We may think of each transmitter as having an area of influence around it: whenever a car is in a given transmitter's area, its phone switches to that transmitter. More technically speaking, we associate with each point $\mathbf{p}_k$ a *tile* $\mathbf{T}_k$ consisting of all points $\mathbf{p}$ that are closer to $\mathbf{p}_k$ than to any other point $\mathbf{p}_i$. The collection of all these tiles is called the *Dirichlet tessellation* of the given point set.[5] Two points are called *neighbors* if their tiles share a common edge. See Figure 2.10.

It is intuitively clear that the tile edges should consist of segments taken from perpendicular bisectors of neighboring points. This observation directly leads to a recursive construction which is due to R. Sibson [477]: suppose that we already constructed the Dirichlet tessellation for a set of points, and we now want to add one more point $\mathbf{p}_L$. First, we determine which of the previously constructed tiles is

[5]This structure is also known as a *Voronoi diagram* or *Thiessen regions*.

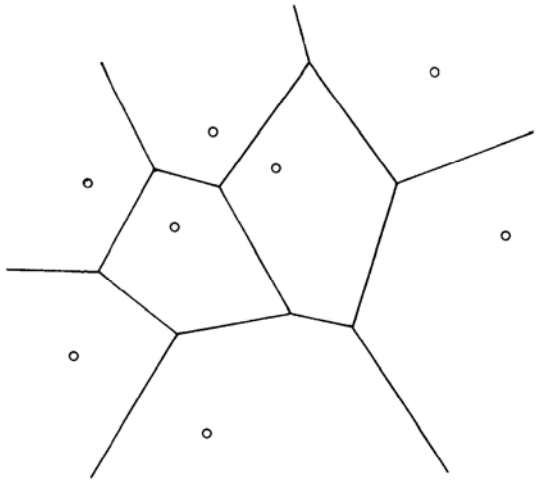

Figure 2.10: Dirichlet tessellations: a point set and its tile edges.

occupied by $\mathbf{p}_L$; referring to Figure 2.11, let us assume it is $\mathbf{T}_k$. We now draw all perpendicular bisectors between $\mathbf{p}_L$ and its neighbors, thus forming $\mathbf{T}_L$. Continuing in this manner, we can construct the tessellation for an arbitrary number of points. Each point is thus in the "center" of a tile, most of them finite, but some infinite. It is

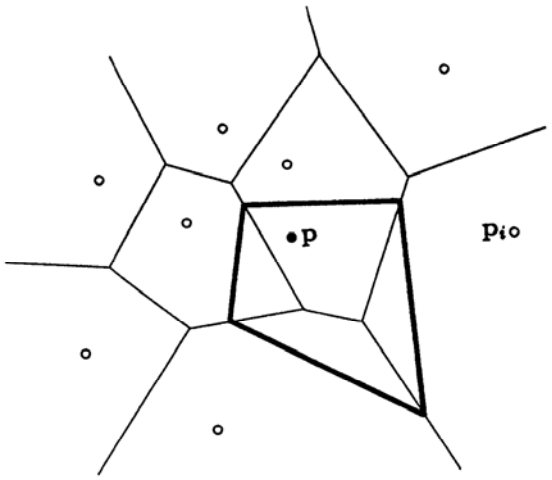

Figure 2.11: Dirichlet tessellations: a new point is inserted into an existing tessellation; its tile is shaded.

not hard to see that all points with infinite tiles determine the convex hull of the data points; see Section 2.1 for a definition.

While the preceding method may not be the most efficient one to construct the Dirichlet tessellation for a set of points, it is very intuitive, and also forms the basis of the following fundamental theorem. The tile $\mathbf{T}_L$ is formed by cutting out parts of $\mathbf{p}_L$'s neighboring tiles. Let $\mathcal{A}_i$ be the area cut of $\mathbf{T}_i$, and let $\mathcal{A}$ be the area of $\mathbf{T}_L$. Then we can write $\mathbf{p}_L$ as a barycentric combination of its neighbors (note that $\sum \mathcal{A}_i = \mathcal{A}$):

$$\mathbf{p}_L = \sum_i \frac{\mathcal{A}_i}{\mathcal{A}} \mathbf{p}_i. \tag{2.21}$$

This identity is also due to R. Sibson [477]; in case the summation is over only three neighbors, it reduces to the barycentric coordinates of Section 2.6.

The Dirichlet tessellation of a set of points determines another fundamental structure that is connected with the point set: its *Delaunay triangulation*. If we connect all neighboring points, we have created a set of triangles that cover the convex hull of the point set and that have the given points as their vertices; see Figure 2.12. The points with infinite tiles are now connected; they are called *boundary points* of the triangulation.

We should mention one problem: while the Dirichlet tessellation is unique, the Delaunay triangulation may not be. As an example, consider four points forming a square: either diagonal produces a valid Delaunay triangulation. Four points that have no unique Delaunay triangulation are called *neutral sets*; such points are always cocircular.

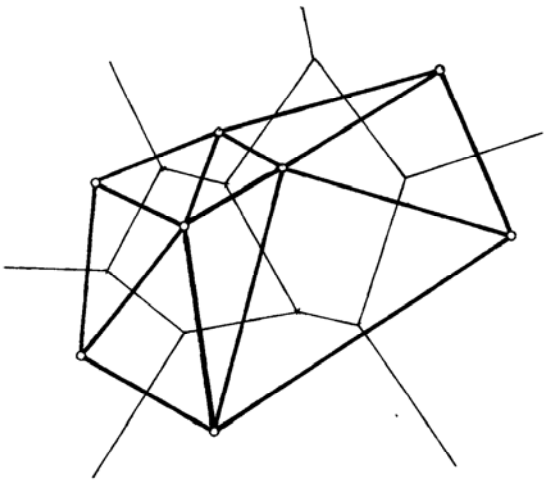

Figure 2.12: Delaunay triangulations: a point set with its Dirichlet tessellation (fine lines) and its Delaunay triangulation (heavy lines).

Clearly, there are many valid triangulations of a given point set. As it turns out, the Delaunay triangulation is one of the "nicer" ones. Intuitively, we might say that a triangulation is "nice" if it consists of triangles that are close to being equilateral. If we compare two different triangulations of a point set, we might then compute the minimal angle of each triangle. The triangulation that has the *largest* minimal angle would be labeled the better one. Of all possible triangulations, the Delaunay triangulation is the one that is guaranteed to produce the largest minimal angle; for a proof, see Lawson [324]. The Delaunay triangulation is thus said to satisfy the max–min criterion.

One might also consider the triangulation that satisfies the min–max criterion: the triangulation whose maximal angle is minimal. These triangulations are not easy to compute; one reason is that their neutral point sets are fairly complex (see Hansford [272]).

An important implementation aspect is the type of data structure to be used for triangulations. Data sets with several million points are not unheard-of, and for those, an intelligent structure is crucial. Such a structure should have the following elements:

1. A point collection of (x, y) coordinate pairs,
2. A collection of triangles, each pointing to three elements in the point list and also to three elements in the triangle collection, namely those that designate a triangle's three neighbors.[6]

These collections are best realized in the form of linked lists, for ease of inserting and deleting points. This data structure goes back to F. Little, who implemented it in 1978 at the University of Utah.

The major use of triangulations is in *piecewise linear interpolation*: suppose that at each data point $\mathbf{p}_k$ we are given a function value z_k. Then we may construct a linear interpolant—using linear interpolation from Section 2.6—over each of the triangles. We obtain a faceted, continuous surface that interpolates to all given data. This surface is not smooth, but it will give a decent idea of the shape of the given data. One application is in cartography: here, the given data points might be coordinates obtained from satellite readings, and the function values might be their elevations. Our piecewise linear surface is an approximation to the landscape being surveyed.

Once function values are involved, it may be advantageous to construct a triangulation that reflects this information. Such triangulations are called *data dependent*; see Dyn *et al.* [163] or Brown [82]. Here, one does not just consider triangles in the plane, but rather the three-dimensional triangles generated by the data points (x_k, y_k, z_k).

[6]Boundary triangles may have only one or two neighbors.

2.8 Function Spaces

This section contains material that will later simplify our work by allowing very concise notation. Although we shall try to develop our material with an emphasis on geometric concepts, it will sometimes simplify our work considerably if we can resort to some elementary topics from functional analysis. Good references are the books by Davis [122] and de Boor [126].

Let $C[a, b]$ be the set of all real-valued continuous functions defined over the interval $[a, b]$ of the real axis. We can define addition and multiplication by a constant for elements $f, g \in C[a, b]$ by setting $(\alpha f + \beta g)(t) = \alpha f(t) + \beta g(t)$ for all $t \in [a, b]$. With these definitions, we can easily show that $C[a, b]$ forms a *linear space* over the reals. The same is true for the sets $C^k[a, b]$, the sets of all real-valued functions defined over $[a, b]$ that are k-times continuously differentiable. Furthermore, for every k, C^{k+1} is a *subspace* of C^k.

We say that n functions $f_1, \ldots, f_n \in C[a, b]$ are *linearly independent* if $\sum c_i f_i = 0$ for all $t \in [a, b]$ implies $c_1 = \cdots = c_n = 0$.

We mention some subspaces of $C[a, b]$ that will be of interest later. The spaces $\mathcal{P}^n$ of all *polynomials* of degree n are:

$$p^n(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n; \quad t \in [a, b].$$

For fixed n, the dimension of $\mathcal{P}^n$ is $n + 1$: each $p^n \in \mathcal{P}^n$ is determined uniquely by the $n + 1$ coefficients $a_0, \ldots, a_n$. These can be interpreted as a vector in $(n + 1)$-dimensional linear space $\mathbb{R}^{n+1}$, which has dimension $n + 1$. We can also name a *basis* for $\mathcal{P}^n$: the *monomials* $1, t, t^2, \ldots, t^n$ are $n + 1$ linearly independent functions and thus form a basis.

Another interesting class of subspaces of $C[a, b]$ is given by piecewise linear functions: let $a = t_0 < t_1 < \cdots < t_n = b$ be a *partition* of the interval $[a, b]$. A continuous function that is linear on each subinterval $[t_i, t_{i+1}]$ is called a *piecewise linear function*. Over a fixed partition of $[a, b]$, the piecewise linear functions form a linear function space. A basis for this space is given by the *hat functions*: a hat function $H_i(t)$ is a piecewise linear function with $H_i(t_i) = 1$ and $H_i(t_j) = 0$ if $i \neq j$. A piecewise linear function f with $f(t_j) = f_j$ can always be written as

$$f(t) = \sum_{j=0}^{n} f_j H_j(t).$$

Figure 2.13 gives an example.

We will also consider *linear operators* that assign a function $\mathcal{A}f$ to a given function f. An operator $\mathcal{A} : C[a, b] \rightarrow C[a, b]$ is called *linear* if it leaves linear combinations invariant:

$$\mathcal{A}(\alpha f + \beta g) = \alpha \mathcal{A} f + \beta \mathcal{A} g; \quad \alpha, \beta \in \mathbb{R}.$$

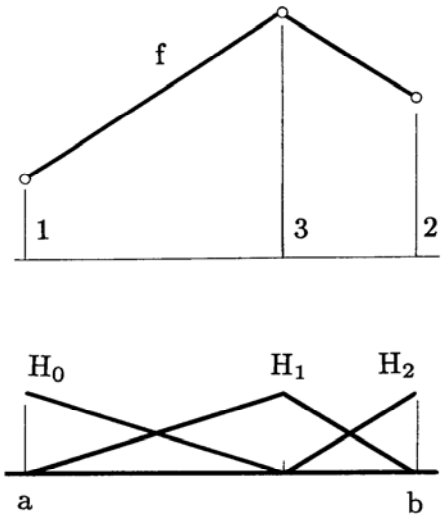

Figure 2.13: Hat functions: the piecewise linear function f can be written as $f = H_0 + 3H_1 + 2H_2$.

An example is given by the derivative operator that assigns the derivative f' to a given function f: $\mathcal{A}f = f'$.

2.9 Exercises

1. Of all affine maps, shears seem to be the least familiar to most people.[7] Construct a matrix that maps the unit square with points (0, 0), (1, 0), (1, 1), (0, 1) to the parallelogram with image points (0, 0), (1, 0), (2, 1), (1, 1).
2. In the definition of the variation diminishing property, we counted the crossings of a polygon with a plane. Discuss the case when the plane contains a whole polygon leg.

*3. We have seen that affine maps leave the ratio of three collinear points constant, i.e., they are ratio-preserving. Show that the converse is also true: every ratio-preserving map is affine.

*4. We defined the convex hull of a point set to be the set of all convex combinations formed by the elements of that set. Another definition is the following: the convex hull of a point set is the intersection of all convex sets that contain the given set. Show that the two definitions are equivalent.

[7] Recall that Figure 2.4 illustrates a shear.

*5. Show that the $n + 1$ functions $f_i(t) = t^i$; $i = 0, \ldots, n$ are linearly independent.

*6. Our definition of barycentric combinations gives the impression that it needs the involved points expressed in terms of some coordinate system. Show that this is not necessary: draw five points on a piece of paper, assign a weight to each one, and *construct* the barycenter of your points using a ruler (or compass and straightedge, if you are more classically inclined).

Remark: For this construction, it is not necessary for the weights to sum to one. This is so because the geometric construction remains the same if we multiplied all weights by a common factor. In fact, one may replace the concept of points (having mass one and requiring barycentric combinations as the basic point operation) by that of *mass points*, having arbitrary weights and yielding their barycenter (with the combined mass of all points) as the basic operation. In such a setting, vectors would also be mass points, but with mass zero.[8]

*7. Let a triangulation consist of b boundary points and of i interior points. Show that the number of triangles is $2i + b - 2$.

P1. Fix two distinct points **a**, **b** on the x-axis. Let a third point **x** trace out all of the x-axis. For each location of **x**, plot the value of the function ratio(**a**, **x**, **b**), thus obtaining a graph of the ratio function.

P2. Use the recursive algorithm from Section 2.7 to implement Dirichlet tessellations.

[8] I was introduced to this concept by A. Swimmer. It was developed by H. Grassmann in 1844.

Chapter 3

The de Casteljau Algorithm

The algorithm described in this chapter is probably the most fundamental one in the field of curve and surface design, yet it is surprisingly simple. Its main attraction is the beautiful interplay between geometry and algebra: a very intuitive geometric construction leads to a powerful theory.

Historically, it is with this algorithm that the work of de Casteljau started in 1959. The only written evidence is in [133] and [134], both of which are technical reports that are not easily accessible. De Casteljau's work went unnoticed until W. Boehm obtained copies of the reports in 1975. From then on, de Casteljau's name gained more popularity.

3.1 Parabolas

We give a simple construction for the generation of a parabola; the straightforward generalization will then lead to Bézier curves. Let $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ be any three points in $\mathbb{E}^3$, and let $t \in \mathbb{R}$. Construct

$$\begin{aligned}\mathbf{b}_0^1(t) &= (1-t)\mathbf{b}_0 + t\mathbf{b}_1,\\ \mathbf{b}_1^1(t) &= (1-t)\mathbf{b}_1 + t\mathbf{b}_2,\\ \\ \mathbf{b}_0^2(t) &= (1-t)\mathbf{b}_0^1(t) + t\mathbf{b}_1^1(t).\end{aligned}$$

Inserting the first two equations into the third one, we obtain

$$\mathbf{b}_0^2(t) = (1-t)^2\mathbf{b}_0 + 2t(1-t)\mathbf{b}_1 + t^2\mathbf{b}_2. \tag{3.1}$$

This is a quadratic expression in t (the superscript denotes the degree), and so $\mathbf{b}_0^2(t)$ traces out a *parabola* as t varies from $-\infty$ to $+\infty$. We denote this parabola by $\mathbf{b}^2$. This construction consists of *repeated linear interpolation*; its geometry is illustrated in

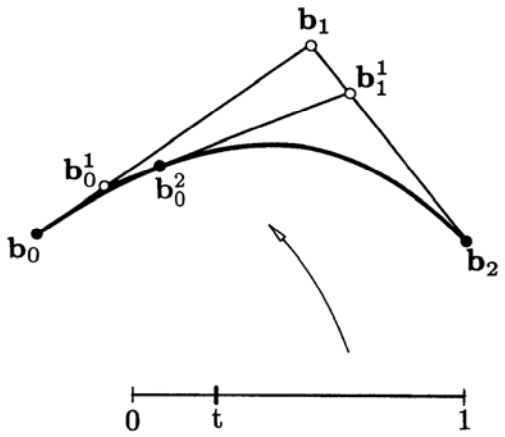

Figure 3.1: Parabolas: construction by repeated linear interpolation.

Figure 3.1. For t between 0 and 1, $\mathbf{b}^2(t)$ is inside the triangle formed by $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$; in particular, $\mathbf{b}^2(0) = \mathbf{b}_0$ and $\mathbf{b}^2(1) = \mathbf{b}_2$.

Inspecting the ratios of points in Figure 3.1, we see that

$$\text{ratio}(\mathbf{b}_0, \mathbf{b}_0^1, \mathbf{b}_1) = \text{ratio}(\mathbf{b}_1, \mathbf{b}_1^1, \mathbf{b}_2) = \text{ratio}(\mathbf{b}_0^1, \mathbf{b}_0^2, \mathbf{b}_1^1) = t/(1-t).$$

Thus our construction of a parabola is *affinely invariant* because piecewise linear interpolation is affinely invariant; see Section 2.4.

We also note that a parabola is a plane curve, because $\mathbf{b}^2(t)$ is always a barycentric combination of three points, as is clear from inspecting (3.1). A parabola is a special case of *conic sections*, which will be discussed in Chapter 13.

Finally we state a theorem from analytic geometry, closely related to our parabola construction. Let $\mathbf{a}, \mathbf{b}, \mathbf{c}$ be three distinct points on a parabola. Let the tangent at $\mathbf{b}$ intersect the tangents at $\mathbf{a}$ and $\mathbf{c}$ in $\mathbf{e}$ and $\mathbf{f}$, respectively. Let the tangents at $\mathbf{a}$ and $\mathbf{c}$ intersect in $\mathbf{d}$. Then $\text{ratio}(\mathbf{a}, \mathbf{e}, \mathbf{d}) = \text{ratio}(\mathbf{e}, \mathbf{b}, \mathbf{f}) = \text{ratio}(\mathbf{d}, \mathbf{f}, \mathbf{c})$. This *three tangent theorem* describes a property of parabolas; the de Casteljau algorithm can be viewed as the constructive counterpart.

3.2 The de Casteljau Algorithm

Parabolas are plane curves. However, many applications require true space curves.[1] For those purposes, the previous construction for a parabola can be generalized to generate a polynomial curve of arbitrary degree n:

[1] Compare the comments by P. Bézier in Chapter 1!

de Casteljau Algorithm

Given: $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{E}^3$ and $t \in \mathbb{R}$,

Set:

$$\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \qquad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases} \tag{3.2}$$

and $\mathbf{b}_i^0(t) = \mathbf{b}_i$. Then $\mathbf{b}_0^n(t)$ is the point with parameter value t on the *Bézier curve* $\mathbf{b}^n$.

The polygon $\mathbf{P}$ formed by $\mathbf{b}_0, \ldots, \mathbf{b}_n$ is called the *Bézier polygon* or *control polygon* of the curve $\mathbf{b}^n$.[2] Similarly, the polygon vertices $\mathbf{b}_i$ are called *control points* or *Bézier points*. Figure 3.2 illustrates the cubic case.

Sometimes we also write $\mathbf{b}^n(t) = \mathcal{B}[\mathbf{b}_0, \ldots, \mathbf{b}_n; t] = \mathcal{B}[\mathbf{P}; t]$ or, shorter, $\mathbf{b}^n = \mathcal{B}[\mathbf{b}_0, \ldots, \mathbf{b}_n] = \mathcal{B}\mathbf{P}$. This notation[3] defines $\mathcal{B}$ to be the (linear) operator that associates the Bézier curve with its control polygon. We say that the curve $\mathcal{B}[\mathbf{b}_0, \ldots, \mathbf{b}_n]$ is the *Bernstein–Bézier approximation* to the control polygon, a terminology borrowed from approximation theory; see also Section 5.10.

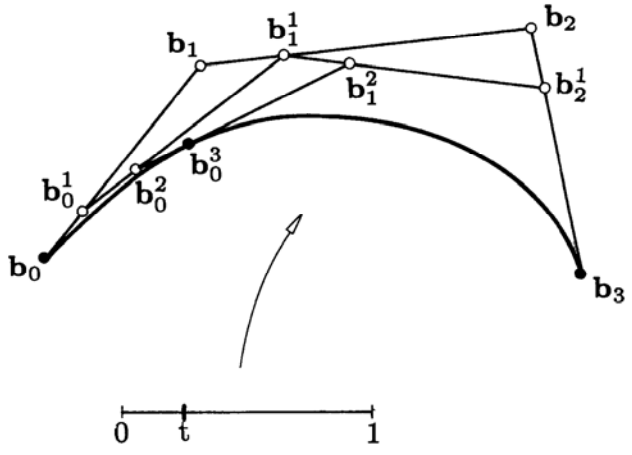

Figure 3.2: The de Casteljau algorithm: the point $\mathbf{b}_0^3(t)$ is obtained from repeated linear interpolation. The cubic case $n = 3$ is shown for $t = 1/4$.

[2] In the cubic case, there are four control points; they form a tetrahedron in the 3D case. This tetrahedron was already mentioned by W. Blaschke [59] in 1923; he called it "osculating tetrahedron."

[3] This notation should not be confused with the blossoming notation used later.

The intermediate coefficients $\mathbf{b}_i^r(t)$ are conveniently written into a triangular array of points, the *de Casteljau scheme*. We give the example of the cubic case:

$$\begin{array}{cccc} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1 & & \\ \mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\ \mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3. \end{array} \tag{3.3}$$

This triangular array of points seems to suggest the use of a two-dimensional array in writing code for the de Casteljau algorithm. That would be a waste of storage, however: it is sufficient to use the left column only and to overwrite it appropriately.

For a numerical example, see Example 3.1. Figure 3.3 shows 50 evaluations of a Bézier curve. The intermediate points $\mathbf{b}_i^r$ are also plotted, and connected.

A de Casteljau scheme for a planar cubic and for $t = \frac{1}{2}$:

$$\begin{array}{cccc} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & & & \\ \begin{bmatrix} 0 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & & \\ \begin{bmatrix} 8 \\ 2 \end{bmatrix} & \begin{bmatrix} 4 \\ 2 \end{bmatrix} & \begin{bmatrix} 2 \\ \frac{3}{2} \end{bmatrix} & \\ \begin{bmatrix} 4 \\ 0 \end{bmatrix} & \begin{bmatrix} 6 \\ 1 \end{bmatrix} & \begin{bmatrix} 5 \\ \frac{3}{2} \end{bmatrix} & \begin{bmatrix} \frac{7}{2} \\ \frac{3}{2} \end{bmatrix} \end{array}$$

Example 3.1: Computing a point on a Bézier curve with the Casteljau algorithm.

3.3 Some Properties of Bézier Curves

The de Casteljau algorithm allows us to infer several important properties of Bézier curves. We will infer these properties from the geometry underlying the algorithm. In the next chapter, we will show how they can also be derived analytically.

Affine invariance. Affine maps were discussed in Section 2.2. They are in the tool kit of every CAD system: objects must be repositioned, scaled, and so on. An important property of Bézier curves is that they are invariant under affine maps, which means that the following two procedures yield the same result: (1) first, compute the point $\mathbf{b}^n(t)$ and then apply an affine map to it; (2) first, apply an affine

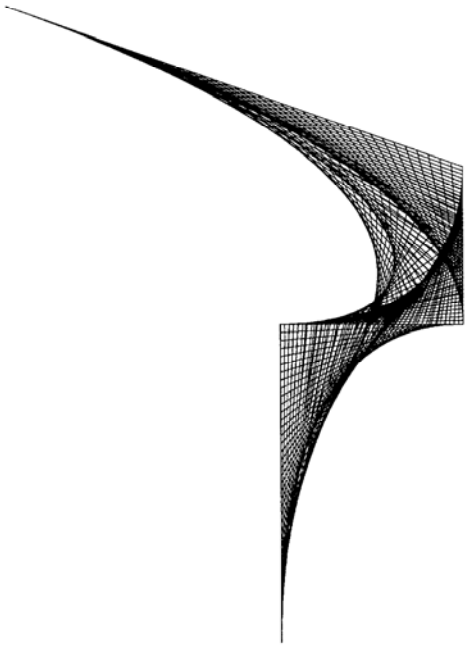

Figure 3.3: The de Casteljau algorithm: 50 points are computed on a quartic curve, and the intermediate points $\mathbf{b}_i^r$ are connected.

map to the control polygon and then evaluate the mapped polygon at parameter value t.

Affine invariance is, of course, a direct consequence of the de Casteljau algorithm: the algorithm is composed of a sequence of linear interpolations (or, equivalently, of a sequence of affine maps). These are themselves affinely invariant, and so is a finite sequence of them.

Let us discuss a practical aspect of affine invariance. Suppose we plot a cubic curve $\mathbf{b}^3$ by evaluating at 100 points and then plotting the resulting point array. Suppose now that we would like to plot the curve after a rotation has been applied to it. We can take the 100 computed points, apply the rotation to each of them, and plot. Or, we can apply the rotation to the 4 control points, then evaluate 100 times and plot. The first method needs 100 applications of the rotation, while the second needs only 4!

Affine invariance may not seem to be a very exceptional property for a useful curve scheme; in fact, it is not straightforward to think of a curve scheme that does not have it (exercise!). It is perhaps worth noting that Bézier curves do *not* enjoy another, also very important, property: they are not *projectively invariant*. Projective maps are used in computer graphics when an object is to be rendered realistically. So if we try to make life easy and simplify a perspective map of

a Bézier curve by mapping the control polygon and then computing the curve, we have actually cheated: that curve is not the perspective image of the original curve! More details on perspective maps can be found in Chapter 13.

Invariance under affine parameter transformations. Very often, one thinks of a Bézier curve as being defined over the interval [0, 1]. This is done because it is convenient, not because it is necessary: the de Casteljau algorithm is "blind" to the actual interval that the curve is defined over because it uses ratios only. One may therefore think of the curve as being defined over any arbitrary interval $a \leq u \leq b$ of the real line—after the introduction of local coordinates $t = (u - a)/(b - a)$, the algorithm proceeds as usual. This property is inherited from the linear interpolation process (2.10). The corresponding generalized de Casteljau algorithm is of the form:

$$\mathbf{b}_i^r(u) = \frac{b-u}{b-a}\mathbf{b}_i^{r-1}(u) + \frac{u-a}{b-a}\mathbf{b}_{i+1}^{r-1}(u). \tag{3.4}$$

The transition from the interval [0, 1] to the interval $[a, b]$ is an *affine map*. Therefore, we can say that Bézier curves are invariant under affine parameter transformations. Sometimes, one sees the term *linear parameter transformation* in this context, but this terminology is not quite correct: the transformation of the interval [0, 1] to $[a, b]$ typically includes a translation, which is not a linear map.

Convex hull property. For $t \in [0, 1]$, $\mathbf{b}^n(t)$ lies in the convex hull (see Figure 2.3) of the control polygon. This follows because every intermediate $\mathbf{b}_i^r$ is obtained as a convex barycentric combination of previous $\mathbf{b}_j^{r-1}$—at no step of the de Casteljau algorithm do we produce points outside the convex hull of the $\mathbf{b}_i$.

A simple consequence of the convex hull property is that a planar control polygon always generates a planar curve.

The importance of the convex hull property lies in what is known as *interference checking*. Suppose we want to know if two Bézier curves intersect each other—for example, each might represent the path of a robot arm, and our aim is to make sure that the two paths do not intersect, thus avoiding expensive collisions of the robots. Instead of actually computing a possible intersection, we can perform a much cheaper test: circumscribe the smallest possible box around the control polygon of each curve such that it has its edges parallel to some coordinate system. Such boxes are called *minmax boxes*, since their faces are created by the minimal and maximal coordinates of the control polygons. Clearly each box contains its control polygon, and, by the convex hull property, also the corresponding Bézier curve. If we can verify that the two boxes do not overlap (a trivial test), we are assured that the two curves do not intersect. If the boxes do overlap, we would have to perform more checks on the curves. The possibility for a quick decision of no interference is extremely important, since

in practice one often has to check one object against thousands of others, most of which can be labeled as "no interference" by the minmax box test.[4]

Endpoint interpolation. The Bézier curve passes through $\mathbf{b}_0$ and $\mathbf{b}_n$: we have $\mathbf{b}^n(0) = \mathbf{b}_0, \mathbf{b}^n(1) = \mathbf{b}_n$. This is easily verified by writing down the scheme (3.3) for the cases $t = 0$ and $t = 1$. In a design situation, the endpoints of a curve are certainly two very important points. It is therefore essential to have direct control over them, which is assured by endpoint interpolation.

Designing with Bézier curves. Figure 3.4 shows two Bézier curves. From the inspection of these examples, one gets the impression that in some sense the Bézier curve "mimics" the Bézier polygon—this statement will be made more precise later. It is why Bézier curves provide such a handy tool for the *design* of curves: To reproduce the shape of a hand-drawn curve, it is sufficient to specify a control polygon that somehow "exaggerates" the shape of the curve. One lets the computer draw the Bézier curve defined by the polygon, and, if necessary, adjusts the location (possibly also the number) of the polygon vertices. Typically, an experienced person will reproduce a given curve after two to three iterations of this *interactive* procedure.

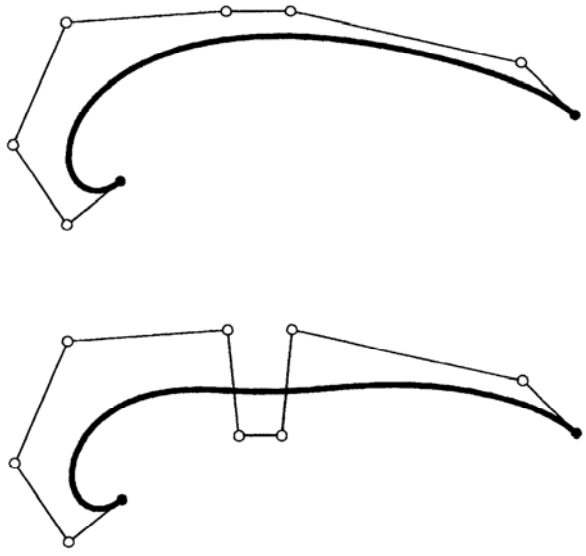

Figure 3.4: Bézier curves: some examples.

[4]It is possible to create volumes (or areas, in the 2D case) that hug the given curve closer than the minmax box does. See Sederberg *et al.* [463].

3.4 The Blossom

In recent years, a new way to look at Bézier curves has been developed; it is called the principle of *blossoming*. This principle was independently developed by de Casteljau [135] and Ramshaw [414], [416]. Other literature includes Seidel [464], [467], [468]; DeRose and Goldman [150]; Boehm [68]; and Lee [328].

We introduce blossoms as a generalization of the de Casteljau algorithm. Written in a scheme as in (3.3), we have to compute n columns. Our generalization is as follows: in column r, do not again perform a de Casteljau step for parameter value t, but use a new value t_r. Restricting ourselves to the cubic case, we obtain:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1[t_1] & & \\ \mathbf{b}_2 & \mathbf{b}_1^1[t_1] & \mathbf{b}_0^2[t_1, t_2] & \\ \mathbf{b}_3 & \mathbf{b}_2^1[t_1] & \mathbf{b}_1^2[t_1, t_2] & \mathbf{b}_0^3[t_1, t_2, t_3]. \end{array} \tag{3.5}$$

The resulting point $\mathbf{b}_0^3[t_1, t_2, t_3]$ is now a function of three independent variables; thus it no longer traces out a curve, but a region of $\mathbb{E}^3$. This trivariate function $\mathbf{b}[\cdot, \cdot, \cdot]$ is called the *blossom* of the curve $\mathbf{b}^3(t)$, after L. Ramshaw [414]. The original curve is recovered if we set all three arguments equal: $t = t_1 = t_2 = t_3$.

To understand the blossom better, we now evaluate it for several special arguments. We already know, of course, that $\mathbf{b}[0, 0, 0] = \mathbf{b}_0$ and $\mathbf{b}[1, 1, 1] = \mathbf{b}_3$. Let us start with $[t_1, t_2, t_3] = [0, 0, 1]$. The scheme (3.5) reduces to:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0 & & \\ \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\ \mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_1 = \mathbf{b}[0, 0, 1]. \end{array} \tag{3.6}$$

Similarly, we can show that $\mathbf{b}[0, 1, 1] = \mathbf{b}_2$. Thus the original Bézier points can be found by evaluating the curve's blossom at arguments consisting only of 0's and 1's.

But the remaining entries in (3.3) may also be written as values of the blossom for special arguments. For instance, setting $[t_1, t_2, t_3] = [0, 0, t]$, we have the scheme

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0 & & \\ \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\ \mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0^1 = \mathbf{b}[0, 0, t]. \end{array} \tag{3.7}$$

Continuing in the same manner, we may write the complete scheme (3.3) as:

$$\begin{array}{llll} \mathbf{b}_0 = \mathbf{b}[0, 0, 0] & & & \\ \mathbf{b}_1 = \mathbf{b}[0, 0, 1] & \mathbf{b}[0, 0, t] & & \\ \mathbf{b}_2 = \mathbf{b}[0, 1, 1] & \mathbf{b}[0, t, 1] & \mathbf{b}[0, t, t] & \\ \mathbf{b}_3 = \mathbf{b}[1, 1, 1] & \mathbf{b}[t, 1, 1] & \mathbf{b}[t, t, 1] & \mathbf{b}[t, t, t]. \end{array} \tag{3.8}$$

This is easily generalized to arbitrary degrees, where we can also express the Bézier points as blossom values:

$$\mathbf{b}_i = \mathbf{b}[0^{\langle n-i\rangle}, 1^{\langle i\rangle}], \tag{3.9}$$

where $t^{\langle r\rangle}$ means that t appears r times as an argument. For example, $\mathbf{b}[0^{\langle 1\rangle}, t^{\langle 2\rangle}, 1^{\langle 0\rangle}] = \mathbf{b}[0, t, t]$.

The de Casteljau recursion (3.2) can now be expressed in terms of the blossom $\mathbf{b}[]$:

$$\begin{aligned}\mathbf{b}[0^{\langle n-r-i\rangle}, t^{\langle r\rangle}, 1^{\langle i\rangle}] &= (1-t)\mathbf{b}[0^{\langle n-r-i+1\rangle}, t^{\langle r-1\rangle}, 1^{\langle i\rangle}] \\ &\quad + t\mathbf{b}[0^{\langle n-r-i\rangle}, t^{\langle r-1\rangle}, 1^{\langle i+1\rangle}].\end{aligned} \tag{3.10}$$

The point on the curve is given by $\mathbf{b}[t^{\langle n\rangle}]$.

We next note that it does not matter in which order we use the t_i for the blossom's evaluation. So we have, again for the cubic case, that $\mathbf{b}[t_1, t_2, t_3] = \mathbf{b}[t_2, t_3, t_1]$, etc. A proof of this statement is obtained using Figure 2.7: point $\mathbf{c}$ in that figure may be written as the value of a quadratic blossom: $\mathbf{c} = \mathbf{b}[t, s] = \mathbf{b}[s, t]$. The general result follows from this special instance.

Functions whose values do not depend on the order of their arguments are called *symmetric*; thus a blossom is a symmetric polynomial function of n variables. Every polynomial curve has a unique blossom associated with it—it is a symmetric polynomial of n variables, mapping $\mathbb{R}^n$ into $\mathbb{E}^3$.

The blossom has yet another important property. If the first argument of the blossom is a barycentric combination of two (or more) numbers, we may compute the blossom values for each argument and then form their barycentric combination:

$$\mathbf{b}[\alpha r + \beta s, t_2, \ldots, t_n] = \alpha\mathbf{b}[r, t_2, \ldots, t_n] + \beta\mathbf{b}[s, t_2, \ldots, t_n]; \quad \alpha + \beta = 1. \tag{3.11}$$

Equation (3.11) states that the blossom $\mathbf{b}$ is affine with respect to its first argument, but it is affine for any of the remaining arguments as well. This is the reason why the blossom is called *multiaffine*. Blossoms are multiaffine since they can be obtained by repeated steps of the de Casteljau algorithm. Each of these steps consists of linear interpolation, an affine map itself; see (2.5).

Knowing that the blossom is uniquely associated with the curve, we could have used (3.11) to *define* the de Casteljau algorithm: we just observe that $t = (1-t) * 0 + t * 1$, and now (3.11) yields (3.10).

We may also consider the blossom of a Bézier curve that is not defined over $[0, 1]$ but over the more general interval $[a, b]$. Proceeding exactly as above—but now utilizing (3.4)—we find that the Bézier points $\mathbf{b}_i$ are found as the blossom values

$$\mathbf{b}_i = \mathbf{b}[a^{\langle n-i\rangle}, b^{\langle i\rangle}]. \tag{3.12}$$

Thus a cubic over $u \in [a, b]$ has Bézier points $\mathbf{b}[a, a, a]$, $\mathbf{b}[a, a, b]$, $\mathbf{b}[a, b, b]$, $\mathbf{b}[b, b, b]$. If the original Bézier curve was defined over $[0, 1]$, the Bézier points of the one

corresponding to $[a, b]$ are simply found by four calls to a blossom routine! See also Figure 4.5.

3.5 Implementation

The header of the de Casteljau algorithm program is:

```
float decas(degree,coeff,t)
/*   uses  de Casteljau to compute one coordinate
     value of a  Bezier curve. Has to be called
     for each coordinate  (x,y, and/or z) of a control polygon.
Input:   degree:  degree of curve.
         coeff:   array with coefficients of curve.
         t:       parameter value.
Output: coordinate value.
*/
```

This procedure invites several comments. First, we see that it requires the use of an auxiliary array `coeffa`. Moreover, this auxiliary array has to be filled for each function call! So on top of the already high computational cost of the de Casteljau algorithm, we add another burden to the routine, keeping it from being very efficient. A faster evaluation method is given at the end of the next chapter.

To plot a Bézier curve, we would then call the routine several times:

```
void bez_to_points(degree,npoints,coeff,points)
/*     Converts Bezier curve into point sequence. Works on
       one coordinate only.
 Input:   degree:  degree of curve.
          npoints: # of coordinates to be generated. (counting
                   from 0!)
          coeff:   coordinates of control polygon.
Output:  points:   coordinates of points on curve.

     Remark: For a 2D curve, this routine needs to be called twice,
          once for the x-coordinates and once for y.
*/
```

The last subroutine has to be called once for each coordinate, i.e., two or three times. The main program `decasmain.c` on the enclosed disk gives an example of how to use it and how to generate postscript output.

3.6 Exercises

1. Suppose a planar Bézier curve has a control polygon that is symmetric with respect to the y-axis. Is the curve also symmetric with respect to the y-axis? Be sure to consider the control polygon $(-1, 0)$, $(0, 1)$, $(1, 1)$, $(0, 2)$, $(0, 1)$, $(-1, 1)$, $(0, 2)$, $(0, 1)$, $(1, 0)$. Generalize to other symmetry properties.

2. Use the de Casteljau algorithm to design a curve of degree four that has its middle control point on the curve. More specifically, try to achieve

$$\mathbf{b}_2 = \mathbf{b}_0^4\left(\frac{1}{2}\right).$$

Five collinear control points are a solution; try to be more ambitious!

*3. The de Casteljau algorithm may be formulated as

$$\mathcal{B}[\mathbf{b}_0, \ldots, \mathbf{b}_n; t] = (1-t)\mathcal{B}[\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}; t] + t\mathcal{B}[\mathbf{b}_1, \ldots, \mathbf{b}_n; t].$$

Show that the computation count is exponential (in terms of the degree) if you implement such a recursive algorithm in a language such as C.

*4. Show that every nonplanar cubic in $\mathbb{E}^3$ can be obtained as an affine map of the *standard cubic* (see Boehm [64]):

$$\mathbf{x}(t) = \begin{bmatrix} t \\ t^2 \\ t^3 \end{bmatrix}.$$

P1. Write an experimental program that replaces $(1-t)$ and t in the recursion (3.2) by $[1-f(t)]$ and $f(t)$, where f is some "interesting" function. Change the routine `decas` accordingly and comment on your results.

P2. Rewrite the routine `decas` to handle blossoms. Evaluate and plot for some "interesting" arguments.

P3. Experiment with the data set `outline_2D.dat` on the floppy: try to recapture its shape using one, two, and four Bézier curves. These curves should have decreasing degrees as you use more of them.

P4. Then repeat the previous problem with `outline_3D.dat`. This data set is three-dimensional, and you will have to use (at least) two views as you approximate the data points. The points, by the way, are taken from the outline of the sole of a high-heeled shoe.

Chapter 4

The Bernstein Form of a Bézier Curve

Bézier curves can be defined by a recursive algorithm, which is how de Casteljau first developed them. It is also necessary, however, to have an *explicit* representation for them; this will facilitate further theoretical development considerably.

4.1 Bernstein Polynomials

We will express Bézier curves in terms of *Bernstein polynomials*, defined explicitly by

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \tag{4.1}$$

where the binomial coefficients are given by

$$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{if } 0 \le i \le n \\ 0 & \text{else.} \end{cases}$$

There is a fair amount of literature on these polynomials. We cite just a few: Bernstein [47], Lorentz [340], Davis [122], and Korovkin [314]. An extensive bibliography is given in Gonska and Meier [234].

Before we explore the importance of Bernstein polynomials to Bézier curves, let us first examine them more closely. One of their important properties is that they satisfy the following recursion:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \tag{4.2}$$

with

$$B_0^0(t) \equiv 1 \tag{4.3}$$

and

$$B_j^n(t) \equiv 0 \quad \text{for } j \notin \{0, \ldots, n\}. \tag{4.4}$$

The proof is simple:

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

Another important property is that Bernstein polynomials form a *partition of unity*:

$$\sum_{j=0}^{n} B_j^n(t) \equiv 1. \tag{4.5}$$

This fact is proved with the help of the binomial theorem:

$$1 = [t + (1-t)]^n = \sum_{j=0}^{n} \binom{n}{j} t^j (1-t)^{n-j} = \sum_{j=0}^{n} B_j^n(t).$$

Figure 4.1 shows the family of the five quartic Bernstein polynomials. Note that the B_i^n are nonnegative over the interval $[0, 1]$.

We are now ready to see why Bernstein polynomials are important for the development of Bézier curves. The intermediate de Casteljau points $\mathbf{b}_i^r$ can be expressed in terms of Bernstein polynomials of degree r:

$$\mathbf{b}_i^r(t) = \sum_{j=0}^{r} \mathbf{b}_{i+j} B_j^r(t) \qquad \begin{array}{r} \in \{0, \ldots, n\} \\ i \in \{0, \ldots, n-r\}. \end{array} \tag{4.6}$$

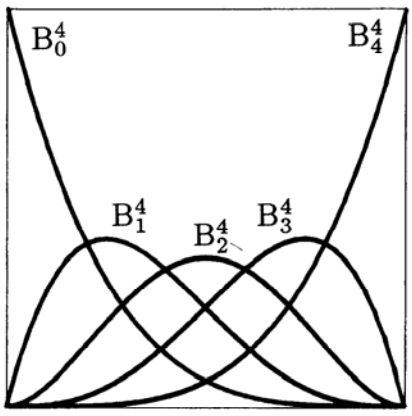

Figure 4.1: Bernstein polynomials: the quartic case.

This equation shows exactly how the intermediate point $\mathbf{b}_i^r$ depends on the given Bézier points $\mathbf{b}_i$. Figure 3.3 shows how these intermediate points form Bézier curves themselves.[1] The main importance of (4.6) lies, of course, in the case $r = n$. The corresponding de Casteljau point is the point on the curve and is given by

$$\mathbf{b}^n(t) = \mathbf{b}_0^n(t) = \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t). \tag{4.7}$$

We still have to prove (4.6). To that end, we use the recursive definition (3.2) of the $\mathbf{b}_i^r$ and the recursion for the Bernstein polynomials (4.2) and (4.4) in an inductive proof:

$$\begin{aligned}\mathbf{b}_i^r(t) &= (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \\ &= (1-t)\sum_{j=i}^{i+r-1} \mathbf{b}_j B_{j-i}^{r-1}(t) + t\sum_{j=i+1}^{i+r} \mathbf{b}_j B_{j-i-1}^{r-1}(t).\end{aligned}$$

Reindexing and invoking (4.4), we can rewrite this as

$$\begin{aligned}\mathbf{b}_i^r(t) &= (1-t)\sum_{j=i}^{i+r} \mathbf{b}_j B_{j-i}^{r-1}(t) + t\sum_{j=i}^{i+r} \mathbf{b}_j B_{j-i-1}^{r-1}(t) \\ &= \sum_{j=i}^{i+r} \mathbf{b}_j[(1-t)B_{j-i}^{r-1}(t) + tB_{j-i-1}^{r-1}(t)].\end{aligned}$$

Application of (4.2) then completes the proof. Note that (4.2) also defines B_0^n and B_n^n, since $B_{-1}^{n-1} = B_n^{n-1} = 0$ by (4.4).

With the intermediate points $\mathbf{b}_i^r$ at hand, we can write a Bézier curve in the form

$$\mathbf{b}^n(t) = \sum_{i=0}^{n-r} \mathbf{b}_i^r(t) B_i^{n-r}(t). \tag{4.8}$$

This is to be interpreted as follows: First, compute r levels of the de Casteljau algorithm with respect to t. Then, interpret the resulting points $\mathbf{b}_i^r(t)$ as control points of a Bézier curve of degree $n - r$ and evaluate it at t.

4.2 Properties of Bézier Curves

Many of the properties in this section have already appeared in the previous chapter. They were derived using geometric arguments. We shall now rederive several of

[1] We can also use Figure (3.2) to provide an example: the point $\mathbf{b}_1^2$ lies on the Bézier curve determined by $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$.

them, using algebraic arguments. If the same heading is used here as in Chapter 3, the reader should look there for a complete description of the property in question.

Affine invariance. Barycentric combinations are invariant under affine maps. Therefore, (4.5) gives the algebraic verification of this property. We note again that this does not imply invariance under perspective maps!

Invariance under affine parameter transformations. Algebraically, this property reads

$$\sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n \left(\frac{u-a}{b-a} \right). \tag{4.9}$$

Convex hull property. This follows, since for $t \in [0, 1]$, the Bernstein polynomials are nonnegative. They sum to one as shown in (4.5).

Endpoint interpolation. This is a consequence of the identities

$$\begin{aligned} B_i^n(0) &= \delta_{i,0} \\ B_i^n(1) &= \delta_{i,n} \end{aligned} \tag{4.10}$$

and (4.5). Here, $\delta_{i,j}$ is the Kronecker delta function: it equals one when its arguments agree, and zero otherwise.

Symmetry. Looking at the examples in Figure 3.4, it is clear that it does not matter if the Bézier points are labeled $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$ or $\mathbf{b}_n, \mathbf{b}_{n-1}, \ldots, \mathbf{b}_0$. The curves that correspond to the two different orderings look the same; they differ only in the direction in which they are traversed. Written as a formula:

$$\sum_{j=0}^{n} \mathbf{b}_j B_j^n(t) = \sum_{j=0}^{n} \mathbf{b}_{n-j} B_j^n(1-t). \tag{4.11}$$

This follows from the identity

$$B_j^n(t) = B_{n-j}^n(1-t), \tag{4.12}$$

which follows from inspection of (4.1). We say that Bernstein polynomials are *symmetric* with respect to t and $1 - t$.

Invariance under barycentric combinations. The process of forming the Bézier curve from the Bézier polygon leaves barycentric combinations invariant. For $\alpha + \beta = 1$, we obtain

$$\sum_{j=0}^{n} (\alpha \mathbf{b}_j + \beta \mathbf{c}_j) B_j^n(t) = \alpha \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t) + \beta \sum_{j=0}^{n} \mathbf{c}_j B_j^n(t). \tag{4.13}$$

In words: we can construct the weighted average of two Bézier curves either by taking the weighted average of corresponding points on the curves, or by taking the weighted average of corresponding control vertices and then computing the curve.

This linearity property is essential for many theoretical purposes, the most important one being the definition of tensor product surfaces in Chapter 15.

Linear precision. The following is a useful identity:

$$\sum_{j=0}^{n} \frac{j}{n} B_j^n(t) = t, \tag{4.14}$$

which has the following application: Suppose the polygon vertices $\mathbf{b}_j$ are uniformly distributed on a straight line joining two points $\mathbf{p}$ and $\mathbf{q}$:

$$\mathbf{b}_j = \left(1 - \frac{j}{n}\right)\mathbf{p} + \frac{j}{n}\mathbf{q}; \quad j = 0, \ldots, n.$$

The curve that is generated by this polygon is the straight line between $\mathbf{p}$ and $\mathbf{q}$, i.e., the initial straight line is reproduced. This property is called *linear precision*.[2]

Pseudo-local control. The Bernstein polynomial B_i^n has only one maximum and attains it at $t = i/n$. This has a design application: if we move only one of the control polygon vertices, say, $\mathbf{b}_i$, then the curve is mostly affected by this change in the region of the curve around the parameter value i/n. This makes the effect of the change reasonably predictable, although the change does affect the whole curve. As a rule of thumb (mentioned to me by P. Bézier), the maximum of each B_i^n is roughly $\frac{1}{3}$; thus a change of $\mathbf{b}_i$ by three units will change the curve by one unit.

4.3 The Derivative of a Bézier Curve

The derivative of a Bernstein polynomial B_i^n is obtained as

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

Thus

$$\frac{\mathrm{d}}{\mathrm{d}t} B_i^n(t) = n\left[B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right]. \tag{4.15}$$

[2] If the points are not uniformly spaced, we will also recapture the straight line segment. However, it will not be linearly parametrized.

We can now determine the derivative of a Bézier curve $\mathbf{b}^n$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\sum_{j=0}^{n}\left[B_{j-1}^{n-1}(t) - B_j^{n-1}(t)\right]\mathbf{b}_j.$$

Because of (4.4), this can be simplified to

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\sum_{j=1}^{n}B_{j-1}^{n-1}(t)\mathbf{b}_j - n\sum_{j=0}^{n-1}B_j^{n-1}(t)\mathbf{b}_j,$$

and now an index transformation of the first sum yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\sum_{j=0}^{n-1}B_j^{n-1}(t)\mathbf{b}_{j+1} - n\sum_{j=0}^{n-1}B_j^{n-1}(t)\mathbf{b}_j,$$

and finally

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\sum_{j=0}^{n-1}(\mathbf{b}_{j+1} - \mathbf{b}_j)B_j^{n-1}(t).$$

The last formula can be simplified somewhat by the introduction of the *forward difference operator* Δ:

$$\Delta\mathbf{b}_j = \mathbf{b}_{j+1} - \mathbf{b}_j. \tag{4.16}$$

We now have for the derivative of a Bézier curve:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\sum_{j=0}^{n-1}\Delta\mathbf{b}_j B_j^{n-1}(t); \quad \Delta\mathbf{b}_j \in \mathbb{R}^3. \tag{4.17}$$

The derivative of a Bézier curve is thus another Bézier curve, obtained by differencing the original control polygon. However, this derivative Bézier curve does not "live" in $\mathbb{E}^3$ any more! Its coefficients are differences of points, i.e., *vectors*, which are elements of $\mathbb{R}^3$. To visualize the derivative curve and polygon in $\mathbb{E}^3$, we can construct a polygon in $\mathbb{E}^3$ that consists of the points $\mathbf{a}+\Delta\mathbf{b}_0, \ldots, \mathbf{a}+\Delta\mathbf{b}_{n-1}$. Here $\mathbf{a}$ is arbitrary; one reasonable choice is $\mathbf{a} = \mathbf{0}$. Figure 4.2 illustrates a Bézier curve and its derivative curve (with the choice $\mathbf{a} = \mathbf{0}$). This derivative curve is sometimes called a *hodograph*. For more information on hodographs, see Forrest [212], Bézier [53], or Sederberg and Wang [462].

4.4 Higher Order Derivatives

To compute higher derivatives, we first generalize the forward difference operator (4.16): the *iterated forward difference operator* Δ^r is defined by

$$\Delta^r\mathbf{b}_j = \Delta^{r-1}\mathbf{b}_{j+1} - \Delta^{r-1}\mathbf{b}_j. \tag{4.18}$$

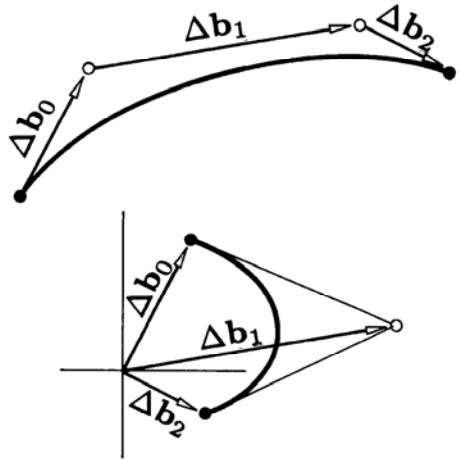

Figure 4.2: Derivatives: a Bézier curve and its first derivative curve (scaled down by a factor of three). Note that this derivative curve does not change if a translation is applied to the original curve.

We list a few examples:

$$\begin{aligned}
\Delta^0\mathbf{b}_i &= \mathbf{b}_i \\
\Delta^1\mathbf{b}_i &= \mathbf{b}_{i+1} - \mathbf{b}_i \\
\Delta^2\mathbf{b}_i &= \mathbf{b}_{i+2} - 2\mathbf{b}_{i+1} + \mathbf{b}_i \\
\Delta^3\mathbf{b}_i &= \mathbf{b}_{i+3} - 3\mathbf{b}_{i+2} + 3\mathbf{b}_{i+1} - \mathbf{b}_i.
\end{aligned}$$

The factors on the right-hand sides are binomial coefficients, forming a Pascal-like triangle. This pattern holds in general:

$$\Delta^r\mathbf{b}_i = \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j} \mathbf{b}_{i+j}. \tag{4.19}$$

We are now in a position to give the formula for the r^{th} derivative of a Bézier curve:

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(t) = \frac{n!}{(n-r)!} \sum_{j=0}^{n-r} \Delta^r\mathbf{b}_j B_j^{n-r}(t). \tag{4.20}$$

The proof of (4.20) is by repeated application of (4.17).

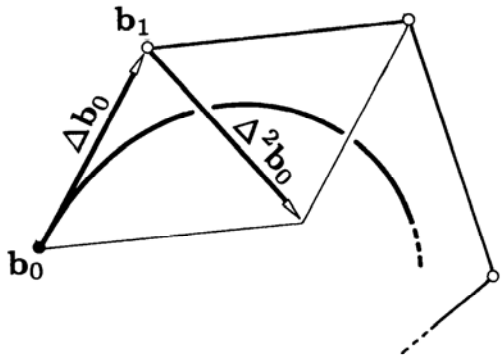

Figure 4.3: Endpoint derivatives: the first and second derivative vectors at $t = 0$ are multiples of the first and second difference vectors at $\mathbf{b}_0$.

Two important special cases of (4.20) are given by $t = 0$ and $t = 1$. Because of (4.10) we obtain

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(0) = \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_0 \tag{4.21}$$

and

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(1) = \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_{n-r}. \tag{4.22}$$

Thus the r^{th} derivative of a Bézier curve at an endpoint depends only on the $r+1$ Bézier points near (and including) that endpoint. For $r = 0$, we get the already established property of endpoint interpolation. The case $r = 1$ states that $\mathbf{b}_0$ and $\mathbf{b}_1$ define the tangent at $t = 0$, provided they are distinct.[3] Similarly, $\mathbf{b}_{n-1}$ and $\mathbf{b}_n$ determine the tangent at $t = 1$. The cases $r = 1$, $r = 2$ are illustrated in Figure 4.3.

If one knows all derivatives of a function at one point, corresponding to $t = 0$, say, one can generate its Taylor series. The Taylor series of a polynomial is just that polynomial itself, in the *monomial form*:

$$\mathbf{x}(t) = \sum_{j=0}^{n} \frac{1}{j!}\mathbf{x}^{(j)}(0)t^j.$$

Utilizing (4.21), we have

$$\mathbf{b}^n(t) = \sum_{j=0}^{n} \binom{n}{j} \Delta^j\mathbf{b}_0\, t^j. \tag{4.23}$$

The monomial form should be avoided wherever possible; it is very unstable for floating-point operations.

[3] In general, the tangent at $\mathbf{b}_0$ is determined by $\mathbf{b}_0$ and the first $\mathbf{b}_i$ that is distinct from $\mathbf{b}_0$. Thus the tangent may be defined even if the tangent vector is the zero vector.

4.5 Derivatives and the de Casteljau Algorithm

Derivatives of a Bézier curve can be expressed in terms of the intermediate points generated by the de Casteljau algorithm:

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(t) = \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_0^{n-r}(t). \tag{4.24}$$

This follows since summation and taking differences commute:

$$\sum_{j=0}^{n-1}\Delta\mathbf{b}_j = \sum_{j=1}^{n}\mathbf{b}_j - \sum_{j=0}^{n-1}\mathbf{b}_j = \Delta\sum_{j=0}^{n-1}\mathbf{b}_j. \tag{4.25}$$

Using this, we have

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(t) = \frac{n!}{(n-r)!}\sum_{j=0}^{n-r}\Delta^r\mathbf{b}_j B_j^{n-r}(t) \tag{4.26}$$

$$= \frac{n!}{(n-r)!}\Delta^r\sum_{j=0}^{n-r}\mathbf{b}_j B_j^{n-r}(t) \tag{4.27}$$

$$= \frac{n!}{(n-r)!}\Delta^r\mathbf{b}_0^{n-r}(t). \tag{4.28}$$

The first and the last of these three equations suggest two different ways of computing the r^{th} derivative of a Bézier curve: for the first method (4.26), compute all r^{th} forward differences of the control points, then interpret them as a new Bézier polygon of degree $n-r$ and evaluate it at t.

The second method, using (4.28), computes the r^{th} derivative as a "by-product" of the de Casteljau algorithm. If we compute a point on a Bézier curve using a triangular arrangement as in (3.3), then for any $n-r$, the corresponding $\mathbf{b}_i^{n-r}$ form a column (with $r+1$ entries) in that scheme. To obtain the r^{th} derivative at t, we simply take the r^{th} difference of these points and then multiply by the constant $n!/(n-r)!$. In some applications (curve/plane intersection, for example), one needs not only a point on the curve, but its first and/or second derivative at the same time. The de Casteljau algorithm offers a quick solution to this problem.

A summary of both methods: to compute the r^{th} derivative of a Bézier curve, perform r difference steps and $n-r$ evaluation steps. It does not matter in which order we perform these two steps.

The case $r = 1$ is important enough to warrant special attention:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n\big[\mathbf{b}_1^{n-1}(t) - \mathbf{b}_0^{n-1}(t)\big]. \tag{4.29}$$

The intermediate points $\mathbf{b}_0^{n-1}$ and $\mathbf{b}_1^{n-1}$ thus determine the *tangent vector* at $\mathbf{b}^n(t)$, which is illustrated in Figures 3.1 and 3.2.

Two ways to compute the tangent vector of a Bézier curve are demonstrated in Example 4.1.

To compute the derivative of the Bézier curve from Example 3.1, we could form the first differences of the control points and evaluate the corresponding quadratic curve at $t = \frac{1}{2}$:

$$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} 8 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} -4 \\ -2 \end{bmatrix} \quad \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Alternatively, we could compute the difference $\mathbf{b}_1^2 - \mathbf{b}_0^2$:

$$\begin{bmatrix} 5 \\ \frac{3}{2} \end{bmatrix} - \begin{bmatrix} 2 \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}.$$

In both cases, the result needs to be multiplied by a factor of 3.

Example 4.1: Two ways to compute derivatives.

4.6 Subdivision

A Bézier curve $\mathbf{b}^n$ is usually defined over the interval (the domain) $[0, 1]$, but it can also be defined over any interval $[0, c]$. The part of the curve that corresponds to $[0, c]$ can also be defined by a Bézier polygon, as illustrated in Figure 4.4. Finding this Bézier polygon is referred to as *subdivision* of the Bézier curve.

The unknown Bézier points $\mathbf{c}_i$ are found without much work if we use the blossoming principle from Section 3.4. There, (3.12) gave us the Bézier points of a polynomial curve that is defined over an arbitrary interval $[a, b]$. We are currently interested in the interval $[0, c]$, and so our Bézier points are:

$$\mathbf{c}_i = \mathbf{b}[0^{\langle n-i \rangle}, c^{\langle i \rangle}].$$

Thus each $\mathbf{c}_i$ is obtained by carrying out i de Casteljau steps with respect to c, in nonblossom notation:

$$\mathbf{c}_j = \mathbf{b}_0^j(c). \tag{4.30}$$

This formula is called the *subdivision formula* for Bézier curves.

Thus it turns out that the de Casteljau algorithm not only computes the point $\mathbf{b}^n(c)$, but also provides the control vertices of the Bézier curve corresponding to the interval $[0, c]$. Because of the symmetry property (4.11), it follows that the control vertices of the part corresponding to $[c, 1]$ are given by the $\mathbf{b}_j^{n-j}$. Thus, in Figures 3.1 and 3.2, we see the two subpolygons defining the arcs from $\mathbf{b}^n(0)$ to $\mathbf{b}^n(c)$ and from $\mathbf{b}^n(c)$ to $\mathbf{b}^n(1)$.

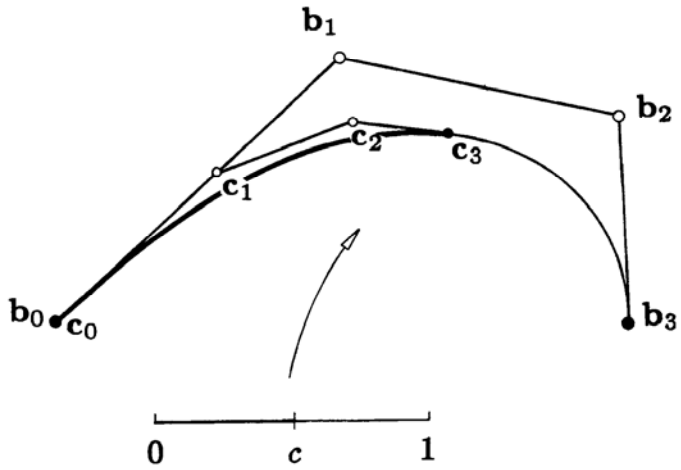

Figure 4.4: Subdivision: two Bézier polygons describing the same curve: one (the $\mathbf{b}_i$) is associated with the parameter interval [0, 1], the other (the $\mathbf{c}_i$) with $[0, c]$.

Figure 4.5 shows the blossom notation if we subdivide at *two* parameter values c and d simultaneously. This is a direct consequence of (3.12).

Instead of subdividing a Bézier curve, we may also *extrapolate* it: in that case, we might be interested in the Bézier points $\mathbf{d}_i$ corresponding to an interval $[1, d]$. They are given by

$$\mathbf{d}_j = \mathbf{b}[1^{\langle n-j \rangle}, d^{\langle j \rangle}] = \mathbf{b}^j_{n-j}(d).$$

It should be mentioned that extrapolation is not a numerically stable process and should be avoided for large values of d.

Subdivision for Bézier curves, although mentioned by de Casteljau [134], was rigorously proved by E. Staerk [478]. Our blossom development is due to Ramshaw [414] and de Casteljau [135].

Subdivision may be repeated: we may subdivide a curve at $t = 1/2$, then split the two resulting curves at $t = 1/2$ of their respective parameters, and so on. After k levels of subdivisions, we end up with 2^k Bézier polygons, each describing a small arc of the original curve. These polygons converge to the curve if we keep increasing k, as was shown by Lane and Riesenfeld [319]. We will prove a more general statement in Section 10.7.

Convergence of this repeated subdivision process is very fast (see Cohen and Schumaker [112] and Dahmen [120]), and thus it has many practical applications. We shall discuss here the process of intersecting a straight line with a Bézier curve: Suppose we are given a planar Bézier curve and we wish to find intersection points with a given straight line $\mathbf{L}$, if they exist.

If the curve and $\mathbf{L}$ are far apart, we would like to be able to flag such configurations as quickly as possible, and then abandon any further attempts to find intersection

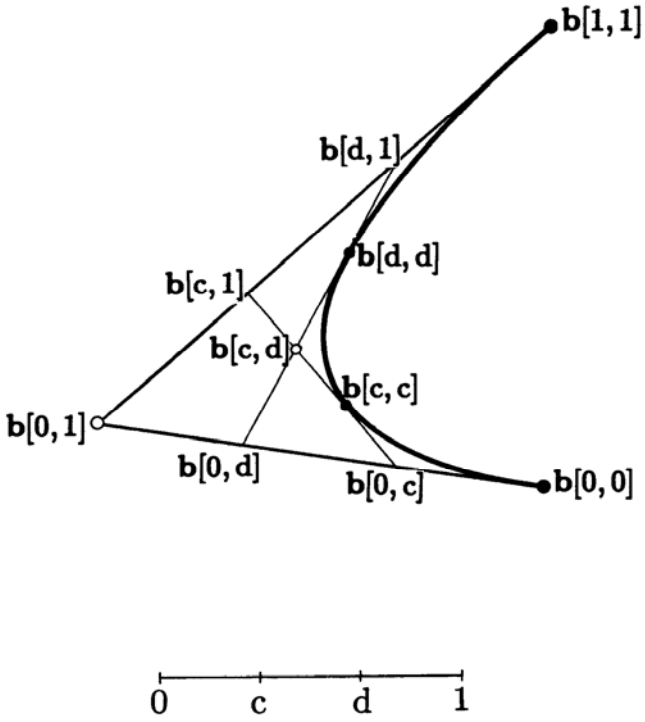

Figure 4.5: Generalized subdivision: evaluation of a quadratic at two parameter values c and d subdivides it into three segments. Its Bézier points are shown in blossom notation.

points. To do this, we create the *minmax box* of the control polygon: this is the smallest rectangle, with sides parallel to the coordinate axes, that contains the polygon. It is found very quickly, and by the convex hull property of Bézier curves, we know that it also contains the curve. Figure 4.6 gives an example.

Having found the minmax box, it is trivial to determine if it interferes with $\mathbf{L}$; if not, we know we will not have any intersections. This quick test is called *trivial reject*.

Now suppose the minmax box *does* interfere with $\mathbf{L}$. Then there may be an intersection. We now subdivide the curve at $t = 1/2$ and carry out our trivial reject test for both subpolygons.[4] If the outcome is still inconclusive, we repeat. Eventually the size of the involved minmax boxes will be so small that we can simply take their centers as the desired intersection points.

[4]The choice $t = 1/2$ is arbitrary, but works well. One might try to find better places to subdivide, but it is most likely cheaper to just perform a few more subdivisions instead.

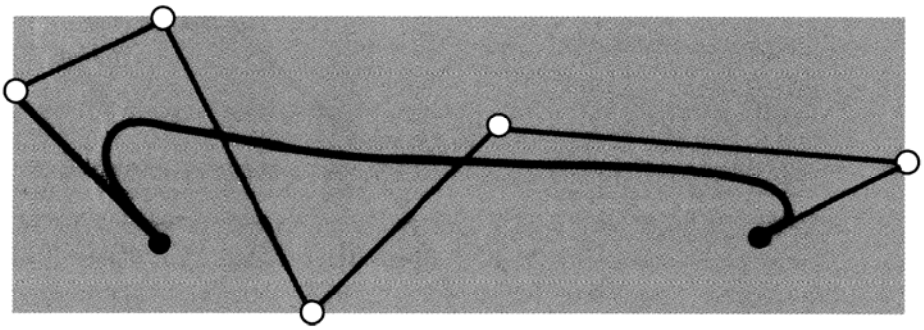

Figure 4.6: The minmax box of a Bézier curve: the smallest rectangle that contains the curve's control polygon.

The routine `intersect` employs this idea, and a little more: as we keep subdividing the curve, zooming in toward the intersection points, the generated subpolygons become simpler and simpler in shape. If the control points of a polygon are almost collinear, we may replace them by a straight line. We could then intersect this straight line with **L** in order to find an intersection point. The extra work here lies in determining if a control polygon is "linear" or not. In our case, this is done by the routine `checkflat`. Figure 4.7 gives two examples. Note how the subdivision process finds *all* intersection points in the bottom example. These points will not, however, be recorded by increasing values of t.

4.7 Blossom and Polar

After the first de Casteljau step with respect to a parameter value t_1, the resulting $\mathbf{b}_0^1(t_1), \ldots, \mathbf{b}_{n-1}^1(t_1)$ may be interpreted as a control polygon of a curve $\mathbf{p}_1(t)$ of degree $n-1$. In the blossoming terminology from Section 3.4, we can write:

$$\mathbf{p}_1(t) = \mathbf{b}[t_1, t^{\langle n-1 \rangle}].$$

Invoking our knowledge about derivatives, we have:

$$\begin{aligned}
\mathbf{p}_1(t) &= \sum_{i=0}^{n-1} \left[(1-t_1)\mathbf{b}_i + t_1\mathbf{b}_{i+1}\right] B_i^{n-1}(t) \\
&= \sum_{i=0}^{n-1} \left[(1-t_1)\mathbf{b}_i + t_1\mathbf{b}_{i+1} - \mathbf{b}_i^1(t)\right] B_i^{n-1}(t) + \sum_{i=0}^{n-1} \mathbf{b}_i^1(t) B_i^{n-1}(t) \\
&= (t_1 - t) \sum_{i=0}^{n-1} [\mathbf{b}_{i+1} - \mathbf{b}_i] B_i^{n-1}(t) + \sum_{i=0}^{n-1} \mathbf{b}_i^1(t) B_i^{n-1}(t).
\end{aligned}$$

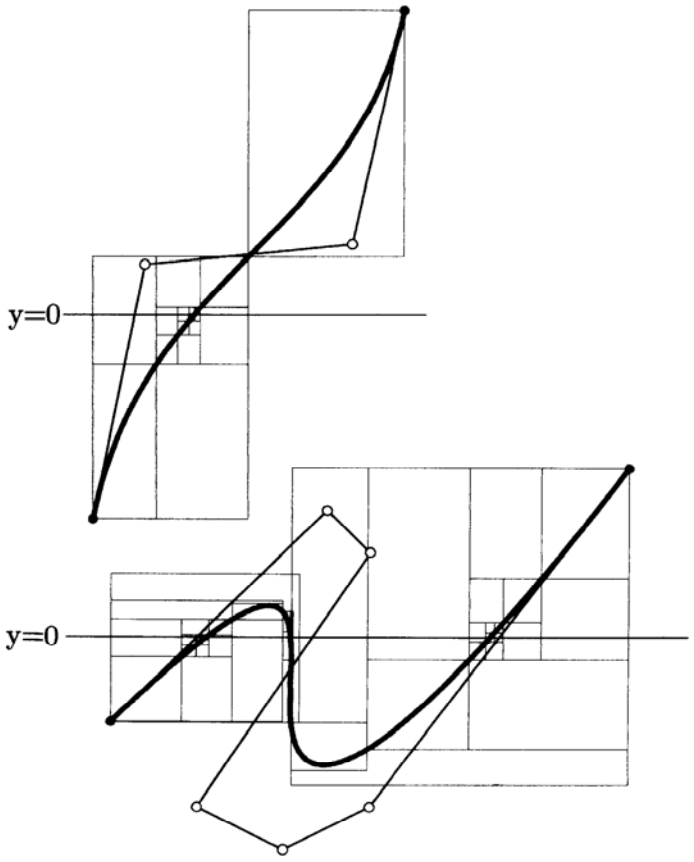

Figure 4.7: Curve intersection by subdivision: two examples are shown. Intersection is with the x-axis in both cases. Note the clustering of minmax boxes near the intersection points.

Therefore,

$$\mathbf{p}_1(t) = \mathbf{b}(t) + \frac{t_1 - t}{n} \frac{\mathrm{d}}{\mathrm{d}t} \mathbf{b}(t). \tag{4.31}$$

The polynomial $\mathbf{p}_1$ is called *first polar* of $\mathbf{b}(t)$ with respect to t_1. Figure 4.8 illustrates the geometric significance of (4.31): the tangent at any point $\mathbf{b}(t)$ intersects the polar $\mathbf{p}_1(t)$ at $\mathbf{p}_1(t)$. Keep in mind that this is not restricted to planar curves, but is equally valid for space curves!

For the special case of a (nonplanar) cubic, we may then conclude the following: the polar $\mathbf{p}_1$ lies in the osculating plane (see Section 11.2) of the cubic at $\mathbf{b}(t_1)$. If we intersect all tangents to the cubic with this osculating plane, we will trace out the

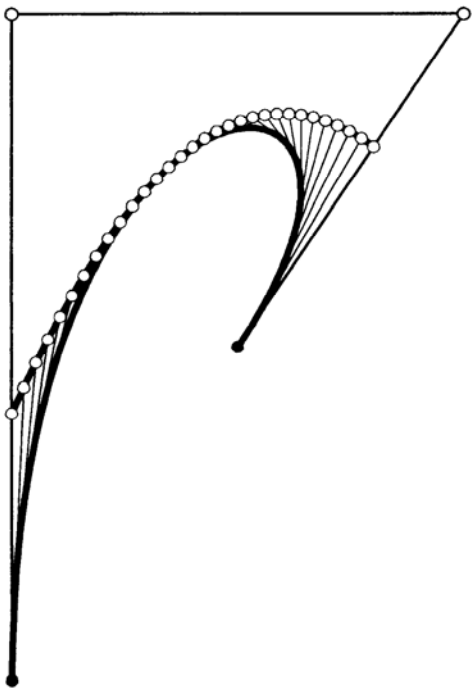

Figure 4.8: Polars: the polar $\mathbf{p}_1(t)$ with respect to $t_1 = 0.4$ is intersected by the tangents of the given curve $\mathbf{b}(t)$.

polar. We can also conclude that for three different parameters t_1, t_2, t_3, the blossom value $\mathbf{b}[t_1, t_2, t_3]$ is the intersection of the corresponding osculating planes.

Another special case is given by $\mathbf{b}[0, t^{\langle n-1\rangle}]$: this is the polynomial defined by $\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}$. Similarly, $\mathbf{b}[1, t^{\langle n-1\rangle}]$ is defined by $\mathbf{b}_1, \ldots, \mathbf{b}_n$. This observation may be used for a proof of (3.9).

Returning to the general case, we may repeat the process of forming polars, thus obtaining a second polar $\mathbf{p}_{1,2}(t) = \mathbf{b}[t_1, t_2, t^{\langle n-2\rangle}]$, etc. We finally arrive at the n^{th} polar, which we have already encountered as the blossom $\mathbf{b}[t_1, \ldots, t_n]$ of $\mathbf{b}(t)$. The relationship between blossoms and polars was observed by Ramshaw in [416]. The above geometric arguments are due to S. Jolles, who developed a geometric theory of blossoming as early as 1886 in [299].[5]

Section 3.4 provided a way to generate the blossom of a curve recursively. We may also find explicit formulas for it; here is the case of a cubic:

[5]W. Boehm first noted the relevance of Jolles's work to the theory of blossoming.

$$
\begin{aligned}
&\mathbf{b}[t_1, t_2, t_3] \\
&= (1 - t_1)\mathbf{b}[0, t_2, t_3] + t_1\mathbf{b}[1, t_2, t_3] \\
&= (1 - t_1)\big[(1 - t_2)\mathbf{b}[0, 0, t_3] + t_2\mathbf{b}[0, 1, t_3]\big] + t_1\big[(1 - t_2)\mathbf{b}[0, 1, t_3] \\
&\quad + t_2\mathbf{b}[1, 1, t_3]\big] = \mathbf{b}[0, 0, 0](1 - t_1)(1 - t_2)(1 - t_3) \\
&\quad + \mathbf{b}[0, 0, 1]\big[(1 - t_1)(1 - t_2)t_3 + (1 - t_1)t_2(1 - t_3) + t_1(1 - t_2)(1 - t_3)\big] \\
&\quad + \mathbf{b}[0, 1, 1]\big[t_1t_2(1 - t_3) + t_1(1 - t_2)t_3 + (1 - t_1)t_2t_3\big] \\
&\quad + \mathbf{b}[1, 1, 1]t_1t_2t_3.
\end{aligned}
$$

For each step, we have exploited the fact that blossoms are multiaffine.

Note how we recover the cubic Bernstein polynomials for $t_1 = t_2 = t_3$. The preceding development would hold for parameter intervals other than $[0, 1]$ equally well, because of the invariance under affine parameter transformations.

We should add that not every multivariate polynomial function can be interpreted as the blossom of a Bézier curve. To qualify as a blossom, the function must be both symmetric and multiaffine.

4.8 The Matrix Form of a Bézier Curve

Some authors (Faux and Pratt [199], Mortenson [364], Chang [96]) prefer to write Bézier curves and other polynomial curves in matrix form. A curve of the form

$$\mathbf{x}(t) = \sum_{j=0}^{n} \mathbf{c}_i C_i(t)$$

can be interpreted as a dot product:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{c}_0 & \dots & \mathbf{c}_n \end{bmatrix} \begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix}.$$

One can take this a step further and write

$$\begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} = \begin{bmatrix} m_{00} & \dots & m_{0n} \\ \vdots & & \vdots \\ m_{n0} & \dots & m_{nn} \end{bmatrix} \begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix}. \tag{4.32}$$

The matrix $M = \{m_{ij}\}$ describes the basis transformation between the basis polynomials $C_i(t)$ and the *monomial basis* t^i.

If the C_i are Bernstein polynomials, $C_i = B_i^n$, the matrix M has elements

$$m_{ij} = (-1)^{j-i}\binom{n}{j}\binom{j}{i}, \tag{4.33}$$

a simple consequence of (4.23).

We list the cubic case explicitly:

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

The matrix form (4.32) does not describe an actual Bézier curve; it is rather the monomial form, which is *numerically unstable* and should be avoided where accuracy in computation is of any importance. See the discussion in Section 24.3 for more details.

4.9 Implementation

First, we provide a routine that evaluates a Bézier curve more efficiently than `decas` from the last chapter. It will have the flavor of Horner's scheme for the evaluation of a polynomial in monomial form. To give an example of Horner's scheme, also called *nested multiplication*, we list the cubic case:

$$\mathbf{c}_0 + t\mathbf{c}_1 + t^2\mathbf{c}_2 + t^3\mathbf{c}_3 = \mathbf{c}_0 + t[\mathbf{c}_1 + t(\mathbf{c}_2 + t\mathbf{c}_3)].$$

A similar nested form can be devised for Bézier curves; again, the cubic case:

$$\mathbf{b}^3(t) = \left\{\left[\binom{3}{0}s\mathbf{b}_0 + \binom{3}{1}t\mathbf{b}_1\right]s + \binom{3}{2}t^2\mathbf{b}_2\right\}s + \binom{3}{3}t^3\mathbf{b}_3,$$

where $s = 1 - t$. Recalling the identity

$$\binom{n}{i} = \frac{n-i+1}{i}\binom{n}{i-1}; \quad i > 0,$$

we arrive at the following program (for the general case):

```
float hornbez(degree,coeff,t)
/*   uses  a Horner-like  scheme to compute one coordinate
     value of a  Bezier curve. Has to be called
     for each coordinate  (x,y, and/or z) of a control polygon.
Input:   degree: degree of curve.
         coeff:  array with coefficients of curve.
         t:      parameter value.
Output:          coordinate value.
*/
```

To use this routine for plotting a Bézier curve, we would replace the call to `decas` in `bez_to_points` by an identical call to `hornbez`. Replacing `decas` with `hornbez` results in a significant savings of time: we do not have to save the control polygon in an auxiliary array; also, `hornbez` is of order n, whereas `decas` is of order n^2.

This is not to say, however, that we have produced super-efficient code for plotting points on a Bézier curve. For instance, we have to call `hornbez` once for each coordinate, and thus have to generate the binomial coefficients `n_choose_i` twice. This could be improved by writing a routine that combines the two calls. A further improvement could be to compute the sequence of binomial coefficients only once, and not over and over for each new value of t. All these (and possibly more) improvements would speed up the program, but would be less modular and thus less understandable. For the code in this book, modularity is placed above efficiency (in most cases).

We also include the programs to convert from the Bézier form to the monomial form:

```
void bezier_to_power(degree,bez,coeff)
/*Converts Bezier form to power (monomial) form. Works on
one coordinate only.

  Input:   degree:    degree of curve.
           bez:       coefficients of Bezier form
  Output:  coeff:     coefficients of power form.

Remark: For a 2D curve, this routine needs to be called twice,
once for the x-coordinates and once for y.
*/
```

The conversion program internally calls iterated forward differences:

```
void differences(degree,coeff,diffs)
/*
Computes all forward differences Delta^i(b_0).
Has to be called for each coordinate  (x,y, and/or z) of a control polygon.
    Input:   degree: length (from 0) of coeff.
             coeff:  array of coefficients.
     Output: diffs:  diffs[i]= Delta^i(coeff[0]).
*/
```

Once the power form is found, it may be evaluated using Horner's scheme:

```
float horner(degree,coeff,t)
/*
     uses  Horner's scheme to compute one coordinate
     value of a curve in power form. Has to be called
     for each coordinate  (x,y, and/or z) of a control polygon.
```

```
     Input:   degree: degree of curve.
              coeff:  array with coefficients of curve.
              t:      parameter value.
      Output: coordinate value.
*/
```

The subdivision routine:

```
void subdiv(degree,coeff,weight,t,bleft,bright,wleft,wright)
/*
       subdivides ratbez curve at parameter value t.
  Input:  degree:    degree of Bezier curve
          coeff:     Bezier points (one coordinate only)
          weight:    weights for rational case
          t:         where to subdivide
  Output:
          bleft,bright: left and right subpolygons
          wleft,wright: their weights


  Note:   1. For the polynomial case, set all entries in weight to 1.
          2. Ordering of right polygon bright is reversed.
*/
```

Actually, this routine computes a more general case than is described in this chapter; namely, it computes subdivison for a *rational* Bézier curve. This will be discussed later; if the entries in `weight` are all unity, then `wleft` and `wright` will also be unity and can be safely ignored in the context of this chapter.

Now the routine to intersect a Bézier curve with a straight line (the straight line is assumed to be the y-axis):

```
void intersect(bx,by,w,degree,tol)
/* Intersects Bezier curve with x-axis  by  adaptive subdivision.
   Subdivision is controlled by tolerance tol. There is
   no check for stack depth! Intersection points are not found  in
  'natural' order.  Results are written into file outfile.
 Input: bx,by,w:     rational Bezier curve
        degree:      its degree
        tol:         accuracy for results
 Output:             intersection points, written into a file
 */
```

This routine (again covering the rational case as well) uses a routine to check if a control polygon is flat:

```
int check_flat(bx,by,degree,tol)
/* Checks if a polygon is flat. If all points
```

```
   are closer  than tol to the connection of the
   two endpoints, then it is flat. Crashes if the endpoints
   are identical.

  Input:    bx,by, degree: the Bezier curve
            tol:           tolerance
  Output:   1 if flat, 0 else.
*/
```

4.10 Exercises

1. Consider the cubic Bézier curve given by the planar control points

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

   At $t = 1/2$, this curve has a *cusp*: its first derivative vanishes and it shows a sharp corner. You should verify this by a sketch. Now perturb the x-coordinates of $\mathbf{b}_1$ and $\mathbf{b}_2$ by opposite amounts, thus maintaining a symmetric control polygon. Discuss what happens to the curve.

2. Show that a nonplanar cubic Bézier curve cannot have a cusp. Hint: use the fact that $\mathbf{b}_0^{n-1}$, $\mathbf{b}_1^{n-1}$, $\mathbf{b}_0^n$ are identical when we evaluate at the cusp.

3. Show that the Bernstein polynomial B_i^n attains its maximum at $t = i/n$. Find the maximum value. What happens for large n?

*4. Show that the Bernstein polynomials B_i^n form a basis for the linear space of all polynomials of degree n.

P1. Compare the run times of `decas` and `hornbez` for curves of various degrees.

P2. Use subdivision to create *smooth fractals*. Start with a degree four Bézier curve. Subdivide it into two curves and then perturb the middle control point $\mathbf{b}_2$ for each of the two subpolygons. Continue for several levels. Try to perturb the middle control point by a random displacement and then by a controlled displacement. Literature on fractals: [30], [346].

P3. Use subdivision to approximate a high-order ($n > 2$) Bézier curve by a collection of quadratic Bézier curves. You will have to write a routine that determines if a given Bézier curve may be replaced by a quadratic one within a given tolerance. Literature on approximating higher order curves by lower order ones: [290], [294].

Chapter 5

Bézier Curve Topics

5.1 Degree Elevation

Suppose we were designing with Bézier curves as described in Section 3.3, trying to use a Bézier curve of degree n. After we modify the polygon a few times, it may turn out that a degree n curve does not possess sufficient flexibility to model the desired shape. One way to proceed in such a situation is to increase the flexibility of the polygon by adding another vertex to it. As a first step, one might want to add another vertex, yet leave the shape of the curve unchanged—this corresponds to raising the degree of the Bézier curve by one. We are thus looking for a curve with control vertices $\mathbf{b}_0^{(1)}, \ldots, \mathbf{b}_{n+1}^{(1)}$ that describes the same curve as the original polygon $\mathbf{b}_0, \ldots, \mathbf{b}_n$.

Using the identities (5.32) to (5.34)—each easy to prove—we rewrite our given curve as $\mathbf{x}(t) = (1-t)\mathbf{x}(t) + t\mathbf{x}(t)$, or

$$\mathbf{x}(t) = \sum_{i=0}^{n} \frac{n+1-i}{n+1} \mathbf{b}_i B_i^{n+1}(t) + \sum_{i=0}^{n} \frac{i+1}{n+1} \mathbf{b}_i B_{i+1}^{n+1}(t).$$

The upper limit of the first sum may be extended to $n+1$ since the corresponding term is zero. The summation of the second sum may be shifted to the limits 1 and $n+1$, and then changed to the lower limit 0 since only a zero term is added. We thus have

$$\mathbf{x}(t) = \sum_{i=0}^{n+1} \frac{n+1-i}{n+1} \mathbf{b}_i B_i^{n+1}(t) + \sum_{i=0}^{n+1} \frac{i}{n+1} \mathbf{b}_{i-1} B_i^{n+1}(t).$$

Combining both sums and comparing coefficients yields the desired result:

$$\mathbf{b}_i^{(1)} = \frac{i}{n+1} \mathbf{b}_{i-1} + \left(1 - \frac{i}{n+1}\right) \mathbf{b}_i; \quad i = 0, \ldots, n+1. \tag{5.1}$$

Thus the new vertices $\mathbf{b}_i^{(1)}$ are obtained from the old polygon by piecewise linear interpolation at the parameter values $i/(n+1)$. It follows that the new polygon $\mathcal{E}\mathbf{P}$

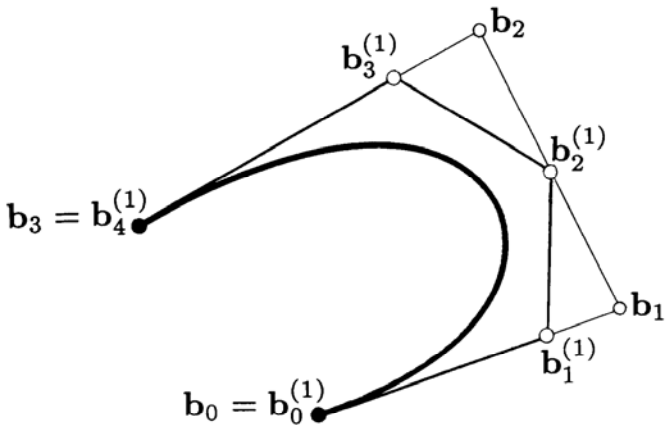

Figure 5.1: Degree elevation: both polygons define the same (degree three) curve.

lies in the convex hull of the old one. Figure 5.1 gives an example. Note how $\mathcal{E}\mathbf{P}$ is "closer" to the curve $\mathcal{B}\mathbf{P}$ than the original polygon $\mathbf{P}$.

While our proof is based on straightforward algebraic manipulations, a more elegant proof is provided through the use of blossoms. If we had the blossom $\mathbf{b}^{(1)}[t_1, \ldots, t_{n+1}]$ of the degree-elevated curve, then we could compute its control polygon using (3.9). After some experimentation (try the case $n = 2$!), it is easy to see that the blossom is given by

$$\mathbf{b}^{(1)}[t_1, \ldots, t_{n+1}] = \frac{1}{n+1} \sum_{j=0}^{n+1} \mathbf{b}[t_1, \ldots, t_{n+1} | t_j]. \tag{5.2}$$

Here, the notation $\mathbf{b}[t_1, \ldots, t_{n+1} | t_j]$ indicates that the argument t_j is omitted from $\mathbf{b}[t_1, \ldots, t_{n+1}]$. The control points are now given by application of (3.9):

$$\mathbf{b}_i^{n+1} = \mathbf{b}^{(1)}[0^{\langle n+1-i \rangle}, 1^{\langle i \rangle}].$$

Inspection of all terms that now arise in (5.2) reveals that the point $\mathbf{b}_{i-1}$ appears i times and that the point $\mathbf{b}_i$ appears $n + 1 - i$ times, thus re-proving our previous result.[1]

Degree elevation has important applications in surface design: for several algorithms that produce surfaces from curve input, it is necessary that these curves be of the same degree. Using degree elevation, we may achieve this by raising the degree of all input curves to the one of the highest degree. Another application lies in the area of *data transfer* between different CAD/CAM or graphics systems: Suppose you have generated a parabola (i.e., a degree two Bézier curve), and you want to feed it into a system that only knows about cubics. All you have to do is degree elevate your parabola.

[1] Again, work out the example $n = 2$ to build your confidence in this technique!

5.2 Repeated Degree Elevation

The process of degree elevation assigns a polygon $\mathcal{E}\mathbf{P}$ to an original polygon $\mathbf{P}$. We may repeat this process and obtain a sequence of polygons $\mathbf{P}$, $\mathcal{E}\mathbf{P}$, $\mathcal{E}^2\mathbf{P}$, etc. After r degree elevations, the polygon $\mathcal{E}^r\mathbf{P}$ has the vertices $\mathbf{b}_0^{(r)}, \ldots, \mathbf{b}_{n+r}^{(r)}$, and each $\mathbf{b}_i^{(r)}$ is explicitly given by

$$\mathbf{b}_i^{(r)} = \sum_{j=0}^{n} \mathbf{b}_j \binom{n}{j} \frac{\binom{r}{i-j}}{\binom{n+r}{i}}. \tag{5.3}$$

This formula is easily proved by induction.

Let us now investigate what happens if we repeat the process of degree elevation again and again. As we shall see, the polygons $\mathcal{E}^r\mathbf{P}$ converge to the curve that all of them define:

$$\lim_{r\to\infty} \mathcal{E}^r\mathbf{P} = \mathcal{B}\mathbf{P}. \tag{5.4}$$

To prove this result, fix some parameter value t. For each r, find the index i such that $i/(n+r)$ is closest to t. We can think of $i/(n+r)$ as a parameter on the polygon $\mathcal{E}^r\mathbf{P}$, and as $r \to \infty$, this ratio tends to t. One can now show (using Stirling's formula) that

$$\lim_{i/(n+r)\to t} \frac{\binom{r}{i-j}}{\binom{r+n}{i}} = t^j(1-t)^{n-j}, \tag{5.5}$$

and therefore

$$\lim_{i/(n+r)\to t} \mathbf{b}_i^{(r)} = \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t) = [\mathcal{B}\mathbf{P}](t).$$

Equation (5.5) will look familiar to readers with a background in probability: it states that the hypergeometric distribution converges to the binomial distribution.

Figure 5.2 shows an example of the limit behavior of the polygons $\mathcal{E}^r\mathbf{P}$.

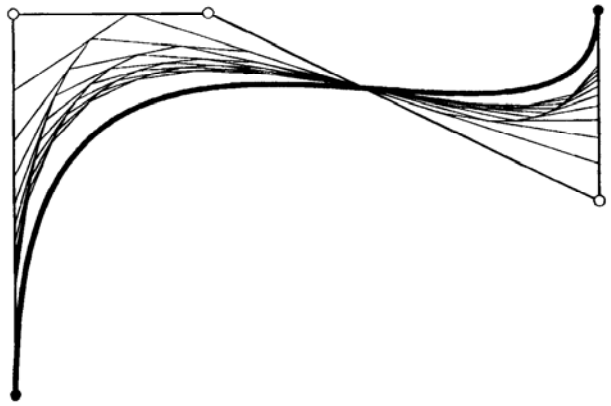

Figure 5.2: Degree elevation: a sequence of polygons approaching the curve that is defined by each of them.

The polygons $\mathcal{E}^r\mathbf{P}$ approach the curve very slowly; thus our convergence result has no practical consequences. However, it helps in the investigation of some theoretical properties, as is seen in the next section.

The convergence of the polygons $\mathcal{E}^r\mathbf{P}$ to the curve was conjectured by R. Forrest [212] and proved in Farin [168]. The above proof follows an approach taken by J. Zhou [509]. Degree elevation may be generalized to "corner-cutting"; for a brief description, see Section 10.7.

5.3 The Variation Diminishing Property

We can now show that Bézier curves enjoy the *variation diminishing property*:[2] the curve $\mathcal{B}\mathbf{P}$ has no more intersections with any plane other than the polygon $\mathbf{P}$. Degree elevation is an instance of piecewise linear interpolation, and we know that operation is variation diminishing (see Section 2.4). Thus each $\mathcal{E}^r\mathbf{P}$ has fewer intersections with a given plane than has its predecessor $\mathcal{E}^{(r-1)}\mathbf{P}$. Since the curve is the limit of these polygons, we have proved our statement. For high-degree Bézier curves, variation diminution may become so strong that the control polygon no longer resembles the curve.

A special case is obtained for *convex* polygons: a planar polygon (or curve) is said to be convex if it has no more than two intersections with any plane. The variation diminishing property thus asserts that a convex polygon generates a convex curve. Note that the inverse statement is not true: convex curves exist that have a nonconvex control polygon!

While the variation diminishing property seems straightforward enough, it is still not totally intuitive. Consider the following statement: two Bézier curves with common endpoints do not intersect more often than their control polygons. This appears to be true just after one jots down a few examples. Yet it is false, as shown by Prautzsch [411].

5.4 Degree Reduction

Degree elevation can be viewed as a process that introduces redundancy: a curve is described by more information than is actually necessary. The inverse process might seem more interesting: can we *reduce* possible redundancy in a curve representation? More specifically, can we write a given curve of degree n as one of degree $n-1$? We shall call this process *degree reduction*.

In general, exact degree reduction is not possible. For example, a cubic with a point of inflection cannot possibly be written as a quadratic. Degree reduction, therefore, can be viewed only as a method to *approximate* a given curve by one of

[2]The variation diminishing property was first investigated by I. Schoenberg [450] in the context of B-spline approximation.

lower degree. Our problem can now be stated as follows: given a Bézier curve with control vertices $\mathbf{b}_i; i = 0, \ldots, n$, can we find a Bézier curve with control vertices $\hat{\mathbf{b}}_i; i = 0, \ldots, n-1$ that approximates the first curve in a "reasonable" way?

Let us now pretend that the $\mathbf{b}_i$ were obtained from the $\hat{\mathbf{b}}_i$ by the process of degree elevation (this is not true, in general, but makes a good working assumption). Then they would be related by

$$\mathbf{b}_i = \frac{i}{n}\hat{\mathbf{b}}_{i-1} + \frac{n-i}{n}\hat{\mathbf{b}}_i; \quad i = 0, 1, \ldots, n. \tag{5.6}$$

This equation can be used to derive two recursive formulas for the generation of the $\hat{\mathbf{b}}_i$ from the $\mathbf{b}_i$:

$$\overrightarrow{\mathbf{b}}_i = \frac{n\mathbf{b}_i - i\overrightarrow{\mathbf{b}}_{i-1}}{n-i}; \quad i = 0, 1, \ldots, n-1 \tag{5.7}$$

and

$$\overleftarrow{\mathbf{b}}_{i-1} = \frac{n\mathbf{b}_i - (n-i)\overleftarrow{\mathbf{b}}_i}{i}; \quad i = n, n-1, \ldots, 1. \tag{5.8}$$

The $\overrightarrow{\mathbf{b}}_i$ are obtained by "unraveling" (5.1) left to right, while the $\overleftarrow{\mathbf{b}}_i$ are obtained in a right-to-left manner. Note that two undefined terms appear: they are $\overrightarrow{\mathbf{b}}_{-1}$ and $\overleftarrow{\mathbf{b}}_n$. Both are multiplied by zero, so no harm is done.

Figure 5.3 illustrates these two recursive formulas: a cubic polygon is given, and two quadratic approximations are obtained.

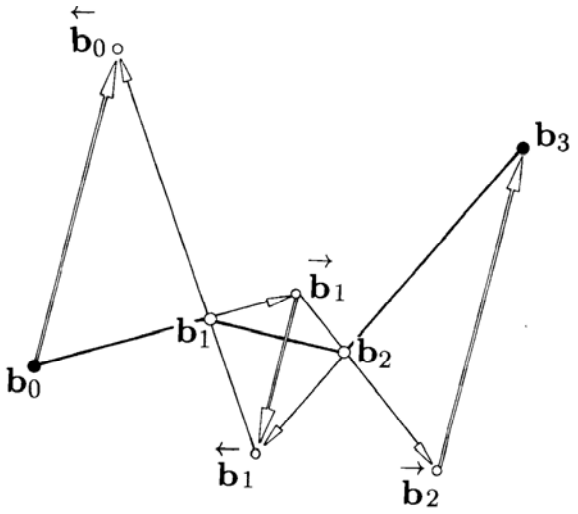

Figure 5.3: Degree reduction: a cubic is approximated by a "right-to-left" and by a "left-to-right" quadratic. Both approximations are very poor.

If the given curve had actually been of degree $n-1$ (i.e., if it had been the result of a degree elevation), than both the $\vec{\mathbf{b}}_i$ and the $\overleftarrow{\mathbf{b}}_i$ would produce that original curve of degree $n-1$. Since in general this is not true, we only obtain approximations—quite bad ones in most cases. The reason is that both (5.7) and (5.8) are *extrapolation* formulas, which are numerically unstable.

Figure 5.3 suggests that all vectors $\vec{\mathbf{b}}_i - \overleftarrow{\mathbf{b}}_i$ are parallel, an observation that was first made by J. Braun [80]. For a proof, we simply observe that the coplanar triangles $\overleftarrow{\mathbf{b}}_{i-1}$, $\vec{\mathbf{b}}_{i-1}$, $\mathbf{b}_i$ and $\mathbf{b}_i$, $\vec{\mathbf{b}}_i$, $\overleftarrow{\mathbf{b}}_i$ are similar.[3] Let us determine one of these vectors, namely $\vec{\mathbf{b}}_{n-1} - \mathbf{b}_n$. To that end, we use an explicit formula for the $\vec{\mathbf{b}}_i$, given in [165] or [168]:

$$\vec{\mathbf{b}}_i = \frac{1}{\binom{n-1}{i}} \sum_{j=0}^{i} (-1)^{i+j} \binom{n}{j} \mathbf{b}_j.$$

We deduce

$$\vec{\mathbf{b}}_{n-1} - \mathbf{b}_n = \Delta^n \mathbf{b}_0 = \frac{\mathrm{d}^n}{\mathrm{d}t^n} \mathbf{x}(t). \tag{5.9}$$

Following the same reasoning, all vectors $\vec{\mathbf{b}}_i - \overleftarrow{\mathbf{b}}_i$ are parallel to the n^{th} derivative vector of the given degree n curve.

One observes that (5.7) tends to produce reasonable approximations near $\mathbf{b}_0$ and that (5.8) behaves decently near $\mathbf{b}_n$. We may take advantage of this and combine both approximations, thus arriving at

$$\hat{\mathbf{b}}_i = (1-\lambda_i)\, \vec{\mathbf{b}}_i + \lambda_i\, \overleftarrow{\mathbf{b}}_i; \quad i = 0, \ldots, n-1. \tag{5.10}$$

We may set $\lambda_i = i/n$ (not so great; see Farin [174]) or $\lambda_i = 0$ for $i < n/2$ and $\lambda_i = 1$ for $i > n/2$ (decent; see Forrest [212]).

A better, and in some sense optimal way was first described by Watkins and Worsey [497] and also by Eck [165]. This optimal solution has its roots in approximation theory and uses the theory of *Chebychev polynomials*.[4] For more information on these polynomials, consult [101] or [122].

Chebychev polynomials T_i of degree i are defined recursively:

$$\begin{aligned} T_{i+1}(t) &= 2tT_i(t) - T_{i-1}(t); \\ T_0(t) &= 1, \\ T_1(t) &= t. \end{aligned}$$

In an approximation theory setting, these polynomials are typically defined over the interval $[-1, 1]$; over the interval $[0, 1]$, we would use the scaled version $T_i(2t-1)$.

[3] Verify in Figure 5.3 for $i = 1$!

[4] Incidentally, while the French automotive companies Citroën and Rénault used Bernstein polynomials for their CAD/CAM systems, the American company Chrysler used Chebychev polynomials in their first system.

Each Chebychev polynomial T_i has the unique property of achieving $i + 1$ extreme values in the interval $[-1, 1]$, alternating between the values $+1$ and -1.[5]

Chebychev polynomials form a basis for all polynomials of degree n, and so every polynomial $\mathbf{p}^n(t)$ has a unique representation

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

What is interesting in our context is the following: if we truncate the leading term $\mathbf{t}_n T_n(t)$ from the above sum, then we have found the—unique—polynomial $\mathbf{p}^{n-1}$ of degree $n - 1$ that deviates from the given one by the least possible amount. More precisely: we have that $\max_{-1 \le t \le 1} ||\mathbf{p}^n(t) - \mathbf{p}^{n-1}(t)||$ is smaller for this $\mathbf{p}^{n-1}$ that for any other $n - 1$ degree polynomial. This process is known as *Chebychev economization*. We could thus transform our Bézier curve of degree n into its Chebychev form, truncate the leading term, and transform back to the Bézier form of degree $n - 1$. This is what Watkins and Worsey [497] did. Eck showed that one can equivalently set

$$\lambda_i = \frac{1}{2^{2n-1}} \sum_{j=0}^{i} \binom{2n}{2j} \tag{5.11}$$

in (5.10). The maximum deviation between the original and the degree reduced curves is given by $||\Delta^n \mathbf{b}_0|| 2^{-(2n-1)}$ (see Figure 5.4).

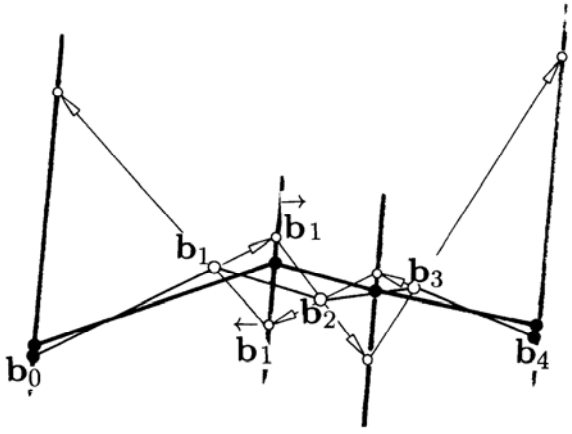

Figure 5.4: Degree reduction: combining the two degree reduction building blocks (solving from left to right and from right to left), together with the concept of Chebychev economization, yields a reasonable approximation.

[5]Compare this to Bernstein polynomials: they only have *one* extreme value in the interval [0, 1]!

A drawback of (5.11) is that it does not guarantee that $\hat{\mathbf{b}}_0 = \mathbf{b}_0$ and $\hat{\mathbf{b}}_{n-1} = \mathbf{b}_n$. The simplest (if not optimal) solution to this dilemma is to simply enforce these two conditions.

5.5 Nonparametric Curves

We have so far considered three-dimensional parametric curves $\mathbf{b}(t)$. Now we shall restrict ourselves to *functional curves* of the form $y = f(x)$, where f denotes a polynomial. These (planar) curves can be written in parametric form:

$$\mathbf{b}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ f(t) \end{bmatrix}.$$

We are interested in functions f that are expressed in terms of the Bernstein basis:

$$f(t) = b_0 B_0^n(t) + \cdots + b_n B_n^n(t).$$

Note that now the coefficients b_j are real numbers, not points. The b_j therefore do not form a polygon, yet functional curves are a subset of parametric curves and therefore must possess a control polygon. To find it, we recall the linear precision property of Bézier curves, as defined by (4.14). We can now write our functional curve as

$$\mathbf{b}(t) = \sum_{j=0}^{n} \begin{bmatrix} j/n \\ b_j \end{bmatrix} B_j^n(t). \tag{5.12}$$

Thus the control polygon of the function $f(t) = \sum b_j B_j^n$ is given by the points $(j/n, b_j)$; $j = 0, \ldots, n$. If we want to distinguish clearly between the parametric and the nonparametric cases, we call $f(t)$ a *Bézier function*. Figure 5.5 illustrates the cubic case. We also emphasize that the b_i are real numbers, not points; we call the b_i *Bézier ordinates*.

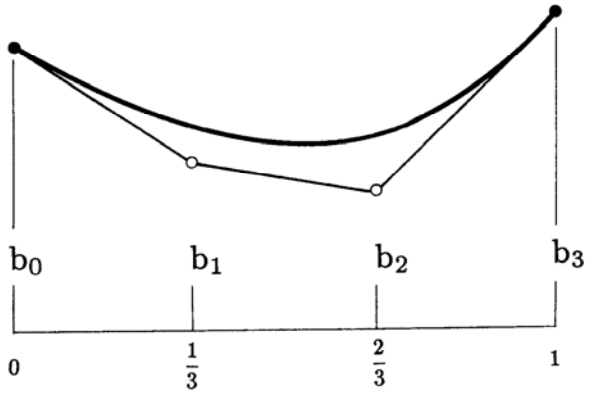

Figure 5.5: Functional curves: the control polygon of a cubic polynomial has abscissa values of 0, $\frac{1}{3}$, $\frac{2}{3}$, 1.

Because Bézier curves are invariant under affine reparametrizations, we may consider any interval $[a, b]$ instead of the special interval $[0, 1]$. Then the abscissa values are $a + i(b - a)/n;\ \ i = 0, \ldots, n$.

5.6 Cross Plots

Parametric Bézier curves are composed of coordinate functions: each component is a Bézier function. For two-dimensional curves, this can be used to construct the *cross plot* of a curve. Figure 5.6 shows the decomposition of a Bézier curve into its two coordinate functions. A cross plot can be a very helpful tool for the investigation not only of Bézier curves, but of general two-dimensional curves. We will use it for the analysis of Bézier and B-spline curves. It can be generalized to more than two dimensions, but is not as useful then.

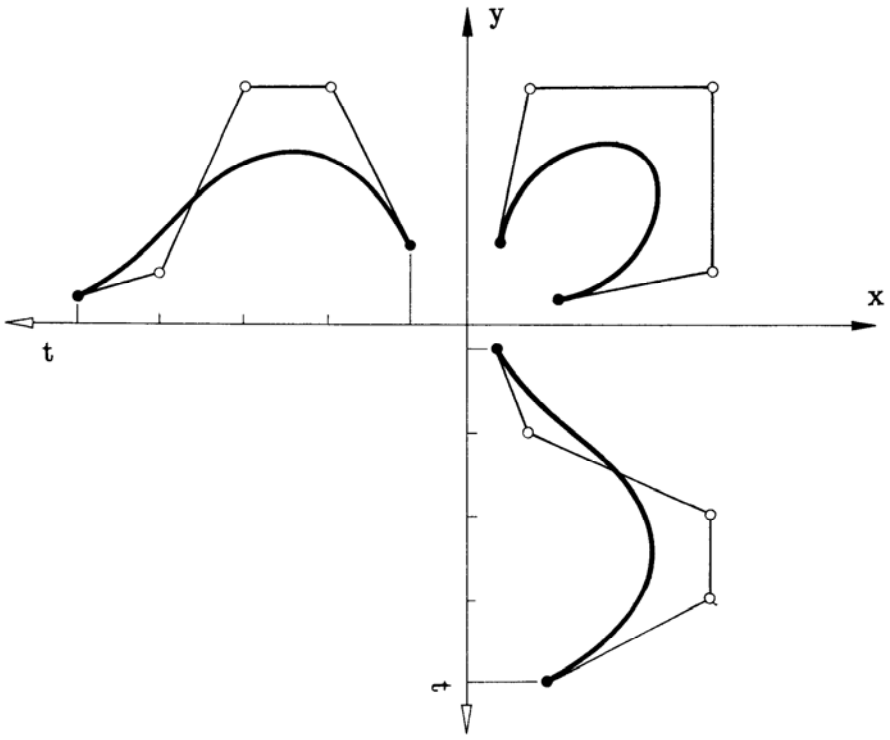

Figure 5.6: Cross plots: a two-dimensional Bézier curve together with its two coordinate functions.

5.7 Integrals

As we have seen, the Bézier polygon $\mathbf{P}$ of a Bézier function is formed by points $(j/n, b_j)$. Let us assign an area $\mathcal{A}\mathbf{P}$ to $\mathbf{P}$ by

$$\mathcal{A}\mathbf{P} = \frac{1}{n+1} \sum_{j=0}^{n} b_j. \tag{5.13}$$

An example for this area is shown in Figure 5.7; it corresponds to approximating the area under the polygon by a particular Riemann sum (of the polygon).

It is now easy to show that this "approximation area" is the same for the polygon $\mathcal{E}\mathbf{P}$, obtained from degree elevation (Section 5.1):

$$\begin{aligned}
\mathcal{A}\mathcal{E}\mathbf{P} &= \frac{1}{n+2} \sum_{j=0}^{n+1} \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j \\
&= \frac{1}{n+2} \sum_{j=0}^{n} \frac{n+2}{n+1} b_j \\
&= \mathcal{A}\mathbf{P}.
\end{aligned}$$

If we repeat the process of degree elevation, we know that the polygons $\mathcal{E}^r\mathbf{P}$ converge to the function $\mathcal{B}\mathbf{P}$. Their area $\mathcal{A}\mathcal{E}^r\mathbf{P}$ stays the same, and in the limit is equal to the Riemann sum of the function, which converges to the integral:

$$\int_0^1 \sum_{j=0}^{n} b_j B_j^n(x)\,\mathrm{d}x = \frac{1}{n+1} \sum_{j=0}^{n} b_j. \tag{5.14}$$

The special case $b_i = \delta_{i,j}$ gives

$$\int_0^1 B_i^n(x)\,\mathrm{d}x = \frac{1}{n+1}, \tag{5.15}$$

i.e., all basis functions B_i^n (for a fixed n) have the same integral.

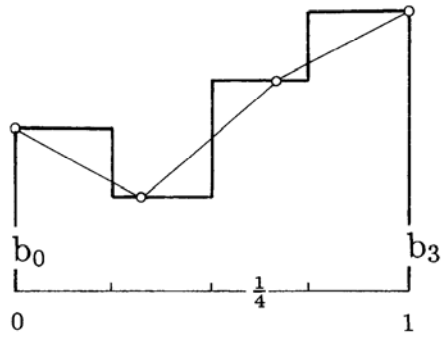

Figure 5.7: Integrals: an approximation to the area under $\mathbf{P}$.

5.8 The Bézier Form of a Bézier Curve

In his work ([50], [51], [52], [53], [54], [55], [57], see also Vernet [490]), Bézier did not use the Bernstein polynomials as basis functions. He wrote the curve $\mathbf{b}^n$ as a linear combination of functions F_i^n:

$$\mathbf{b}^n(t) = \sum_{j=0}^{n} \mathbf{c}_j F_j^n(t), \tag{5.16}$$

where the F_j^n are polynomials that obey the following recursion:

$$F_i^n(t) = (1-t)F_i^{n-1}(t) + tF_{i-1}^{n-1}(t) \tag{5.17}$$

with

$$F_0^0(t) = 1, \quad F_{r+1}^r(t) = 0, \quad F_{-1}^r(t) = 1. \tag{5.18}$$

Note that the third condition in the last equation is the only instance where the definition of the F_i^n differs from that of the B_i^n! An explicit expression for the F_i^n is given by

$$F_i^n = \sum_{j=i}^{n} B_j^n. \tag{5.19}$$

A consequence of (5.18) is that $F_0^n \equiv 1$ for all n. Since $F_j^n(t) \geq 0$ for $t \in [0, 1]$, it follows that (5.16) is not a barycentric combination of the $\mathbf{c}_j$. In fact, $\mathbf{c}_0$ is a point while the other $\mathbf{c}_j$ are vectors. The following relations hold:

$$\mathbf{c}_0 = \mathbf{b}_0, \tag{5.20}$$

$$\mathbf{c}_j = \Delta\mathbf{b}_{j-1}; \quad j > 0. \tag{5.21}$$

This undesirable distinction between points and vectors was abandoned soon after R. Forrest's discovery that the Bézier form (5.16) of a Bézier curve could be written in terms of Bernstein polynomials (see the appendix in [53]). Why is the point-only form more desirable? Just try to write down the de Casteljau algorithm in the point-vector form!

5.9 The Barycentric Form of a Bézier Curve

In this section, we present different notation for Bézier curves that will be useful later. Let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two distinct points on the real line. Then, as described in Section 2.3, we can write any point $\mathbf{p}$ on the straight line in terms of barycentric coordinates of $\mathbf{p}_1$ and $\mathbf{p}_2$: $\mathbf{p} = u\mathbf{p}_1 + v\mathbf{p}_2$, thus identifying $\mathbf{p}$ with $\mathbf{u} = (u, v)$ and $u + v = 1$. In particular, $\mathbf{p}_1 = (1, 0)$ and $\mathbf{p}_2 = (0, 1)$. The real line can be mapped into $\mathbb{E}^3$, where it

defines a polynomial curve $\mathbf{b}(\mathbf{u})$; namely,

$$\mathbf{b}(\mathbf{u}) = \sum_{\substack{i+j=n \\ i,j\geq 0}} \binom{n}{i,\ j} u^i v^j \mathbf{b}_{i,j} = \sum_{\substack{i+j=n \\ i,j\geq 0}} B^n_{i,j}(\mathbf{u})\mathbf{b}_{i,j}, \tag{5.22}$$

where

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

Note that, although (5.22) *looks* bivariate, it really isn't: the condition $u + v = 1$ ensures that we still define a curve, not a surface. The connection with the standard Bézier form is established by setting $t = v$, $\mathbf{b}_j = \mathbf{b}_{i,j}$.

The barycentric form demonstrates nicely two important properties of Bézier curves: invariance under affine parameter transformations and, as a consequence, symmetry, as discussed in Section 3.3. The location of the two points $\mathbf{p}_1$ and $\mathbf{p}_2$ becomes completely irrelevant—all that matters is the relative location of $\mathbf{p}$ with respect to them, described by u and v.

Here is what the de Casteljau algorithm becomes in barycentric notation:

$$\mathbf{b}^r_{i,j}(\mathbf{u}) = u\mathbf{b}^{r-1}_{i+1,j}(\mathbf{u}) + v\mathbf{b}^{r-1}_{i,j+1}(\mathbf{u}) \quad \begin{cases} r = 1, \ldots, n \\ i + j = n - r \end{cases} . \tag{5.23}$$

The point on the curve is then given by $\mathbf{b}^n_{0,0}(\mathbf{u})$.

We can also define derivatives in terms of the barycentric form. Derivatives produce tangent vectors, and these have a sense of direction, which we abandoned for the sake of symmetry. We may reintroduce a direction into our calculations by relating $\mathbf{u}$ to the "standard" parameter t:

$$\mathbf{u} = \mathbf{u}(t) = (1 - t, t).$$

We obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}[\mathbf{u}(t)] = \frac{\partial}{\partial u}\mathbf{b} \cdot \frac{\mathrm{d}u}{\mathrm{d}t} + \frac{\partial}{\partial v}\mathbf{b} \cdot \frac{\mathrm{d}v}{\mathrm{d}t}.$$

Inserting the known values for $\frac{\mathrm{d}u}{\mathrm{d}t}$ and $\frac{\mathrm{d}v}{\mathrm{d}t}$, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}[\mathbf{u}(t)] = \frac{\partial}{\partial v}\mathbf{b} - \frac{\partial}{\partial u}\mathbf{b}. \tag{5.24}$$

If we define a vector $\mathbf{d}$ by $\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1 = (-1, 1)$, this equation may be written as a directional derivative with respect to $\mathbf{d}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}[\mathbf{u}(t)] = D_{\mathbf{d}}\mathbf{b}(\mathbf{u}). \tag{5.25}$$

We shall now see how the de Casteljau algorithm ties in with these directional derivatives.

Instead of evaluating at a *point* $\mathbf{u}$ with $u + v = 1$, let us evaluate at the *vector* $\mathbf{d} = (-1, 1)$. The de Casteljau algorithm (5.23) becomes

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

Thus a *vector* argument for the de Casteljau algorithm produces forward differences! In other words,

$$\mathbf{b}_{i,j}^r(\mathbf{d}) = \Delta^r \mathbf{b}_j,$$

where the term on the right-hand side is in standard, nonbarycentric notation.

We thus have, for the first derivative,

$$D_{\mathbf{d}}\mathbf{b}[\mathbf{u}(t)] = n\sum_{j=0}^{n-1} \Delta \mathbf{b}_j B_j^{n-1}(t) = n \sum_{i+j=n-1} \mathbf{b}_{i,j}^1(\mathbf{d}) B_{i,j}^{n-1}(\mathbf{u}). \tag{5.26}$$

The last part of this equation asserts that our directional derivative is obtained by taking one de Casteljau step with respect to $\mathbf{d}$ and $n - 1$ steps with respect to $\mathbf{u}$. This calls for the blossom notation!

The Bézier points of a curve can be expressed as blossom values of the arguments $\mathbf{p}_1$ and $\mathbf{p}_2$; we thus have three possible ways to label Bézier points, using the standard, the barycentric, and the blossom notation:

$$\mathbf{b}_j = \mathbf{b}_{i,j} = \mathbf{b}[\mathbf{p}_1^{\langle i \rangle}, \mathbf{p}_2^{\langle j \rangle}]; \quad i + j = n.$$

The intermediate points in the de Casteljau algorithm can now be written as

$$\mathbf{b}_{i,j}^r = \mathbf{b}[\mathbf{p}_1^{\langle i \rangle}, \mathbf{p}_2^{\langle j \rangle}, \mathbf{u}^{\langle r \rangle}]; \quad i + j + r = n,$$

and the point on the curve is given by $\mathbf{b}[\mathbf{u}^{\langle n \rangle}]$.

Returning to (5.26), we get

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

The preceding arguments easily generalize this to

$$D_{\mathbf{d}}^r\mathbf{b}(\mathbf{u}) = D_{\mathbf{d}}^r\mathbf{b}[\mathbf{u}^{\langle n \rangle}] = \frac{n!}{(n-r)!}\mathbf{b}[\mathbf{u}^{\langle n-r \rangle}, \mathbf{d}^{\langle r \rangle}]. \tag{5.27}$$

Thus the r^{th} derivative of a curve involves r vector steps and $n - r$ point steps of the de Casteljau algorithm. Of course, it is immaterial in which order these steps are performed. Figure 5.8 illustrates the quadratic case.

We finish this section with an identity that is due to L. Euler. We may formally replace $\mathbf{d}$ by $\mathbf{u}$ in (5.27):

$$D_{\mathbf{u}}^r\mathbf{b}(\mathbf{u}) = \frac{n!}{(n-r)!}\mathbf{b}[\mathbf{u}^{\langle n \rangle}].$$

This shows how closely related the processes of differentiating and evaluating are when we combine the barycentric notation and blossoms.

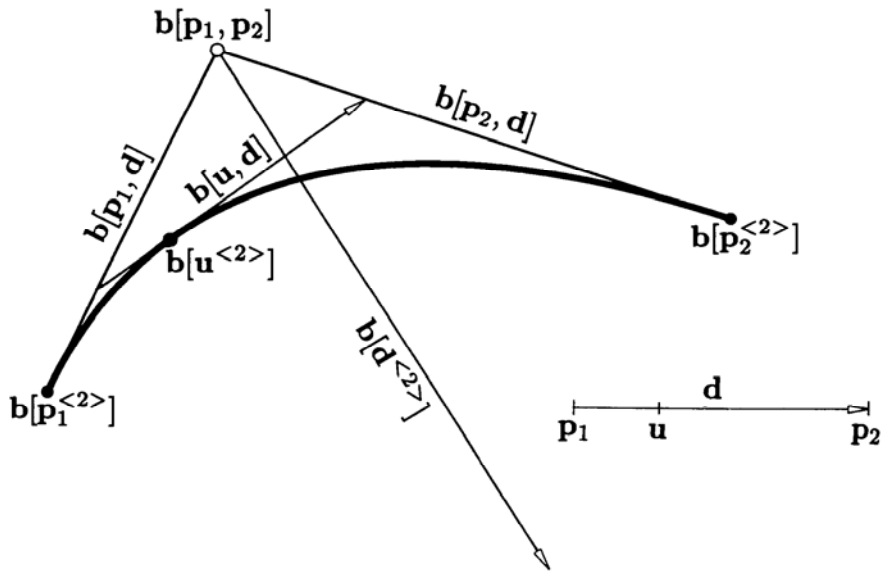

Figure 5.8: Blossoms and derivatives in barycentric form: a point on a quadratic, together with its derivatives: The constant second derivative is given by $\mathbf{b}[\mathbf{d}^{(2)}]$.

5.10 The Weierstrass Approximation Theorem

One of the most important results in approximation theory is the Weierstrass approximation theorem. S. Bernstein invented the polynomials that now bear his name in order to formulate a constructive proof of this theorem. The interested reader is referred to Davis [122] or to Korovkin [314].

We will give a "customized" version of the theorem, namely, we state it in the context of parametric curves. So let $\mathbf{c}$ be a continuous curve that is defined over $[0, 1]$. For some fixed n, we can sample $\mathbf{c}$ at parameter values i/n. The points $\mathbf{c}(i/n)$ can now be interpreted as the Bézier polygon of a polynomial curve $\mathbf{x}_n$:

$$\mathbf{x}_n(t) = \sum_{i=0}^{n} \mathbf{c}\left(\frac{i}{n}\right) B_i^n(t).$$

We say that $\mathbf{x}_n$ is the n^{th} degree Bernstein–Bézier approximation to $\mathbf{c}$.

We are next going to increase the density of our samples, i.e., we increase n. This generates a sequence of approximations $\mathbf{x}_n, \mathbf{x}_{n+1}, \ldots$. The Weierstrass approximation theorem states that this sequence of polynomials converges to the curve $\mathbf{c}$:

$$\lim_{n\to\infty} \mathbf{x}_n(t) = \mathbf{c}(t).$$

At first sight, this looks like a handy way to approximate a given curve by polynomials: we just have to pick a degree n that is sufficiently large, and we are as

close to the curve as we like. This is only theoretically true, however. In practice, one would have to choose values of n in the thousands or even millions in order to obtain a reasonable closeness of fit (see Korovkin [314] for more details).

The value of the theorem is therefore more of a theoretical nature. It shows that every curve may be approximated arbitrarily closely by a polynomial curve.

5.11 Formulas for Bernstein Polynomials

This section is a collection of formulas; some appeared in the text, some did not. Credit for some of these goes to R. Goldman, R. Farouki, and V. Rajan [197].

A Bernstein polynomial is defined by

$$B_i^n(t) = \begin{cases} \binom{n}{i} t^i (1-t)^{n-i} & \text{if } i \in [0, n], \\ 0 & \text{else.} \end{cases}$$

The power basis $\{t^i\}$ and the Bernstein basis $\{B_i^n\}$ are related by

$$t^i = \sum_{j=i}^{n} \frac{\binom{j}{i}}{\binom{n}{i}} B_j^n(t) \tag{5.28}$$

and

$$B_i^n(t) = \sum_{j=i}^{n} (-1)^{j-i} \binom{n}{j} \binom{j}{i} t^j. \tag{5.29}$$

Recursion:

$$B_i^n(t) = (1-t) B_i^{n-1}(t) + t B_{i-1}^{n-1}(t).$$

Subdivision:

$$B_i^n(ct) = \sum_{j=0}^{n} B_i^j(c) B_j^n(t). \tag{5.30}$$

Derivative:

$$\frac{\mathrm{d}}{\mathrm{d}t} B_i^n(t) = n\left[B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right].$$

Integral:

$$\int_0^t B_i^n(x)\,\mathrm{d}x = \frac{1}{n+1} \sum_{j=i+1}^{n+1} B_j^{n+1}(t), \tag{5.31}$$

$$\int_0^1 B_i^n(x)\,\mathrm{d}x = \frac{1}{n+1}.$$

Three degree-elevation formulas:

$$(1-t)B_i^n(t) = \frac{n+1-i}{n+1}B_i^{n+1}(t), \tag{5.32}$$

$$tB_i^n(t) = \frac{i+1}{n+1}B_{i+1}^{n+1}(t), \tag{5.33}$$

$$B_i^n(t) = \frac{n+1-i}{n+1}B_i^{n+1}(t) + \frac{i+1}{n+1}B_{i+1}^{n+1}(t). \tag{5.34}$$

Product:

$$B_i^m(u)B_j^n(u) = \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}}B_{i+j}^{m+n}(u). \tag{5.35}$$

5.12 Implementation

A C routine for degree elevation follows. Note that we have to treat the cases $i = 0$ and $i = n + 1$ separately; the program would not like the corresponding nonexisting array elements. The program actually handles the rational case, which will be covered later. For the polynomial case, fill `wb` with 1's and ignore `wc`.

```
void degree_elevate(bx,by,wb,degree,cx,cy,wc)
/*      input: two-d Bezier polygon in bx, by and with weights
               in wb. Degree is degree.
        Output:degree elevated curve in cx,cy and with weights in wc.
        Note: for nonrational (polynomial) case, fill wc with 1's.
*/
```

5.13 Exercises

*1. Prove (5.19).

*2. Prove the relationship between the "Bézier" and the Bernstein form for a Bézier curve (5.16).

*3. Prove that

$$\int_0^t b^n(x)\,dx = \frac{t}{n+1}\sum_{j=0}^{n} b_0^j(t).$$

*4. With the result from the previous problem, prove

$$F_i^n(t) = n\int_0^t B_i^{n-1}(x)\,dx.$$

*5. Show that the control points $\vec{\mathbf{b}}_i$ from (5.7) define a curve that is the original curve's Taylor expansion of degree $n-1$ at $t=0$.

P1. The recursion formula for Bernstein polynomials is equivalent to the de Casteljau algorithm. Devise a recursive curve evaluation algorithm for curves in Chebychev form based on the recursion for Chebychev polynomials. Program it up and experiment!

P2. Program up degree reduction with some of the methods outlined in Section 5.4. Work with the Bézier polygon supplied in the file `degred.dat`.

Chapter 6

Polynomial Interpolation

Polynomial interpolation is the most fundamental of all interpolation concepts; the earliest method is probably attributable to I. Newton. Nowadays, polynomial interpolation is mostly of theoretical value; faster and more accurate methods have been developed. Those methods are *piecewise polynomial*; thus they intrinsically rely on the polynomial methods that are presented in this chapter.

6.1 Aitken's Algorithm

A common problem in curve design is *point data interpolation*: from data points $\mathbf{p}_i$ with corresponding parameter values t_i, find a curve that passes through the $\mathbf{p}_i$.[1] One of the oldest techniques to solve this problem is to find an *interpolating polynomial* through the given points. That polynomial must satisfy the interpolatory constraints

$$\mathbf{p}(t_i) = \mathbf{p}_i; \quad i = 0, \ldots, n.$$

Several algorithms exist for this problem—any textbook on numerical analysis will discuss several of them. In this section we shall present a recursive technique that is due to A. Aitken.

We have already solved the linear case, $n = 1$, in Section 2.3. The Aitken recursion computes a point on the interpolating polynomial through a sequence of *repeated linear interpolations*, starting with

$$\mathbf{p}_i^1(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i}\mathbf{p}_i + \frac{t - t_i}{t_{i+1} - t_i}\mathbf{p}_{i+1}; \quad i = 0, \ldots, n-1.$$

Let us now suppose (as one does in recursive techniques) that we have already solved the problem for the case $n - 1$. To be more precise, assume that we have found a polynomial $\mathbf{p}_0^{n-1}$ that interpolates to the n first data points $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$, and also

[1] The shape of the curve depends heavily on the parameter values t_i. Methods for their determination will be discussed later in the context of spline interpolation; see Section 9.4.

a polynomial $\mathbf{p}_1^{n-1}$ that interpolates to the n last data points $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Under these assumptions, it is easy to write down the form of the final interpolant, now called $\mathbf{p}_0^n$:

$$\mathbf{p}_0^n(t) = \frac{t_n - t}{t_n - t_0}\mathbf{p}_0^{n-1}(t) + \frac{t - t_0}{t_n - t_0}\mathbf{p}_1^{n-1}(t). \tag{6.1}$$

Figure 6.1 illustrates this form for the cubic case.

Let us verify that (6.1) does in fact interpolate to all given data points $\mathbf{p}_i$: for $t = t_0$,

$$\mathbf{p}_0^n(t_0) = 1 * \mathbf{p}_0^{n-1}(t_0) + 0 * \mathbf{p}_1^{n-1}(t_0) = \mathbf{p}_0.$$

A similar result is derived for $t = t_n$. Under our assumption, we have $\mathbf{p}_0^{n-1}(t_i) = \mathbf{p}_1^{n-1}(t_i) = \mathbf{p}_i$ for all other values of i.

Since the weights in (6.1) sum to one identically, we get the desired $\mathbf{p}_0^n(t_i) = \mathbf{p}_i$.

We can now generalize (6.1) to solve the polynomial interpolation problem: starting with the given parameter values t_i and the data points $\mathbf{p}_i = \mathbf{p}_i^0$, we set

$$\mathbf{p}_i^r(t) = \frac{t_{i+r} - t}{t_{i+r} - t_i}\mathbf{p}_i^{r-1}(t) + \frac{t - t_i}{t_{i+r} - t_i}\mathbf{p}_{i+1}^{r-1}(t); \begin{cases} r = 1, \ldots, n; \\ i = 0, \ldots, n - r \end{cases} . \tag{6.2}$$

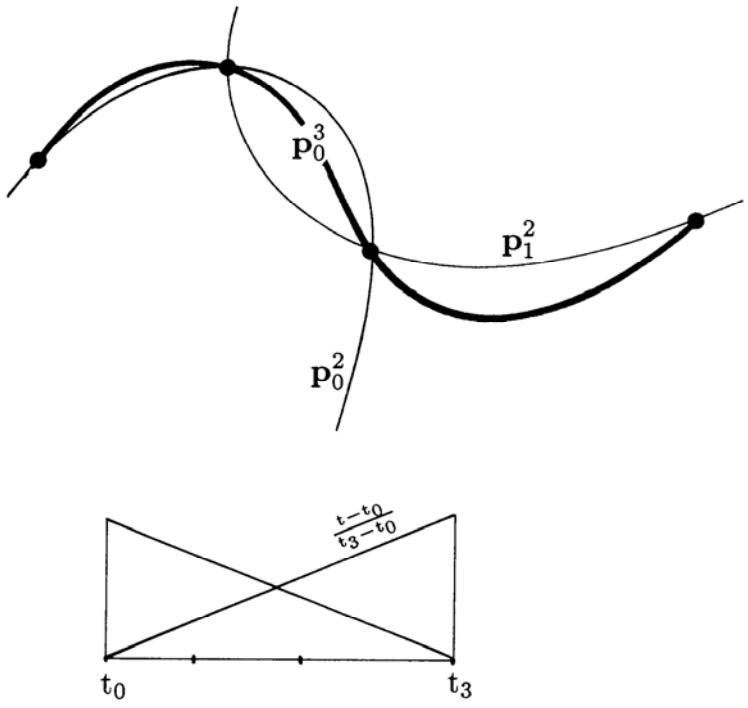

Figure 6.1: Polynomial interpolation: a cubic interpolating polynomial may be obtained as a "blend" of two quadratic interpolants.

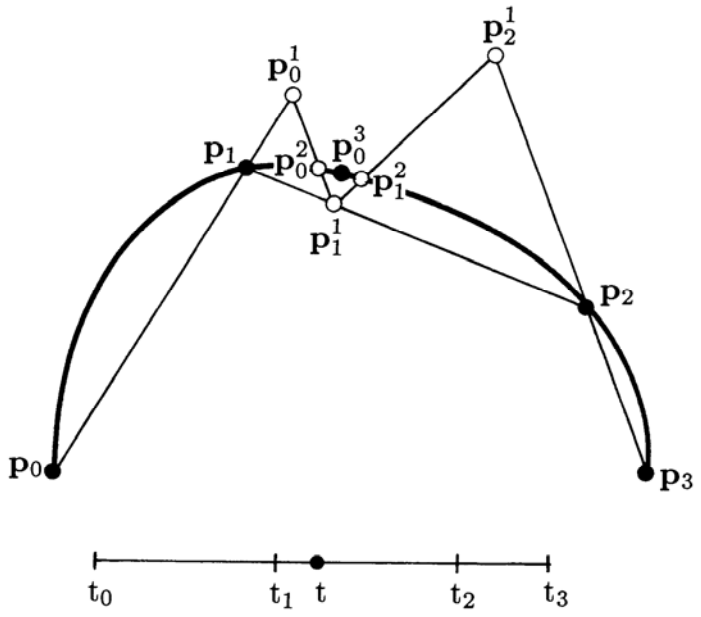

Figure 6.2: Aitken's algorithm: a point on an interpolating polynomial may be found from repeated linear interpolation.

It is clear from the preceding consideration that $\mathbf{p}_0^n(t)$ is indeed a point on the interpolating polynomial. The recursive evaluation (6.2) is called *Aitken's algorithm.*[2]

It has the following geometric interpretation: to find $\mathbf{p}_i^r$, map the interval $[t_i, t_{i+r}]$ onto the straight line segment through $\mathbf{p}_i^{r-1}$, $\mathbf{p}_{i+1}^{r-1}$. That affine map takes t to $\mathbf{p}_i^r$. The geometry of Aitken's algorithm is illustrated in Figure 6.2 for the cubic case.

It is convenient to write the intermediate $\mathbf{p}_i^r$ in a triangular array; the cubic case would look like

$$\begin{array}{llll} \mathbf{p}_0 & & & \\ \mathbf{p}_1 & \mathbf{p}_0^1 & & \\ \mathbf{p}_2 & \mathbf{p}_1^1 & \mathbf{p}_0^2 & \\ \mathbf{p}_3 & \mathbf{p}_2^1 & \mathbf{p}_1^2 & \mathbf{p}_0^3. \end{array} \tag{6.3}$$

We can infer several properties of the interpolating polynomial from Aitken's algorithm:

- *Affine invariance:* This follows since the Aitken algorithm uses only barycentric combinations.

[2]The particular organization of the algorithm as presented here is due to Neville.

- *Linear precision*: If all $\mathbf{p}_i$ are uniformly distributed[3] on a straight line segment, all intermediate $\mathbf{p}_i^r(t)$ are identical for $r > 0$. Thus the straight line segment is reproduced.

- *No convex hull property:* The parameter t in (6.2) does not have to lie between t_i and t_{i+r}. Therefore, Aitken's algorithm does not use convex combinations only: $\mathbf{p}_0^n(t)$ is not guaranteed to lie within the convex hull of the $\mathbf{p}_i$. We should note, however, that no smooth curve interpolation scheme exists that has the convex hull property.

- *No variation diminishing property:* By the same reasoning, we do not get the variation diminishing property. Again, no "decent" interpolation scheme has this property. However, interpolating polynomials can be variation augmenting to an extent that renders them useless for practical problems.

6.2 Lagrange Polynomials

Aitken's algorithm allows us to compute a point $\mathbf{p}^n(t)$ on the interpolating polynomial through $n + 1$ data points. It does not provide an answer to the following questions: (1) Is the interpolating polynomial unique? (2) What is a closed form for it? Both questions are resolved by the use of the *Lagrange polynomials* L_i^n.

The explicit form of the interpolating polynomial $\mathbf{p}$ is given by

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{p}_i L_i^n(t), \tag{6.4}$$

where the L_i^n are *Lagrange polynomials*,

$$L_i^n(t) = \frac{\prod_{j=0, j\neq i}^{n} (t - t_j)}{\prod_{j=0, j\neq i}^{n} (t_i - t_j)}. \tag{6.5}$$

Before we proceed further, we should note that the L_i^n must sum to one in order for (6.4) to be a barycentric combination and thus be geometrically meaningful; we will return to this topic later.

We verify (6.4) by observing that the Lagrange polynomials are *cardinal*: they satisfy

$$L_i^n(t_j) = \delta_{i,j}, \tag{6.6}$$

with $\delta_{i,j}$ being the Kronecker delta. In other words, the i^{th} Lagrange polynomial vanishes at all knots except at the i^{th} one, where it assumes the value 1. Because of this property of Lagrange polynomials, (6.4) is called the *cardinal* form of the interpolating polynomial $\mathbf{p}$. The polynomial $\mathbf{p}$ has many other representations, of

[3] If the points are on a straight line, but distributed unevenly, we will still recapture the graph of the straight line, but it will not be parametrized linearly.

course (we can rewrite it in monomial form, for example), but (6.4) is the only form in which the data points appear explicitly.

We have thus justified our use of the term *the* interpolating polynomial. In fact, the polynomial interpolation problem always has a solution, and it always has a *unique* solution. The reason is that, because of (6.6), the L_i^n form a basis of all polynomials of degree n. Thus (6.4) is the unique representation of the polynomial $\mathbf{p}$ in this basis. This is why one sometimes refers to all polynomial interpolation schemes as *Lagrange interpolation*.[4]

We can now be sure that Aitken's algorithm yields the same point as does (6.4). Based on that knowledge, we can conclude a property of Lagrange polynomials that was already mentioned right after (6.5), namely, that they sum to one:

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

This is a simple consequence of the affine invariance of polynomial interpolation, as shown for Aitken's algorithm.

6.3 The Vandermonde Approach

Suppose we want the interpolating polynomial $\mathbf{p}^n$ in the monomial basis:

$$\mathbf{p}^n(t) = \sum_{j=0}^{n} \mathbf{a}_j t^j. \tag{6.7}$$

The standard approach to finding the unknown coefficients from the known data is simply to write down everything one knows about the problem:

$$\begin{aligned}
\mathbf{p}^n(t_0) = \mathbf{p}_0 &= \mathbf{a}_0 + \mathbf{a}_1 t_0 + \cdots + \mathbf{a}_n t_0^n, \\
\mathbf{p}^n(t_1) = \mathbf{p}_1 &= \mathbf{a}_0 + \mathbf{a}_1 t_1 + \cdots + \mathbf{a}_n t_1^n, \\
&\vdots \\
\mathbf{p}^n(t_n) = \mathbf{p}_n &= \mathbf{a}_0 + \mathbf{a}_1 t_n + \cdots + \mathbf{a}_n t_n^n.
\end{aligned}$$

In matrix form:

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & \dots & t_0^n \\ 1 & t_1 & t_1^2 & \dots & t_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & t_n & t_n^2 & \dots & t_n^n \end{bmatrix} \begin{bmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{bmatrix}. \tag{6.8}$$

[4]More precisely, we refer to all those schemes that interpolate to a given set of data points. Other forms of polynomial interpolation exist and are discussed later.

We can shorten this to

$$\mathbf{p} = T\mathbf{a}. \tag{6.9}$$

We already know that a solution $\mathbf{a}$ to this linear system exists, but one can show independently that the determinant $\det T$ is nonzero (for distinct parameter values t_i). This determinant is known as the *Vandermonde* of the interpolation problem. The solution, i.e., the vector $\mathbf{a}$ containing the coefficients $\mathbf{a}_i$, can be found from

$$\mathbf{a} = T^{-1}\mathbf{p}. \tag{6.10}$$

This should be taken only as a shorthand notation for the solution—not as an algorithm! Note that the linear system (6.9) really consists of *three* linear systems with the same coefficient matrix, one system for each coordinate. It is known from numerical analysis that in such cases the LU decomposition of T is a more economical way to obtain the solution $\mathbf{a}$. This will be even more important when we discuss tensor product surface interpolation in Section 15.12.

The interpolation problem can also be solved if we use basis functions other than the monomials. Let $\{F_i^n\}_{i=0}^n$ be such a basis. We then seek an interpolating polynomial of the form

$$\mathbf{p}^n(t) = \sum_{j=0}^{n} \mathbf{c}_j F_i^n j(t). \tag{6.11}$$

The preceding reasoning again leads to a linear system (three linear systems, to be more precise) for the coefficients $\mathbf{c}_j$, this time with the *generalized Vandermonde* F:

$$F = \begin{bmatrix} F_0^n(t_0) & F_1^n(t_0) & \dots & F_n^n(t_0) \\ F_0^n(t_1) & F_1^n(t_1) & \dots & F_n^n(t_1) \\ \vdots & \vdots & & \vdots \\ F_0^n(t_n) & F_1^n(t_n) & \dots & F_n^n(t_n) \end{bmatrix}. \tag{6.12}$$

Since the F_i^n form a basis for all polynomials of degree n, it follows that the generalized Vandermonde $\det F$ is nonzero.

Thus, for instance, we are able to find the Bézier curve that passes through a given set of data points: the F_i^n would then be the Bernstein polynomials B_i^n.

6.4 Limits of Lagrange Interpolation

We have seen that polynomial interpolation is simple, unique, and has a nice geometric interpretation. One might therefore expect this interpolation scheme to be used frequently; yet it is virtually unknown in a design environment. The main reason is illustrated in Figure 6.3: polynomial interpolants *oscillate*. For quite reasonable data points and parameter values, the polynomial interpolant exhibits wild wiggles that are not inherent in the data. One may say that polynomial interpolation is not *shape preserving*.

Figure 6.3: Lagrange interpolation: while the data points suggest a convex interpolant, the Lagrange interpolant exhibits extraneous wiggles.

This phenomenon is not due to numerical effects; it is actually inherent in the polynomial interpolation process. Suppose we are given a finite arc of a smooth curve $\mathbf{c}$. We can then sample the curve at parameter values t_i and pass the interpolating polynomial through those points. If we increase the number of points on the curve, thus producing interpolants of higher and higher degree, one would expect the corresponding interpolants to converge to the sampled curve $\mathbf{c}$. But this is not generally true: smooth curves exist for which this sequence of interpolants diverges. This fact is dealt with in numerical analysis, where it is known by the name of its discoverer: it is called the "Runge phenomenon" [427]. Note, however, that the Runge phenomenon does *not* contradict the Weierstrass approximation theorem!

As a second consideration, let us examine the cost of polynomial interpolation, i.e., the number of operations necessary to construct and then evaluate the interpolant. Solving the Vandermonde system (6.8) requires roughly n^3 operations; subsequent computation of a point on the curve requires n operations. The operation count for the construction of the interpolant is much smaller for other schemes, as is the cost of evaluations (here piecewise schemes are far superior). This latter cost is the more important one, of course: construction of the interpolant happens once, but it may have to be evaluated thousands of times!

6.5 Cubic Hermite Interpolation

Polynomial interpolation is not restricted to interpolation to point data; one can also interpolate to other information, such as derivative data. This leads to an interpolation scheme that is more useful than Lagrange interpolation: it is called *Hermite interpolation*. We treat the cubic case first, in which one is given two points $\mathbf{p}_0, \mathbf{p}_1$ and two tangent vectors $\mathbf{m}_0, \mathbf{m}_1$. The objective is to find a cubic polynomial curve $\mathbf{p}$ that interpolates to these data:

$$\begin{aligned} \mathbf{p}(0) &= \mathbf{p}_0, \\ \dot{\mathbf{p}}(0) &= \mathbf{m}_0, \\ \dot{\mathbf{p}}(1) &= \mathbf{m}_1, \\ \mathbf{p}(1) &= \mathbf{p}_1, \end{aligned}$$

where the dot denotes differentiation.

We will write $\mathbf{p}$ in cubic Bézier form, and therefore must determine four Bézier points $\mathbf{b}_0, \ldots, \mathbf{b}_3$. Two of them are quickly determined:

$$\mathbf{b}_0 = \mathbf{p}_0, \quad \mathbf{b}_3 = \mathbf{p}_1.$$

For the remaining two, we recall (from Section 4.3) the endpoint derivative for Bézier curves:

$$\dot{\mathbf{p}}(0) = 3\Delta\mathbf{b}_0, \quad \dot{\mathbf{p}}(1) = 3\Delta\mathbf{b}_2.$$

We can easily solve for $\mathbf{b}_1$ and $\mathbf{b}_2$:

$$\mathbf{b}_1 = \mathbf{p}_0 + \frac{1}{3}\mathbf{m}_0, \quad \mathbf{b}_2 = \mathbf{p}_1 - \frac{1}{3}\mathbf{m}_1.$$

This situation is shown in Figure 6.4.

Having solved the interpolation problem, we now attempt to write it in *cardinal form*; we would like to have the given data appear *explicitly* in the equation for the interpolant. So far, our interpolant is in Bézier form:

$$\mathbf{p}(t) = \mathbf{p}_0 B_0^3(t) + \left(\mathbf{p}_0 + \frac{1}{3}\mathbf{m}_0\right) B_1^3(t) + \left(\mathbf{p}_1 - \frac{1}{3}\mathbf{m}_1\right) B_2^3(t) + \mathbf{p}_1 B_3^3(t).$$

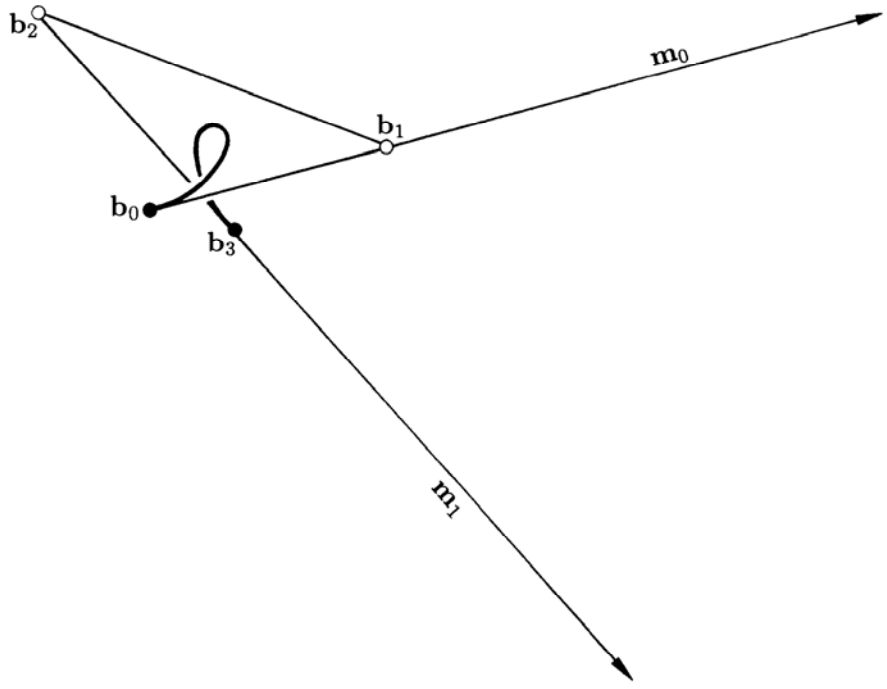

Figure 6.4: Cubic Hermite interpolation: the given data—points and tangent vectors—together with the interpolating cubic in Bézier form.

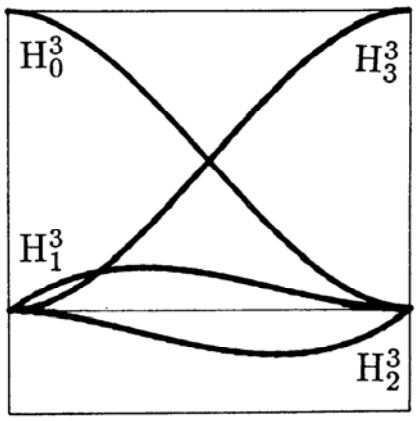

Figure 6.5: Cubic Hermite polynomials: the four H_i^3 are shown over the interval [0, 1].

To obtain the cardinal form, we simply rearrange:

$$\mathbf{p}(t) = \mathbf{p}_0 H_0^3(t) + \mathbf{m}_0 H_1^3(t) + \mathbf{m}_1 H_2^3(t) + \mathbf{p}_1 H_3^3(t), \tag{6.13}$$

where we have set[5]

$$\begin{aligned} H_0^3(t) &= B_0^3(t) + B_1^3(t), \\ H_1^3(t) &= \frac{1}{3} B_1^3(t), \\ H_2^3(t) &= -\frac{1}{3} B_2^3(t), \\ H_3^3(t) &= B_2^3(t) + B_3^3(t). \end{aligned} \tag{6.14}$$

The H_i^3 are called "cubic Hermite polynomials" and are shown in Figure 6.5.

What are the properties necessary to make the H_i^3 cardinal functions for the cubic Hermite interpolation problem? They must be cardinal with respect to evaluation and differentiation at $t = 0$ and $t = 1$, i.e., each of the H_i^3 equals 1 for one of these four operations and is zero for the remaining three:

$$\begin{aligned} &H_0^3(0) = 1, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_0^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_0^3(1) = 0, \quad H_0^3(1) = 0, \\ &H_1^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_1^3(0) = 1, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_1^3(1) = 0, \quad H_1^3(1) = 0, \\ &H_2^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_2^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_2^3(1) = 1, \quad H_2^3(1) = 0, \\ &H_3^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_3^3(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t} H_3^3(1) = 0, \quad H_3^3(1) = 1. \end{aligned}$$

[5]This is a deviation from standard notation. Standard notation groups by orders of derivatives, i.e., first the two positions, then the two derivatives. The form of (6.13) was chosen because it groups coefficients according to their geometry.

Another important property of the H_i^3 follows from the geometry of the interpolation problem; (6.13) contains combinations of points and vectors. We know that the point coefficients must sum to one if (6.13) is to be geometrically meaningful:

$$H_0^3(t) + H_3^3(t) \equiv 1.$$

This is of course also verified by inspection of (6.14).

Cubic Hermite interpolation has one annoying peculiarity: it is not invariant under affine domain transformations. Let a cubic Hermite interpolant be given as in (6.13), i.e., having the interval [0, 1] as its domain. Now apply an affine domain transformation to it by changing t to $\hat{t} = (1 - t)a + tb$, thereby changing [0, 1] to some $[a, b]$. The interpolant (6.13) becomes

$$\hat{\mathbf{p}}(\hat{t}) = \mathbf{p}_0\hat{H}_0^3(\hat{t}) + \mathbf{m}_0\hat{H}_1^3(\hat{t}) + \mathbf{m}_1\hat{H}_2^3(\hat{t}) + \mathbf{p}_1\hat{H}_3^3(\hat{t}), \tag{6.15}$$

where the $\hat{H}_i^3(\hat{t})$ are defined through their cardinal properties:

$$\begin{aligned}
&\hat{H}_0^3(a) = 1, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_0^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_0^3(b) = 0, \quad \hat{H}_0^3(b) = 0,\\
&\hat{H}_1^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_1^3(a) = 1, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_1^3(b) = 0, \quad \hat{H}_1^3(b) = 0,\\
&\hat{H}_2^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_2^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}H_2^3(b) = 1, \quad H_2^3(b) = 0,\\
&\hat{H}_3^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_3^3(a) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\hat{H}_3^3(b) = 0, \quad \hat{H}_3^3(b) = 1.
\end{aligned}$$

To satisfy these requirements, the new $\hat{H}_i^3$ must differ from the original H_i^3. We obtain

$$\begin{aligned}
\hat{H}_0^3(\hat{t}) &= H_0^3(t),\\
\hat{H}_1^3(\hat{t}) &= (b - a)H_1^3(t),\\
\hat{H}_2^3(\hat{t}) &= (b - a)H_2^3(t),\\
\hat{H}_3^3(\hat{t}) &= H_3^3(t),
\end{aligned} \tag{6.16}$$

where $t \in [0, 1]$ is the local parameter of the interval $[a, b]$.

Evaluation of (6.15) at $\hat{t} = a$ and $\hat{t} = b$ yields $\hat{\mathbf{p}}(a) = \mathbf{p}_0$, $\hat{\mathbf{p}}(b) = \mathbf{p}_1$. The derivatives have changed, however. Invoking the chain rule, we find that $\mathrm{d}\hat{\mathbf{p}}(a)/\mathrm{d}t = (b - a)\mathbf{m}_0$ and, similarly, $\mathrm{d}\hat{\mathbf{p}}(b)/\mathrm{d}t = (b - a)\mathbf{m}_1$.

Thus an affine domain transformation changes the curve unless the defining tangent vectors are changed accordingly—a drawback that is not encountered with the Bernstein–Bézier form.

To maintain the same curve after a domain transformation, we must change the length of the tangent vectors: if the length of the domain interval is changed by a factor α, we must replace $\mathbf{m}_0$ and $\mathbf{m}_1$ by $\mathbf{m}_0/\alpha$ and $\mathbf{m}_1/\alpha$, respectively. There is an intuitive argument for this: interpreting the parameter as time, we assume we had one time unit to traverse the curve. After changing the interval length by a factor of 10, for example, we have 10 time units to traverse the same curve, resulting in a much

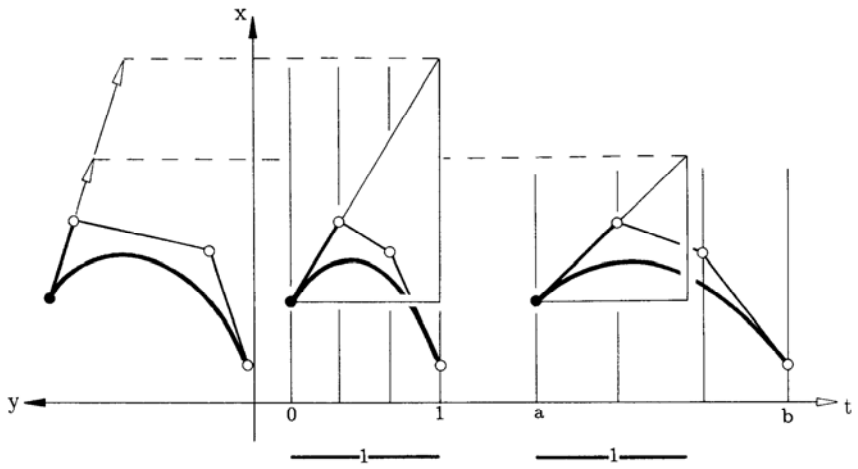

Figure 6.6: Lengths of tangent vectors and domain intervals: the longer the domain interval (right cubic function), the shorter the tangent vector of the parametric curve.

lower speed of traversal. Since the magnitude of the derivative equals that speed, it must also shrink by a factor of 10.

An illustration is given in Figure 6.6. It shows—using a parametric cubic and the x-portion of its cross plot—how a stretching of the domain interval "flattens" the x-component function. This results in a shorter tangent vector of the parametric curve. In this figure, we have made use of the fact that the slope of a function may be expressed as the height of a right triangle with base length one.

We also note that the Hermite form is not symmetric: if we replace t by $1 - t$ (assuming again the interval [0, 1] as the domain), the curve coefficients cannot simply be renumbered (as in the case of Bézier curves). Rather, the tangent vectors must be *reversed*. This follows from the foregoing application of the affine map to the [0, 1] that maps that interval to [1, 0], thus reversing its direction.

The dependence of the cubic Hermite form on the domain interval is rather unpleasant—it is often overlooked and can be blamed for countless programming errors by both students and professionals. We will use the Bézier form whenever possible.

6.6 Quintic Hermite Interpolation

Instead of prescribing only position and first derivative information at two points, one might add information for second-order derivatives. Then our data are $\mathbf{p}_0$, $\mathbf{m}_0$, $\mathbf{s}_0$ and $\mathbf{p}_1$, $\mathbf{m}_1$, $\mathbf{s}_1$, where $\mathbf{s}_0$ and $\mathbf{s}_1$ denote second derivatives. The lowest order polynomial

to interpolate to these data is of degree five. Its Bézier points are easily obtained following the preceding approach. If we rearrange the Bézier form to obtain a cardinal form of the interpolant $\mathbf{p}$, we find

$$\mathbf{p}(t) = \mathbf{p}_0 H_0^5(t) + \mathbf{m}_0 H_1^5(t) + \mathbf{s}_0 H_2^5(t) + \mathbf{s}_1 H_3^5(t) + \mathbf{m}_1 H_4^5(t) + \mathbf{p}_1 H_5^5(t), \quad (6.17)$$

where

$$\begin{aligned} H_0^5 &= B_0^5 + B_1^5 + B_2^5, \\ H_1^5 &= \frac{1}{5}[B_1^5 + 2B_2^5], \\ H_2^5 &= \frac{1}{20}B_2^5, \\ H_3^5 &= \frac{1}{20}B_3^5, \\ H_4^5 &= -\frac{1}{5}[2B_3^5 + B_4^5], \\ H_5^5 &= B_3^5 + B_4^5 + B_5^5. \end{aligned}$$

It is easy to verify the cardinal properties of the H_i^5: they are the straightforward generalization of the cardinal properties for cubic Hermite polynomials. If used in the context of piecewise curves, the quintic Hermite polynomials guarantee C^2 continuity since adjoining curve pieces interpolate to the same second-order data. For most applications, one will have to estimate the second derivatives that are needed as input. This estimation is a very sensitive procedure—so unless the quintic form is mandated by a particular problem, the simpler C^2 cubic splines from Chapter 9 are recommended.

6.7 The Newton Form and Forward Differencing

All methods in this chapter—and in the Bézier curve chapters as well—were concerned with the construction of polynomial curves. We shall now introduce a way to display or plot such curves. The underlying theory makes use of the *Newton form of a polynomial*; the resulting display algorithm is called *forward differencing* and is well established in the computer graphics community. For this section, we only deal with the cubic case; the general case is then not hard to work out.

So suppose that we are given a cubic polynomial curve $\mathbf{p}(t)$. Also suppose that we are given four points $\mathbf{p}(t_0)$, $\mathbf{p}(t_1)$, $\mathbf{p}(t_2)$, $\mathbf{p}(t_3)$ on it such that $t_{i+1} - t_i = h$, i.e., they are at equally spaced parameter intervals. Then it can be shown that this polynomial

may be written as

$$\mathbf{p}(t) = \mathbf{p}_0 + \frac{1}{h}(t-t_0)\Delta\mathbf{p}_0 + \frac{1}{2!h^2}(t-t_0)(t-t_1)\Delta^2\mathbf{p}_0 + \frac{1}{3!h^3}(t-t_0)(t-t_1)(t-t_2)\Delta^3\mathbf{p}_0. \tag{6.18}$$

The derivation of this *Newton form* is in any standard text on numerical analysis.

The differences $\Delta^i\mathbf{p}_j$ are defined as

$$\Delta^i\mathbf{p}_j = \Delta^{i-1}\mathbf{p}_{j+1} - \Delta^{i-1}\mathbf{p}_j \tag{6.19}$$

and $\Delta^0\mathbf{p}_j = \mathbf{p}_j$.

The coefficients in (6.18) are conveniently written in a table such as the following (setting $g = 1/h$):

$$\begin{array}{llll} \mathbf{p}_0 & & & \\ \mathbf{p}_1 & g\Delta\mathbf{p}_0 & & \\ \mathbf{p}_2 & g\Delta\mathbf{p}_1 & g^2\Delta^2\mathbf{p}_0 & \\ \mathbf{p}_3 & g\Delta\mathbf{p}_2 & g^2\Delta^2\mathbf{p}_1 & g^3\Delta^3\mathbf{p}_0. \end{array}$$

The diagonal contains the coefficients of the Newton form. The computation of this table is called the *startup phase* of the forward differencing scheme.

We could now evaluate $\mathbf{p}$ at any parameter value t by simply evaluating (6.18) there. Since our our evaluation points t_j are equally spaced, a much faster way exists. Suppose we had computed $\mathbf{p}_j = \mathbf{p}(t_j)$, etc., from (6.18). Then we could compute all entries in the following table:

$$\begin{array}{llll} \mathbf{p}_0 & & & \\ \mathbf{p}_1 & g\Delta\mathbf{p}_0 & & \\ \mathbf{p}_2 & g\Delta\mathbf{p}_1 & g^2\Delta^2\mathbf{p}_0 & \\ \mathbf{p}_3 & g\Delta\mathbf{p}_2 & g^2\Delta^2\mathbf{p}_1 & g^3\Delta^3\mathbf{p}_0 \\ \mathbf{p}_4 & g\Delta\mathbf{p}_3 & g^2\Delta^2\mathbf{p}_2 & g^3\Delta^3\mathbf{p}_1 \\ \mathbf{p}_5 & g\Delta\mathbf{p}_4 & g^2\Delta^2\mathbf{p}_3 & g^3\Delta^3\mathbf{p}_2 \\ \vdots & \vdots & \vdots & \vdots \end{array}. \tag{6.20}$$

Now consider the last column of this table, containing terms of the form $g^3\Delta^3\mathbf{p}_j$. All these terms are equal! This is so because the third derivative of a third-degree polynomial is constant, and because the third derivative of (6.18) is given by $g^3\Delta^3\mathbf{p}_0 = g^3\Delta^3\mathbf{p}_1 = \ldots$.

We thus have a new way of constructing the table (6.20) from *right to left:* instead of computing the entry $\mathbf{p}_4$ from (6.18), first compute $g^2\Delta^2\mathbf{p}_2$ from (6.19):

$$g^2\Delta^2\mathbf{p}_2 = g^3\Delta^3\mathbf{p}_1 + g^2\Delta^2\mathbf{p}_1,$$

then compute $g\Delta\mathbf{p}_3$ from

$$g\Delta\mathbf{p}_3 = g^2\Delta^2\mathbf{p}_2 + g\Delta\mathbf{p}_2,$$

and finally

$$\mathbf{p}_4 = g\Delta\mathbf{p}_3 + \mathbf{p}_3.$$

Then compute $\mathbf{p}_5$ in the same manner, and so on. The general formula is, with $\mathbf{q}_j^i = g^i \Delta^i \mathbf{p}_j$:

$$\mathbf{q}_j^i = \mathbf{q}_{j-1}^{i+1} + \mathbf{q}_{j-1}^i; \quad i = 2, 1, 0. \tag{6.21}$$

It yields the points $\mathbf{p}_j = \mathbf{q}_j^0$.[6]

This way of computing the $\mathbf{p}_j$ does not involve a single multiplication after the startup phase! It is therefore extremely fast and has been implemented in many graphics systems. Given four initial points $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and a stepsize h, it generates a sequence of points on the cubic polynomial through the initial four points. Typically the polynomial will be given in Bézier form, so those four points have to be computed as a startup operation.

In a graphics environment, it is desirable to adjust the stepsize h such that each pixel along the curve is hit. One way of doing this is to adjust the stepsize while marching along the curve. This is called *adaptive forward differencing* and is described by Lien, Shantz, and Pratt [333] and by Chang, Shantz, and Rochetti [99].

Although fast, forward differencing is not foolproof: As we compute more and more points on the curve, they begin to be affected by roundoff. So while we intend to march along our curve, we may instead leave its path, deviating from it more and more as we continue. For more literature on this method, see Abi-Ezzi [1], Bartels *et al.* [42], or Shantz and Chang [472].

6.8 Implementation

The code for Aitken's algorithm is very similar to that for the de Casteljau algorithm. Here is its header:

```
float aitken(degree,coeff,t)
/*   uses  Aitken to compute one coordinate
     value of a  Lagrange interpolating polynomial. Has to be called
     for each coordinate  (x,y, and/or z) of data points.
Input:   degree: degree of curve.
         coeff:  array with coordinates to be interpolated.
         t:      parameter value.
Output:  coordinate value.

      Note: we  assume a uniform knot sequence!
*/
```

6.9 Exercises

1. Show that the cubic and quintic Hermite polynomials are linearly independent.
2. Generalize Hermite interpolation to degrees 7, 9, etc.

[6] It holds for any degree n if we replace $i = 2, 1, 0$ by $i = n-1, n-2, \ldots, 0$.

*3. The de Casteljau algorithm for Bézier curves has as its "counterpart" the recursion formula (4.2) for Bernstein polynomials. Deduce a recursion formula for Lagrange polynomials from Aitken's algorithm.

*4. The Hermite form is not invariant under affine domain transformations, while the Bézier form is. What about the Lagrange and monomial forms? What are the general conditions for a curve scheme to be invariant under affine domain transformations?

P1. Aitken's algorithm looks very similar to the de Casteljau algorithm. Use both to define a whole class of algorithms, of which each would be a special case (see [184]). Write a program that uses as input a parameter specifying if the output curve should be "more Bézier" or "more Lagrange."

P2. The function that was used by Runge to demonstrate the effect that now bears his name is given by

$$f(x) = \frac{1}{1+x^2}; \quad x \in [-1, 1].$$

Use the routine `aitken` to interpolate at equidistant parameter intervals. Keep increasing the degree of the interpolating polynomial until you notice "bad" behavior on the part of the interpolant.

P3. In Lagrange interpolation, each $\mathbf{p}_i$ is assigned a corresponding parameter value t_i. Experiment (graphically) by interchanging two parameter values t_i and t_j without interchanging $\mathbf{p}_i$ and $\mathbf{p}_j$. Explain your results.

Chapter 7

Spline Curves in Bézier Form

Bézier curves provide a powerful tool in curve design, but they have some limitations: if the curve to be modeled has a complex shape, then its Bézier representation will have a prohibitively high degree (for practical purposes, degrees exceeding 10 are prohibitive). Such complex curves can, however, be modeled using *composite Bézier curves*. We shall also use the name *B-spline curves* for such *piecewise polynomial curves*. This chapter describes the main properties of cubic and quadratic spline curves. More general spline curves will be presented in Chapter 10.

7.1 Global and Local Parameters

Before we start to develop a theory for piecewise curves, let us establish the main definitions that we will use. When we considered single Bézier curves, we assumed that they were the map of the interval $0 \leq t \leq 1$. We could make this assumption because of the invariance of Bézier curves under affine parameter transformations; see Section 3.3. Life is not quite that easy with piecewise curves: while we can assume that each individual segment of a spline curve $\mathbf{s}$ is the map of the interval $[0, 1]$, the curve as a whole is the map of a collection of intervals, and their relative lengths play an important role.

A *spline curve* $\mathbf{s}$ is the continuous map of a collection of intervals $u_0 < \ldots < u_L$ into $\mathbb{E}^3$, where each interval $[u_i, u_{i+1}]$ is mapped onto a polynomial curve segment. Each real number u_i is called a *breakpoint* or a *knot*. The collection of all u_i is called the *knot sequence*. For every parameter value u, we thus have a corresponding point $\mathbf{s}(u)$ on the curve $\mathbf{s}$. Let this value u be from an interval $[u_i, u_{i+1}]$. We can introduce a *local coordinate* (or local parameter) t for the interval $[u_i, u_{i+1}]$ by setting

$$t = \frac{u - u_i}{u_{i+1} - u_i} = \frac{u - u_i}{\Delta_i}. \tag{7.1}$$

One checks that t varies from 0 to 1 as u varies from u_i to u_{i+1}.

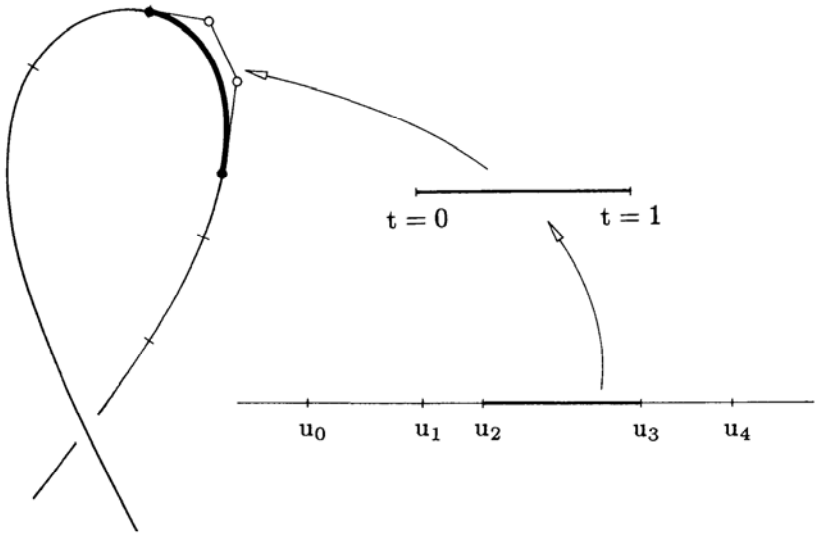

Figure 7.1: Local coordinates: the interval $[u_2, u_3]$ has been endowed with a local coordinate t. The third segment of the spline curve is shown with its Bézier polygon.

When we investigate properties of the curve $\mathbf{s}$, it will be more convenient to do so in terms of the global parameter u. (An example of such a property is the concept of differentiability.) The individual segments of $\mathbf{s}$ may be written as Bézier curves, and it is often easier to describe each one of them in terms of local coordinates. We adopt the definition $\mathbf{s}_i$ for the i^{th} segment of $\mathbf{s}$, and we write $\mathbf{s}(u) = \mathbf{s}_i(t)$ to denote a point on it. Figure 7.1 illustrates the interplay between local and global coordinates.

The introduction of local coordinates has some ramifications concerning the use of derivatives. For $u \in [u_i, u_{i+1}]$, the chain rule gives

$$\frac{\mathrm{d}\mathbf{s}(u)}{\mathrm{d}u} = \frac{\mathrm{d}\mathbf{s}_i(t)}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}u} \tag{7.2}$$

$$= \frac{1}{\Delta_i}\frac{\mathrm{d}\mathbf{s}_i(t)}{\mathrm{d}t}. \tag{7.3}$$

Two more definitions: the points $\mathbf{s}(u_i) = \mathbf{s}_i(0) = \mathbf{s}_{i-1}(1)$ are called *junction points* or *joints*. The collection of the Bézier polygons for all curve segments itself forms a polygon; it is called the *piecewise Bézier polygon* of $\mathbf{s}$.

7.2 Smoothness Conditions

Suppose we are given two Bézier curves $\mathbf{s}_0$ and $\mathbf{s}_1$, with polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$, respectively. We may think of each curve as existing by itself, defined over the interval $t \in [0, 1]$ or some other interval. We may also think of the two

curves as two segments of one *composite* curve, defined as the map of the interval $[u_0, u_2]$ into $\mathbb{E}^3$. The "left" segment $\mathbf{s}_0$ is defined over an interval $[u_0, u_1]$, while the "right" segment $\mathbf{s}_1$ is defined over $[u_1, u_2]$ (see Section 7.1).

Let us pretend for a moment that both curves are arcs of one global polynomial curve $\mathbf{b}^n(u)$, defined over the interval $[u_0, u_2]$. Section 4.6 tells us that the two polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$ must be the result of a subdivision process. Then their control vertices must be related by

$$\mathbf{b}_{n+i} = \mathbf{b}^i_{n-i}(t); \quad i = 0, \ldots, n, \tag{7.4}$$

where $t = (u_2 - u_0)/(u_1 - u_0)$ is the local coordinate of u_2 with respect to the interval $[u_0, u_1]$.

Now suppose we arbitrarily change $\mathbf{b}_{2n}$; the two curves then no longer describe the same global polynomial. However, they still agree in all derivatives of order $0, \ldots, n-1$ at $u = u_1$! This is simply because $\mathbf{b}_{2n}$ has no influence on derivatives of order less than n at $u = u_1$. Similarly, we may change $\mathbf{b}_{2n-r}$ and still maintain continuity of all derivatives of order $0, \ldots, n-r-1$.

We therefore have the C^r condition for Bézier curves: the two Bézier curves defined over $u_0 \le u \le u_1$ and $u_1 \le u \le u_2$, by the polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$, respectively, are r times continuously differentiable at $u = u_1$ if and only if

$$\mathbf{b}_{n+i} = \mathbf{b}^i_{n-i}(t); \quad i = 0, \ldots, r, \tag{7.5}$$

where $t = (u_2 - u_0)/(u_1 - u_0)$ is the local coordinate of u_2 with respect to the interval $[u_0, u_1]$. See Example 7.1 for a specific case.

Suppose the curve from Example 3.1 is defined over [0,1]. What are the Bézier points of a second Bézier curve, defined over [1,3] and with a C^2 join to the first curve? We have to evaluate at $t = 3$:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} 8 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 24 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 72 \\ -6 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{4} \\ \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{-4} \\ \mathbf{-4} \end{bmatrix} \quad \begin{bmatrix} \mathbf{-60} \\ \mathbf{-18} \end{bmatrix}.$$

The boldface points are the desired ones. If they are the first three Bézier points of the "right" curve, both curves will be C^2 over the interval [0,3].

Example 7.1: Computing the C^2 extension of a Bézier curve.

Thus the de Casteljau algorithm also governs the continuity conditions between adjacent Bézier curves. Note that (7.5) is a theoretical tool; it should not be used to *construct* C^r curves—this would lead to numerical problems because of the extrapolations that are used in (7.5).

Another condition for C^r continuity should also be mentioned here. By equating derivatives using (4.20) and applying the chain rule,[1] we obtain

$$\left(\frac{1}{\Delta_0}\right)^i \Delta^i \mathbf{b}_{n-i} = \left(\frac{1}{\Delta_1}\right)^i \Delta^i \mathbf{b}_n \quad i = 0, \ldots, r. \tag{7.6}$$

Conditions for continuity of higher derivatives of Bézier curves were first derived by E. Staerk [478] in 1976. The cases $r = 1$ and $r = 2$ are probably the ones of most practical relevance, and we shall discuss them in more detail next.

7.3 C^1 and C^2 Continuity

We know that only the three Bézier points $\mathbf{b}_{n-1}$, $\mathbf{b}_n$, $\mathbf{b}_{n+1}$ influence the first derivatives at the junction point $\mathbf{b}_n$. According to (7.5), $\mathbf{b}_{n+1}$ is obtained by linear interpolation of the two points $\mathbf{b}_{n-1}$, $\mathbf{b}_n$. These three points must therefore be collinear and must also be in the ratio $(u_1 - u_0) : (u_2 - u_1) = \Delta_0 : \Delta_1$. This C^1 condition is illustrated in Figure 7.2.

It is important to note that collinearity of three distinct control points $\mathbf{b}_{n-1}$, $\mathbf{b}_n$, $\mathbf{b}_{n+1}$ is not sufficient to guarantee C^1 continuity! This is because the notion of C^1 continuity is based on the interplay between domain and range configurations. Collinearity of three points is purely a range phenomenon. Without additional information on the domain of the curve under consideration, we cannot make any statements concerning differentiability. However, collinearity of three distinct control points $\mathbf{b}_{n-1}$, $\mathbf{b}_n$, $\mathbf{b}_{n+1}$ does guarantee a continuously varying tangent line.

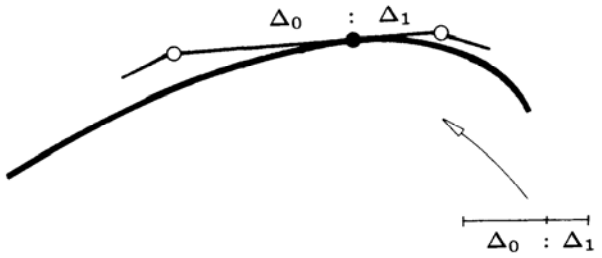

Figure 7.2: C^1 condition: the three shown Bézier points must be collinear with ratio $\Delta_0 : \Delta_1$.

[1] Equation (4.20) is with respect to the local parameter of an interval. We are interested in differentiability with respect to the global parameter.

A special situation arises if $\Delta \mathbf{b}_{n-1} = \Delta \mathbf{b}_n = \mathbf{0}$, i.e., if all three points $\mathbf{b}_{n-1}, \mathbf{b}_n, \mathbf{b}_{n+1}$ coincide. In this case, the composite curve $\mathbf{s}$ has a zero tangent vector at the junction point $\mathbf{b}_n$ and is differentiable regardless of the interval lengths Δ_0, Δ_1. Zero tangent vectors may give rise to corners or cusps in curves, a fact that intuitively contradicts the concept of differentiability.

Smoothness and differentiability only agree for functional curves—the connection between them is lost in the parametric case. Differentiable curves may not be smooth (see cusps above) and smooth curves may not be differentiable (see Figures 7.6 and 7.7).

Moving on to C^2 continuity, let us now assume that $\mathbf{s}$ is C^1, so that (7.5) and (7.6) are satisfied for $r = 1$. The additional C^2 condition, with $r = 2$ in (7.5), states that the two quadratic polynomials with control polygons $\mathbf{b}_{n-2}, \mathbf{b}_{n-1}, \mathbf{b}_n$ and $\mathbf{b}_n, \mathbf{b}_{n+1}, \mathbf{b}_{n+2}$, defined over $[u_0, u_1]$ and $[u_1, u_2]$, describe the same global quadratic polynomial. Therefore, a polygon $\mathbf{b}_{n-2}, \mathbf{d}, \mathbf{b}_{n+2}$ must exist that describes that polynomial over the interval $[u_0, u_2]$. The two subpolygons are then obtained from it by subdivision at the parameter value u_1.

A C^2 condition for a C^1 curve $\mathbf{s}$ at u_1 is thus the existence of a point $\mathbf{d}$ such that

$$\mathbf{b}_{n-1} = (1 - t_1)\mathbf{b}_{n-2} + t_1\mathbf{d}, \tag{7.7}$$

$$\mathbf{b}_{n+1} = (1 - t_1)\mathbf{d} + t_1\mathbf{b}_{n+2}, \tag{7.8}$$

where $t_1 = \Delta_0/(u_2 - u_0)$ is the local parameter of u_1 with respect to the interval $[u_0, u_2]$. Figure 7.3 gives an example.

This condition provides us with an easy test to see if a curve is C^2 at a given breakpoint u_i: we simply construct auxiliary points $\mathbf{d}_-, \mathbf{d}_+$ from both the right and the left and check for equality. Figure 7.4 shows two curve segments that fail the C^2 test.

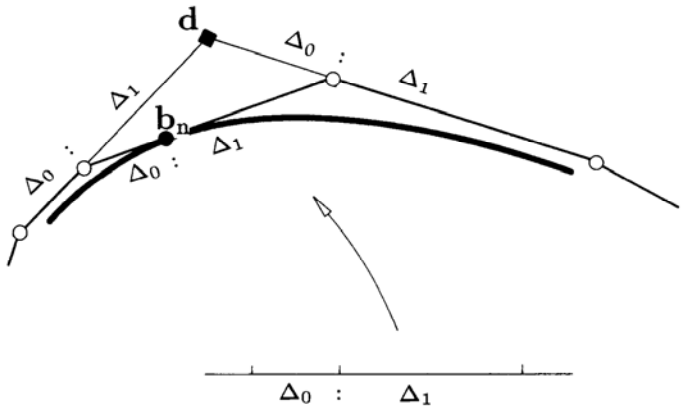

Figure 7.3: C^2 condition: two Bézier curves are twice differentiable at the junction point $\mathbf{b}_n$ if the auxiliary point $\mathbf{d}$ exists uniquely.

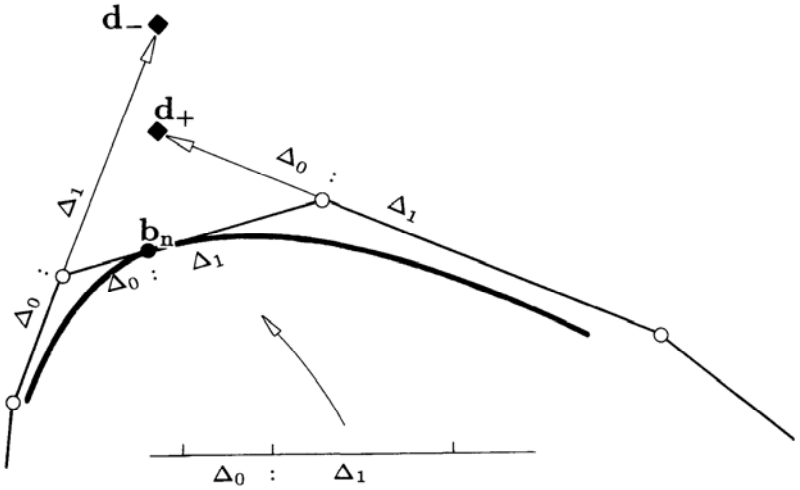

Figure 7.4: C^2 condition: the two segments shown generate different auxiliary points $\mathbf{d}_{\pm}$; hence they are only C^1.

Another derivation of the C^2 condition would be to compute the left and right second derivatives at the junction point $\mathbf{b}_n$ and to equate them. The second derivatives at a junction point are essentially second differences of nearby Bézier points. For the simpler case of uniform parameter spacing, $\Delta_0 = \Delta_1$, Figure 7.5 shows how this approach leads to the same C^2 condition as before.

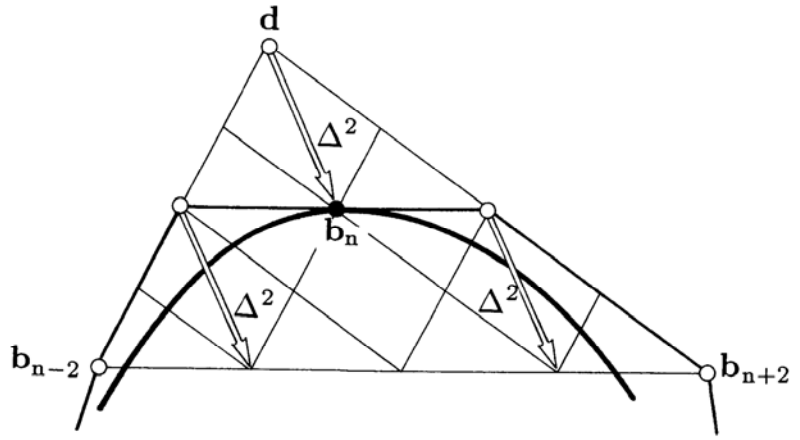

Figure 7.5: C^2 condition for uniform parameter spacing: if $\Delta^2\mathbf{b}_{n-2} = \Delta^2\mathbf{b}_n = \Delta^2$, a unique auxiliary point $\mathbf{d}$ exists. (Proof by the use of similar triangles.)

While both ways of checking C^2 continuity are mathematically equivalent, the first one is more practical: it compares *points* ($\mathbf{d}_-$ and $\mathbf{d}_+$), while the second one compares *vectors* (left and right second derivatives). One usually has a point tolerance[2] present in an application, but it would be hard to define a tolerance according to which two second derivative vectors can be labeled equal. The problem of checking for C^2 continuity arises when a piecewise cubic curve is given and one tries to convert it to B-spline format, see Section 7.6.

7.4 Finding a C^1 Parametrization

Suppose we are given a piecewise Bézier curve. A probable question would now be: "Is this curve C^1?" This question is meaningless! The concept of C^1 continuity is based upon an interplay between the knot sequence and the control polygons of the curve. Hence a meaningful question would be: "Can we find a knot sequence such that the curve is C^1 with respect to it?"

We can determine such a knot sequence from inspection of the piecewise Bézier polygon: if it has "corners" at the junction points, it cannot define a C^1 spline curve, and the notion of a knot sequence is meaningless. (A C^0 spline curve is C^0 over *any* knot sequence.) Suppose then that we have a piecewise Bézier polygon with $\mathbf{b}_{in-1}$, $\mathbf{b}_{in}$, $\mathbf{b}_{in+1}$ collinear for all i. We can now construct a knot sequence as follows: set $u_0 = 0$, $u_1 = 1$ and for $i = 2, \ldots, L$ determine u_i by solving

$$\frac{\Delta_{i-1}}{\Delta_i} = \frac{||\Delta \mathbf{b}_{in-1}||}{||\Delta \mathbf{b}_{in}||} \tag{7.9}$$

for Δ_i. If desired, we may now normalize the u_i by dividing through by u_L. This forces all u_i to be in the unit interval $[0, 1]$. Of course, any scaling or translation of the knot sequence is allowed: our C^1 conditions are invariant under affine parameter transformations!

Any other choice of parameter intervals will not change the shape of the piecewise curve—that shape is uniquely determined by the Bézier polygons. However, different knot spacing *will* change the continuity class of the curve defined by its Bézier polygons; the cross plots that are shown in Figures 7.6 and 7.7 demonstrate this. We see that the continuity class of a curve is not a geometric property that is intrinsically linked to the shape of the curve—it is a result of the parametrization.

7.5 C^1 Quadratic B-spline Curves

Let us consider a C^1 piecewise quadratic spline curve $\mathbf{s}$ that is defined over L intervals $u_0 < \ldots < u_L$, as in Figure 7.8. We call the Bézier points $\mathbf{b}_{2i+1}$ *inner Bézier points*, and the $\mathbf{b}_{2i}$ *junction points*.

[2] If two points are closer together than this tolerance, they are regarded as equal.

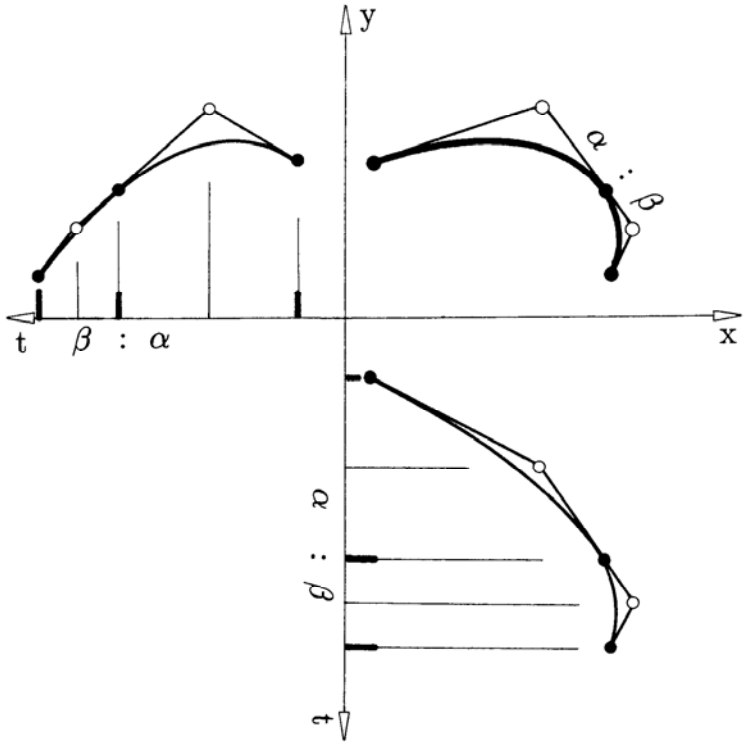

Figure 7.6: A C^1 parametrization: the piecewise quadratic Bézier curve is C^1 when the parameter intervals are chosen to be in the same ratio $\alpha : \beta$ as the Bézier points $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$.

We can completely determine a quadratic spline curve by prescribing the knot sequence and the Bézier points

$$\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_3, \ldots, \mathbf{b}_{2i+1}, \ldots, \mathbf{b}_{2L-1}, \mathbf{b}_{2L}.$$

The remaining junction points are computed from the C^1 conditions

$$\mathbf{b}_{2i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i}\mathbf{b}_{2i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i}\mathbf{b}_{2i+1}; \quad i = 1, \ldots, L-1. \tag{7.10}$$

We can thus define a C^1 quadratic Bézier curve with fewer data than are necessary to define the complete piecewise Bézier polygon. The minimum amount of information that is needed is (1) the polygon $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_3, \ldots, \mathbf{b}_{2i+1} \ldots, \mathbf{b}_{2L-1}, \mathbf{b}_{2L}$, called the *B-spline polygon* or *de Boor polygon* of $\mathbf{s}$, and (2) the knot sequence $u_0, \ldots, u_L$.[3] If the curve is described in termsof this B-spline polygon, it is sometimes called a *B-spline*

[3]For readers familiar with the IGES definition of B-splines: there, the knots u_0 and u_L would have to be listed three times each.

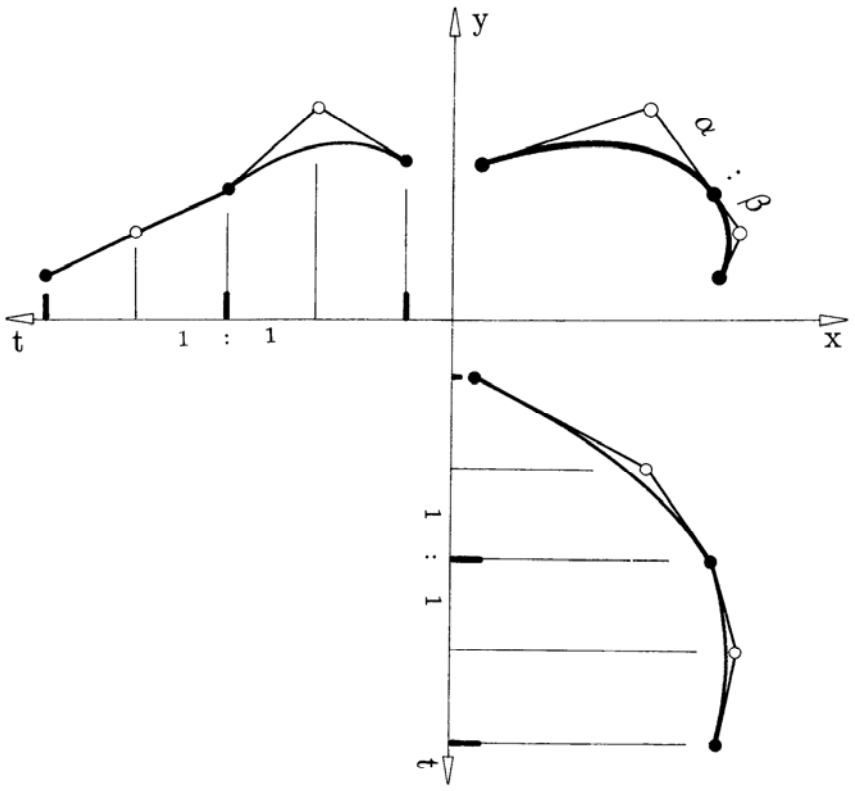

Figure 7.7: A C^0 parametrization: the piecewise Bézier curve is the same as in the previous figure. It is *not* a C^1 curve with the choice of uniform parameter intervals as indicated in the cross plot.

curve. We also denote the quadratic B-spline polygon by $\mathbf{d}_{-1}, \mathbf{d}_0, \ldots, \mathbf{d}_{L-1}, \mathbf{d}_L$; see Figure 7.8. Each B-spline polygon, together with a knot sequence, determines a C^1 quadratic spline curve, and, conversely, each quadratic C^1 spline curve possesses a unique B-spline polygon.[4]

From the definition of a quadratic B-spline polygon, we can deduce several properties, which we shall simply list since their derivation is a direct consequence of the previous definitions:

- Convex hull property
- Linear precision
- Affine invariance

[4]The numbering of knots and control points here is strictly aimed at the quadratic case. A different numbering scheme is employed in Chapter 10 for more general configurations.

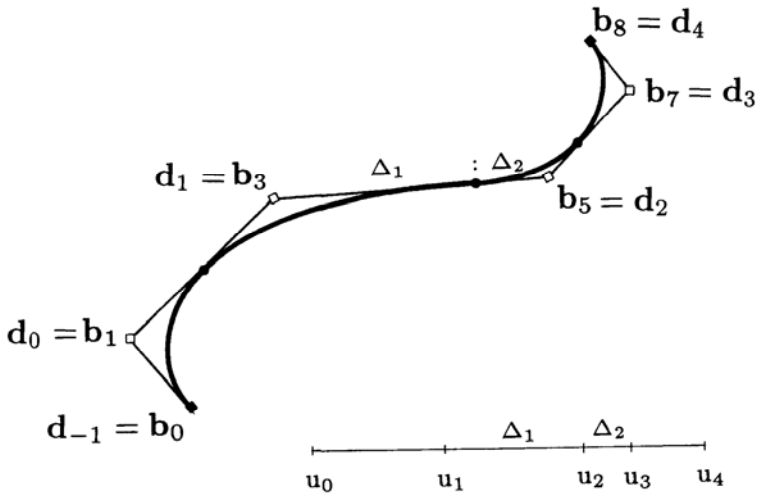

Figure 7.8: C^1 quadratic splines: the junction points $\mathbf{b}_{2i}$ are determined by the inner Bézier points and the knot sequence.

- Symmetry
- Endpoint interpolation
- Variation diminishing property.

The last property follows because the piecewise Bézier polygon of $\mathbf{s}$ is obtained by piecewise linear interpolation of the B-spline polygon, a process that is variation diminishing, as seen in Section 2.4.

All of the preceding properties are shared with Bézier curves, although the convex hull property may be sharpened considerably for quadratic B-spline curves: the curve $\mathbf{s}$ lies in the union of the convex hulls of the triangles $\mathbf{b}_{2i-1}$, $\mathbf{b}_{2i+1}$, $\mathbf{b}_{2i+3}$; $i = 1 \ldots, L-2$ and the triangles $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_3$ and $\mathbf{b}_{2L-3}$, $\mathbf{b}_{2L-1}$, $\mathbf{b}_{2L}$, as shown in Figure 7.9. One Bézier curve, on the other hand, could only be guaranteed to lie within the convex hull of its whole control polygon.

By definition, quadratic B-spline curves consist of parabolic segments, i.e., planar curves. However, the B-spline control polygon may be truly three-dimensional—we thus have a method to generate C^1 space curves that are piecewise planar.

One important property that single Bézier curves do not share with B-spline curves is *local control*. If we are dealing with a single Bézier curve,[5] we know that a change of one of the control vertices affects the whole curve—it is a *global* change. Changing a control vertex of a quadratic B-spline curve, on the other hand, affects at most three curve segments. It is this local control property that made B-spline

[5] In this context, we do not consider Bézier curves as parts of composite curves!

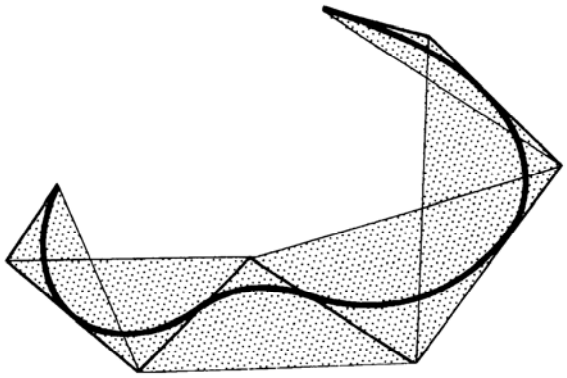

Figure 7.9: The convex hull property: a C^1 quadratic B-spline curve lies in the union of a set of triangles. The triangles are formed by triples of consecutive control vertices.

curves as popular as they are. If a part of a curve is completely designed, it is highly undesirable to jeopardize this result by changing the curve in other regions. With single Bézier curves, this is unavoidable.

As a consequence of the local control property, we may include straight line segments in a quadratic B-spline curve: if three subsequent control vertices are collinear, the quadratic segment that is determined by them must be linear. A single (higher degree) Bézier curve cannot contain linear segments unless it is itself linear; this is yet another reason why B-spline curves are much more flexible than single Bézier curves. Figure 7.10 shows a quadratic B-spline curve that includes straight line segments. Such curves occur frequently in technical design applications, as well as in font design.

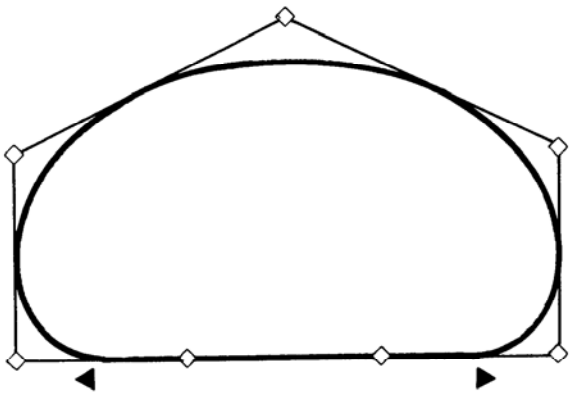

Figure 7.10: Quadratic B-spline curves: curves can be designed that include straight line segments.

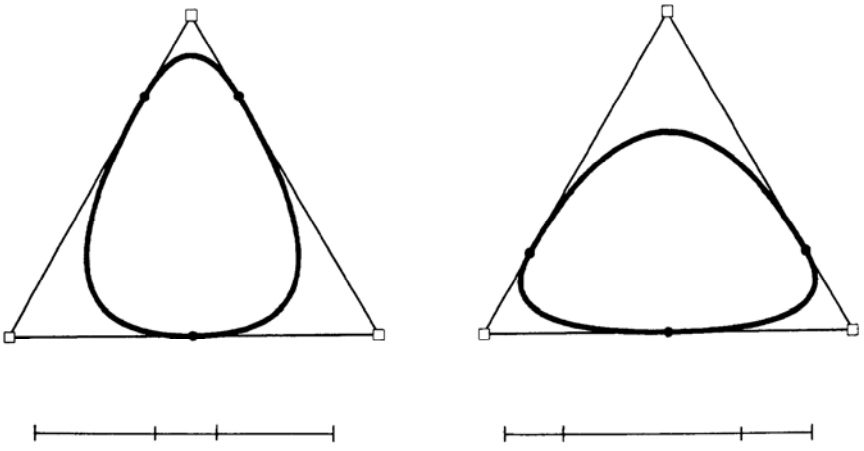

Figure 7.11: Closed curves: two closed quadratic B-spline curves are shown that have the same control polygon but different knot sequences.

From inspection of Figure 7.8, we see that the endpoints of a B-spline curve are treated in a special way. This is not the case with *closed curves*. Closed curves are defined by $\mathbf{s}(u_0) = \mathbf{s}(u_L)$. Figure 7.11 shows two closed quadratic B-spline curves. For such curves, C^1 continuity is defined by the additional constraint $(\mathrm{d}/\mathrm{d}u)\mathbf{s}(u_0) = (\mathrm{d}/\mathrm{d}u)\mathbf{s}(u_L)$.

The figure also shows that a B-spline curve depends not only on the B-spline polygon, but also on the knot sequence.

7.6 C^2 Cubic B-spline Curves

We are now interested in C^2 piecewise cubic spline curves, again defined over L intervals $u_0 < \ldots < u_L$. Consider any two adjacent curve segments $\mathbf{s}_{i-1}$ and $\mathbf{s}_i$. To be C^1 at u_i, the relevant Bézier points must be in the ratio $\Delta_{i-1} : \Delta_i$, or

$$\mathbf{b}_{3i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i}\mathbf{b}_{3i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i}\mathbf{b}_{3i+1}. \tag{7.11}$$

To be C^2 as well, an auxiliary point $\mathbf{d}_i$ must exist such that the points $\mathbf{b}_{3i-2}$, $\mathbf{b}_{3i-1}$, $\mathbf{d}_i$ and $\mathbf{d}_i$, $\mathbf{b}_{3i+1}$, $\mathbf{b}_{3i+2}$ are in the same ratio $\Delta_{i-1} : \Delta_i$, as follows from the C^2 conditions (7.8). Figure 7.12 illustrates this point.

A C^2 cubic spline curve defines the auxiliary points $\mathbf{d}_i$, which form a polygon $\mathbf{P}$. Conversely, a polygon $\mathbf{P}$ and a knot sequence $\{u_i\}$ also define a C^2 cubic spline curve. Set

$$\mathbf{b}_{3i-2} = \frac{\Delta_{i-1} + \Delta_i}{\Delta}\mathbf{d}_{i-1} + \frac{\Delta_{i-2}}{\Delta}\mathbf{d}_i, \tag{7.12}$$

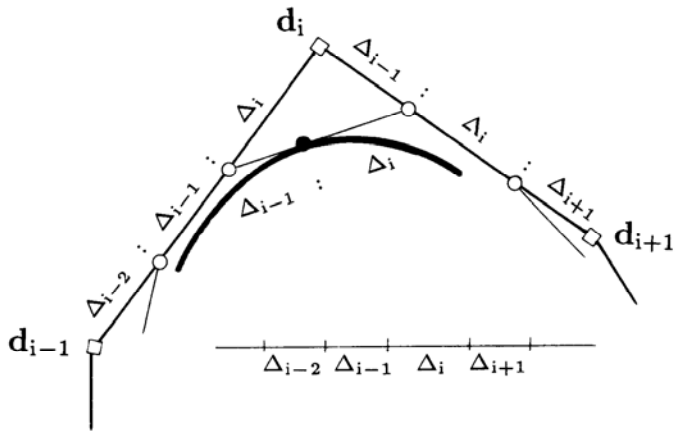

Figure 7.12: C^2 cubic B-spline curves: the auxiliary points $\mathbf{d}_i$ define the B-spline polygon of the curve.

$$\mathbf{b}_{3i-1} = \frac{\Delta_i}{\Delta}\mathbf{d}_{i-1} + \frac{\Delta_{i-2} + \Delta_{i-1}}{\Delta}\mathbf{d}_i \tag{7.13}$$

for $i = 2, L-1$, where

$$\Delta = \Delta_{i-2} + \Delta_{i-1} + \Delta_i. \tag{7.14}$$

With the junction points $\mathbf{b}_{3i}$ defined in (7.11), the piecewise Bézier curve defined by the $\mathbf{d}_i$ meets the C^2 conditions at every knot u_i.

Near the ends, things are a little more complicated. We define the cubic B-spline polygon to have vertices $\mathbf{d}_{-1}, \mathbf{d}_0, \ldots, \mathbf{d}_L, \mathbf{d}_{L+1}$ and then set

$$\begin{aligned} \mathbf{b}_0 &= \mathbf{d}_{-1}, \\ \mathbf{b}_1 &= \mathbf{d}_0, \\ \mathbf{b}_2 &= \tfrac{\Delta_1}{\Delta_0+\Delta_1}\mathbf{d}_0 + \tfrac{\Delta_0}{\Delta_0+\Delta_1}\mathbf{d}_1, \end{aligned} \tag{7.15}$$

$$\begin{aligned} \mathbf{b}_{3L-2} &= \tfrac{\Delta_{L-1}}{\Delta_{L-2}+\Delta_{L-1}}\mathbf{d}_{L-1} + \tfrac{\Delta_{L-2}}{\Delta_{L-2}+\Delta_{L-1}}\mathbf{d}_L, \\ \mathbf{b}_{3L-1} &= \mathbf{d}_L, \\ \mathbf{b}_{3L} &= \mathbf{d}_{L+1}. \end{aligned} \tag{7.16}$$

Now the spline curve is C^2 at every interior knot. This construction is due to W. Boehm [61]. An illustration is given in Figure 7.13.

If a cubic spline curve is expressed in terms of the *B-spline polygon* (the polygon consisting of the $\mathbf{d}_i$), it is usually called a C^2 cubic B-spline curve.

Cubic B-spline curves enjoy the same properties as do quadratic ones:

- Convex hull property
- Linear precision

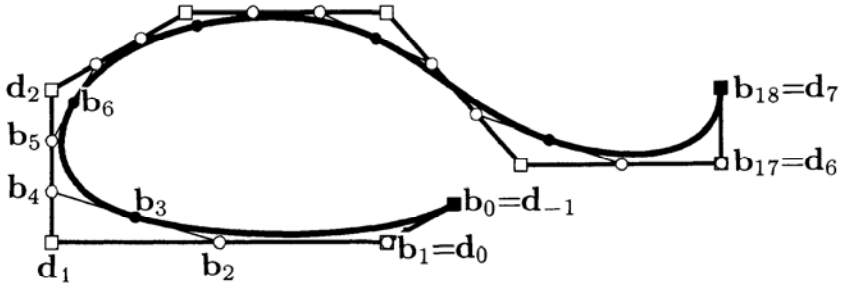

Figure 7.13: B-splines: a cubic B-spline curve with its control polygon.

- Affine invariance
- Symmetry
- Endpoint interpolation
- Variation diminishing property
- Local control.

Local control for cubic B-spline curves is not quite as local as it is for quadratic ones. If a control vertex $\mathbf{d}_i$ of a cubic B-spline curve is moved, *four* segments of the curve will be changed, as shown in Figure 7.14.

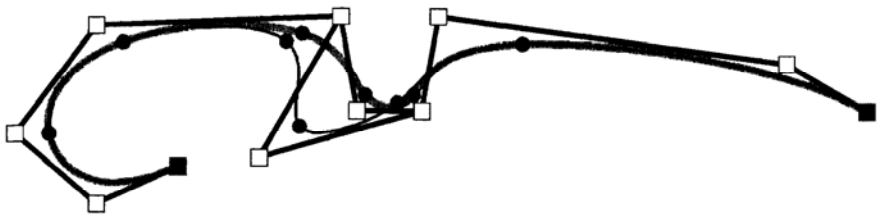

Figure 7.14: Local control: as one control vertex is moved, only the four "nearby" curve segments change.

7.7 Finding a Knot Sequence

"Given the de Boor polygon and the knot sequence, construct the corresponding piecewise Bézier polygon" was the topic of the last two sections. In freeform design, one creates the de Boor polygon interactively, but how does one create the knot sequence? An easy answer is to set $u_i = i$, or some other (equivalent) uniform spacing, but this method is too rigid in many cases. The jury is still out on what constitutes an "optimal" parametrization. As a rule of thumb, better curves are obtained from a given polygon if the geometry of the polygon is incorporated into the knot sequence.

For example, one may set (in the cubic case)

$$\begin{aligned} u_0 &= 0, \\ u_1 &= ||\mathbf{d}_1 - \mathbf{d}_{-1}||, \\ u_i &= u_{i-1} + ||\mathbf{d}_i - \mathbf{d}_{i-1}||; \quad i = 2, \ldots, L-1, \\ u_L &= u_{L-1} + ||\mathbf{d}_{L+1} - \mathbf{d}_{L-1}||. \end{aligned} \tag{7.17}$$

This is a *chord length parametrization* for cubic B-spline curves when the polygon is given.[6] This parametrization often produces "smoother" curves than the uniform one described above (see Sapidis [440]).

7.8 Design and Inverse Design

Quadratic B-spline curves seemed to do a pretty good job of producing complex shapes, so why increase the degree to cubic? Cubic polynomials are *true space curves*, i.e., they are not planar. For 2D shapes, piecewise quadratics might suffice, but when it comes to 3D, they can only produce piecewise 2D segments. Two examples why this is not desirable: 3D curves that are used to describe robot paths will exhibit jumps in their torsion—this is bad for the joints of the robot arm. Secondly, 3D curves that have to satisfy aesthetic requirements would simply *look* bad if described by piecewise planar shapes.[7]

Another advantage of piecewise cubics is the fact that they may have inflection points *inside* a segment. With piecewise quadratics, one would have to make sure that there is a junction point at every inflection point.

How do we design curves using cubic B-splines? The typical freeform design takes place in a 2D environment, with the use of a mouse or some other interactive input device. A control polygon is sketched on the terminal, the resulting curve is drawn, the control polygon is adjusted, and so on. The parametrization that is being used should be kept away from the designer, and would most likely be one of the two methods described in the previous section. To obtain a 3D curve, one would then change to another view (by rotating the curve) and continue adjusting control points.

[6] Another chord length parametrization exists if data points are given for an interpolatory spline, as described in Chapter 9.

[7] Of course, this could be overcome by using a large number of quadratic pieces. But then, why not go a step further and do everything piecewise linear, with even more pieces?

The final result might look reasonable on the terminal, but should probably undergo a final smoothing process as discussed in Chapter 23.

Sometimes designers do not like to deal with control polygons and prefer to manipulate the curve directly. In that case, the curve should be obtained from an interpolation process as described in Chapters 8 or 9. It may be represented internally in B-spline or piecewise Bézier form. If the designer wants to change a certain junction point $\mathbf{x}_i$ to a new location, this may easily be done *locally* using B-spline technology as follows.

Displacing $\mathbf{x}(u_i)$ by a vector $\mathbf{v}_i$ would require a displacement of $\mathbf{d}_i$ to a new location $\mathbf{d}_i + \mathbf{e}_i$. Clearly $\mathbf{e}_i$ must be parallel to $\mathbf{v}_i$, and so we can state

$$\mathbf{e}_i = \frac{1}{c_i}\mathbf{v}_i. \tag{7.18}$$

The value of c_i may be computed to

$$c_i = \frac{1}{u_{i+1} - u_{i-1}} \left(\Delta_i \frac{u_i - u_{i-2}}{u_{i+1} - u_{i-2}} + \Delta_{i-1} \frac{u_{i+2} - u_i}{u_{i+2} - u_{i-1}} \right). \tag{7.19}$$

Thus we have solved our problem, called *inverse design*. In this mode, we would directly move the junction point $\mathbf{x}_i$ and simply hide the equivalent change in the control polygon from the designer. We need to keep in mind that this procedure (since it changes $\mathbf{d}_i$) will also change $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$, although by smaller amounts. Note that we can interpret Fig. 7.14 as an illustration of inverse design!

7.9 Implementation

The following is a program for the conversion of a cubic B-spline curve to piecewise Bézier form:

```
void bspline_to_bezier(bspl,knot,l,bez)
/*   converts  cubic B-spline polygon into piecewise Bezier polygon.
Input:   bspl: B-spline control polygon
         knot: knot sequence
         l:    no. of intervals
Output:  bez:  piecewise Bezier polygon. Each junction point b_3i is
               only stored once.
Remark:  bspl starts from 0 and not -1 as in the text. All
         subscripts are therefore shifted by one. For those familiar
         with Chapter 10: don't try to use multiple knots here --
         in terms of that chapter, the end knots u_0 and u_l
         have multiplicity 3, but  all other
         knots are simple, and the curve is C2.
*/
```

This routine has to be called for each coordinate (i.e., two or three times). Speedups are therefore possible.

7.10 Exercises

1. If we write the de Casteljau algorithm in the form of a triangular array as in (3.3), subdivision tells us how the three "sides" of that array are related to each other. Write explicitly how to generate the elements of one side from those of any other one.
2. Describe the chord length parametrization for closed B-spline curves.

*3. Consider two Bézier curves with polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$. Let $\mathbf{b}_{n-r} = \ldots = \mathbf{b}_n = \ldots \mathbf{b}_{n+r}$ so that both curves form one (degenerate) C^r curve. Under what conditions on $\mathbf{b}_{n-r-1}$ and $\mathbf{b}_{n+r+1}$ is that curve also C^{r+1}?

*4. We are given a closed polygon. Suppose we want to make the polygon vertices the inner Bézier points $\mathbf{b}_{2i+1}$ of a piecewise quadratic and that we pick arbitrary points on the polygon legs to become the junction Bézier points $\mathbf{b}_{2i}$. Can we always find a C^1 parametrization for this (tangent continuous) piecewise quadratic curve?

P1. Design a helix-like C^2 cubic B-spline curve. Then plot the blossom $\mathbf{b}[t, t, s]$ for each cubic piece, for the range $0 \leq t \leq 1$ and $-0.5 \leq s \leq 0.5$ (assuming that t is the local parameter for each piece). You should obtain a surface. Discuss its properties.

Chapter 8

Piecewise Cubic Interpolation

Polynomial interpolation is a fundamental theoretical tool, but for practical purposes, better methods exist. The most popular class of methods is that of piecewise polynomial schemes. All these methods construct curves that consist of polynomial pieces of the same degree and that are of a prescribed overall smoothness. The given data are usually points and parameter values; sometimes, tangent information is added as well.

In practice, one usually encounters the use of piecewise cubic curves. They may be C^2—the next chapter on cubic spline interpolation is dedicated to that case. If they are only C^1, the trade-off for the lower differentiability class is *locality*: if a data point is changed, the interpolating curve only changes in the vicinity of that data point. We call this class of interpolants *piecewise cubic interpolants*.

This chapter can only cover the basic ideas behind piecewise cubic interpolation. A large variety of interpolation methods exist that are designed to cope with special problems. Most such methods try to preserve shape features inherent in the given data, for example, convexity or monotonicity. We mention the work by Fritsch and Carlson [222], McLaughlin [355], Foley [209], [210], McAllister and Roulier [352], and Schumaker [453].

8.1 C^1 Piecewise Cubic Hermite Interpolation

This is conceptually the simplest of all C^1 interpolants, although not the most practical one. It solves the following problem:

Given: Data points $\mathbf{x}_0, \ldots, \mathbf{x}_L$, corresponding parameter values $u_0, \ldots, u_L$, and corresponding tangent vectors $\mathbf{m}_0, \ldots, \mathbf{m}_L$.

Find: A C^1 piecewise cubic polynomial $\mathbf{s}$ that interpolates to the given data, i.e.,

$$\mathbf{s}(u_i) = \mathbf{x}_i, \quad \frac{\mathrm{d}}{\mathrm{d}u}\mathbf{s}(u_i) = \mathbf{m}_i; \quad i = 0, \ldots, L. \tag{8.1}$$

We construct the solution as a piecewise Bézier curve, as illustrated in Figure 8.1. We find the junction Bézier points immediately: $\mathbf{b}_{3i} = \mathbf{x}_i$. To obtain the inner Bézier points, we recall the derivative formula for Bézier curves from Section 7.3:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}u}\mathbf{s}(u_i) &= \frac{3}{\Delta_{i-1}}(\mathbf{b}_{3i} - \mathbf{b}_{3i-1}) \\ &= \frac{3}{\Delta_i}(\mathbf{b}_{3i+1} - \mathbf{b}_{3i}),\end{aligned}$$

where $\Delta_i = \Delta u_i$. Thus the inner Bézier points $\mathbf{b}_{3i+1}; i = 0, \ldots, L-1$ are given by

$$\mathbf{b}_{3i+1} = \mathbf{b}_{3i} + \frac{\Delta_i}{3}\mathbf{m}_i, \tag{8.2}$$

and the inner Bézier points $\mathbf{b}_{3i-1}$, $i = 1, \ldots, L$ are

$$\mathbf{b}_{3i-1} = \mathbf{b}_{3i} - \frac{\Delta_{i-1}}{3}\mathbf{m}_i. \tag{8.3}$$

What we have done so far is construct the piecewise Bézier form of the C^1 piecewise cubic Hermite interpolant. Of course, we can utilize the material on cubic Hermite interpolation from Section 6.5 as well. Over the interval $[u_i, u_{i+1}]$, the

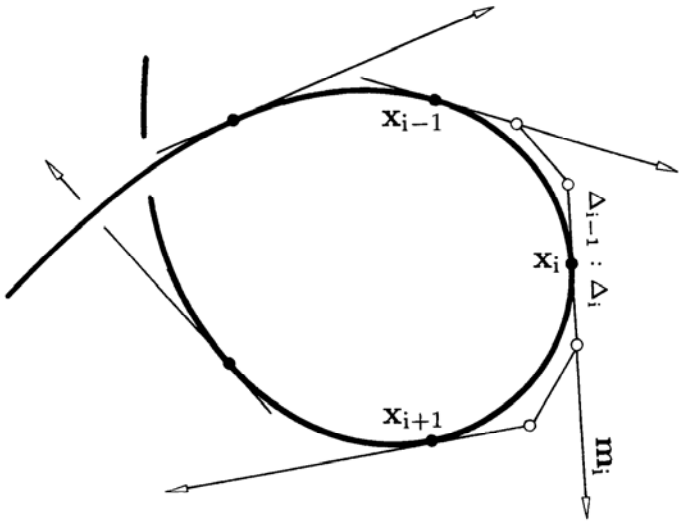

Figure 8.1: Piecewise cubic Hermite interpolation: the Bézier points are obtained directly from the data.

interpolant $\mathbf{s}$ can be expressed in terms of the cubic Hermite polynomials $\hat{H}_i^3(u)$ that were defined by (6.16). In the situation at hand, the definitions become:

$$\begin{aligned}
\hat{H}_0^3(u) &= B_0^3(t) + B_1^3(t), \\
\hat{H}_1^3(u) &= \frac{\Delta_i}{3} B_1^3(t), \\
\hat{H}_2^3(u) &= -\frac{\Delta_i}{3} B_2^3(t), \\
\hat{H}_3^3(u) &= B_2^3(t) + B_3^3(t),
\end{aligned} \tag{8.4}$$

where $t = (u - u_i)/\Delta_i$ is the local parameter of the interval $[u_i, u_{i+1}]$. The interpolant can now be written as

$$\mathbf{s}(u) = \mathbf{x}_i \hat{H}_0^3(u) + \mathbf{m}_i \hat{H}_1^3(u) + \mathbf{m}_{i+1} \hat{H}_2^3(u) + \mathbf{x}_{i+1} \hat{H}_3^3(u). \tag{8.5}$$

This interpolant is important for some theoretical developments; of more practical value are those developed in the following sections.

8.2 C^1 Piecewise Cubic Interpolation I

The title of this section is not very different from the one of the preceding section, and indeed the problems addressed in both sections differ only by a subtle nuance. Here, we try to solve the following problem:

Given: Data points $\mathbf{x}_0, \ldots, \mathbf{x}_L$ and tangent directions $\mathbf{l}_0, \ldots, \mathbf{l}_L$ at those data points.

Find: A C^1 piecewise cubic polynomial that passes through the given data points and is tangent to the given tangent directions there.

Comparing this problem to the one in the previous section, we find that this one is more vaguely formulated: the "Find" part does not contain a single formula. This reflects a typical practical situation: one is not always given parameter values u_i or tangent vectors $\mathbf{m}_i$; very often, the only available information is data points and tangent directions, as illustrated in Figure 8.2. It is important to note that we only

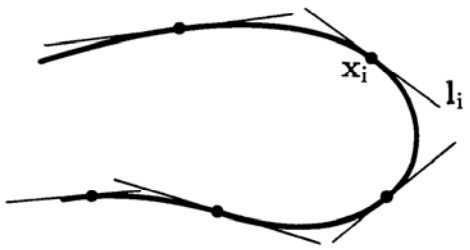

Figure 8.2: C^1 piecewise cubics: example data set.

have tangent *directions*, i.e., we have no vectors with a prescribed length. We can assume without loss of generality that the tangent directions $\mathbf{l}_i$ have been normalized to be of unit length:

$$||\mathbf{l}_i|| = 1.$$

The easiest step in finding the desired piecewise cubic is the same as before: the junction Bézier points $\mathbf{b}_{3i}$ are again given by $\mathbf{b}_{3i} = \mathbf{x}_i,\ i = 0, \ldots, L$.

For each inner Bézier point, we have a one-parameter family of solutions: we only have to ensure that each triple $\mathbf{b}_{3i-1}, \mathbf{b}_{3i}, \mathbf{b}_{3i+1}$ is collinear on the tangent at $\mathbf{b}_{3i}$ and ordered by increasing subscript in the direction of $\mathbf{l}_i$. We can then find a parametrization with respect to which the generated curve is C^1 [see (7.9)].

In general, we must determine the inner Bézier points from

$$\mathbf{b}_{3i+1} = \mathbf{b}_{3i} + \alpha_i \mathbf{l}_i, \tag{8.6}$$

$$\mathbf{b}_{3i-1} = \mathbf{b}_{3i} - \beta_{i-1} \mathbf{l}_i, \tag{8.7}$$

so that the problem boils down to finding reasonable values for α_i and β_i. While any nonnegative value for these numbers is a formally valid solution, values for α_i and β_i that are too small cause the curve to have a corner at $\mathbf{x}_i$, while values that are too large can create loops. There is probably no optimal choice for α_i and β_i that holds up in every conceivable application—an optimal choice must depend on the desired application.

A "quick and easy" solution that has performed decently many times (but also failed sometimes) is simply to set

$$\alpha_i = \beta_i = 0.4||\Delta \mathbf{x}_i||. \tag{8.8}$$

(The factor 0.4 is, of course, heuristic.)

The parametrization with respect to which this interpolant is C^1 is the *chord length parametrization*. It is characterized by

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{\beta_i}{\alpha_{i+1}} = \frac{||\Delta \mathbf{x}_i||}{||\Delta \mathbf{x}_{i+1}||}. \tag{8.9}$$

A more sophisticated solution is the following: if we consider the planar curve in Figure 8.3, we see that it can be interpreted as a function, where the parameter t varies along the straight line through $\mathbf{b}_0$ and $\mathbf{b}_3$. Then

$$||\Delta \mathbf{b}_{3i}|| = \frac{||\mathbf{b}_{3i+3} - \mathbf{b}_{3i}||}{3 \cos \Theta_i},$$

$$||\Delta \mathbf{b}_{3i+2}|| = \frac{||\mathbf{b}_{3i+3} - \mathbf{b}_{3i}||}{3 \cos \Psi_{i+1}}.$$

We are dealing with parametric curves, however, which are in general not planar and for which the angles Θ and Ψ could be close to 90 degrees, causing the preceding expressions to be undefined. But for curves with Θ_i, Ψ_{i+1} smaller than, say, 60

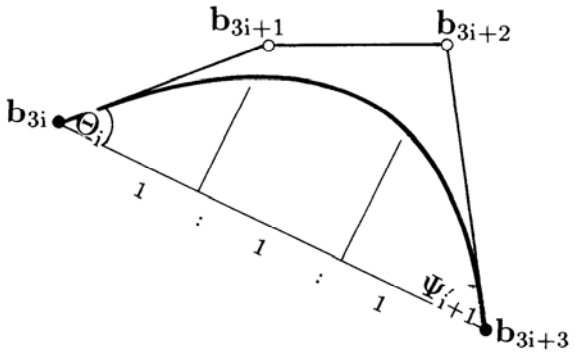

Figure 8.3: Inner Bézier points: this planar curve can be interpreted as a *function* in an oblique coordinate system with $\mathbf{b}_{3i}$, $\mathbf{b}_{3i+3}$ as the x-axis.

degrees, the foregoing could be utilized to find reasonable values for α_i and β_i:

$$\alpha_i = \frac{1}{3\cos\Theta_i}||\Delta\mathbf{x}_i||,$$
$$\beta_i = \frac{1}{3\cos\Psi_{i+1}}||\Delta\mathbf{x}_i||.$$

Since $\cos 60^\circ = 1/2$, we can now make a case distinction:

$$\alpha_i = \begin{cases} \frac{||\Delta\mathbf{x}_i||^2}{3\mathbf{l}_i\Delta\mathbf{x}_i} & \text{if } |\Theta_i| \le 60^\circ \\ \frac{2}{3}||\Delta\mathbf{x}_i|| & \text{otherwise} \end{cases} \tag{8.10}$$

and

$$\beta_i = \begin{cases} \frac{||\Delta\mathbf{x}_i||^2}{3\mathbf{l}_{i+1}\Delta\mathbf{x}_i} & \text{if } |\Psi_{i+1}| \le 60^\circ \\ \frac{2}{3}||\Delta\mathbf{x}_i|| & \text{otherwise.} \end{cases} \tag{8.11}$$

This method has the advantage of having *linear precision*. It is C^1 when the knot sequence satisfies $\Delta_i/\Delta_{i+1} = \beta_i/\alpha_{i+1}$.

Note that neither of these two methods is affinely invariant: the first method, (8.8), does not preserve the ratios of the three points $\mathbf{b}_{3i-1}$, $\mathbf{b}_{3i}$, $\mathbf{b}_{3i+1}$ because the ratios $||\Delta\mathbf{x}_{i-1}|| : ||\Delta\mathbf{x}_i||$ are not generally invariant under affine maps.[1] The second method uses angles, which are not preserved under affine transformations. However, both methods are invariant under euclidean transformations.

[1] Recall that only the ratio of three *collinear* points is preserved under affine maps!

8.3 C^1 Piecewise Cubic Interpolation II

Continuing with the relaxation of given constraints for the interpolatory C^1 cubic spline curve, we now address the following problem:

Given: Data points $\mathbf{x}_0, \ldots, \mathbf{x}_L$ together with corresponding parameter values $u_0, \ldots, u_L$.

Find: A C^1 piecewise cubic polynomial that passes through the given data points.

One solution to this problem is provided by C^2 (and hence also C^1) cubic splines, which are discussed in Chapter 9. Here, we will determine tangent directions $s\mathbf{l}_i$ or tangent vectors $\mathbf{m}_i$ and then apply the methods from the previous two sections.

The simplest method for tangent estimation is known under the name FMILL. It constructs the tangent direction $\mathbf{l}_i$ at $\mathbf{x}_i$ to be parallel to the chord through $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$:

$$\mathbf{v}_i = \mathbf{x}_{i+1} - \mathbf{x}_{i-1}; \quad i = 1, \ldots, L-1. \tag{8.12}$$

Once the tangent direction $\mathbf{v}_i$ has been found,[2] the inner Bézier points are placed on it according to Figure 8.4:

$$\mathbf{b}_{3i-1} = \mathbf{b}_{3i} - \frac{\Delta_{i-1}}{3(\Delta_{i-1} + \Delta_i)}\mathbf{v}_i, \tag{8.13}$$

$$\mathbf{b}_{3i+1} = \mathbf{b}_{3i} + \frac{\Delta_i}{3(\Delta_{i-1} + \Delta_i)}\mathbf{v}_i. \tag{8.14}$$

This interpolant is also known as a Catmull–Rom spline.

This construction of the inner Bézier points does not work at $\mathbf{x}_0$ and $\mathbf{x}_L$. The next method, Bessel tangents, does not have that problem.

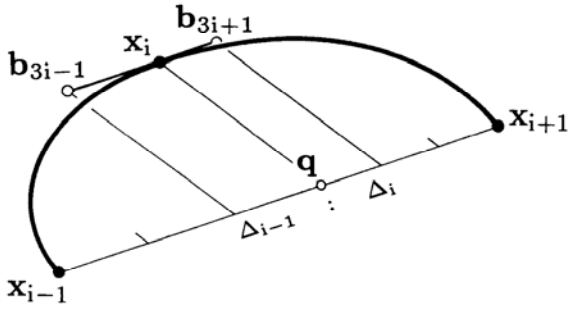

Figure 8.4: FMILL tangents: the tangent at $\mathbf{x}_i$ is parallel to the chord through $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$.

[2]Note that here we do not have $||\mathbf{v}_i|| = 1$!

The idea behind *Bessel tangents*[3] is as follows: to find the tangent vector $\mathbf{m}_i$ at $\mathbf{x}_i$, pass the interpolating parabola $\mathbf{q}_i(u)$ through $\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}$ with corresponding parameter values u_{i-1}, u_i, u_{i+1} and let $\mathbf{m}_i$ be the derivative of $\mathbf{q}_i$. We differentiate $\mathbf{q}_i$ at u_i:

$$\mathbf{m}_i = \frac{\mathrm{d}}{\mathrm{d}u}\mathbf{q}_i(u_i).$$

Written in terms of the given data, this gives

$$\mathbf{m}_i = \frac{(1-\alpha_i)}{\Delta_{i-1}}\Delta\mathbf{x}_{i-1} + \frac{\alpha_i}{\Delta_i}\Delta\mathbf{x}_i; \quad i = 1, \ldots, L-1, \tag{8.15}$$

where

$$\alpha_i = \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i}.$$

The endpoints are treated in the same way: $\mathbf{m}_0 = \mathrm{d}/\mathrm{d}u\mathbf{q}_1(u_0)$, $\mathbf{m}_L = \mathrm{d}/\mathrm{d}u\mathbf{q}_{L-1}(u_L)$, which gives

$$\mathbf{m}_0 = 2\frac{\Delta\mathbf{x}_0}{\Delta_0} - \mathbf{m}_1,$$

$$\mathbf{m}_L = 2\frac{\Delta\mathbf{x}_{L-1}}{\Delta_{L-1}} - \mathbf{m}_{L-1}.$$

Another interpolant that makes use of the parabolas $\mathbf{q}_i$ is known as an *Overhauser spline*, after work by A. Overhauser [379] (see also [81] and [141]). The i^{th} segment $\mathbf{s}_i$ of such a spline (defined over $[u_i, u_{i+1}]$) is defined by

$$\mathbf{s}_i(u) = \frac{u_{i+1} - u}{\Delta_i}\mathbf{q}_i(u) + \frac{u - u_i}{\Delta_i}\mathbf{q}_{i+1}(u); \quad i = 1, \ldots, L-2.$$

In other words, each $\mathbf{s}_i$ is a linear blend between $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$. At the ends, one sets $\mathbf{s}_0(u) = \mathbf{q}_0(u)$ and $\mathbf{s}_{L-1}(u) = \mathbf{q}_{L-1}(u)$.

On closer inspection it turns out that the last two interpolants are not different at all: they both yield the same C^1 piecewise cubic interpolant (see Exercises). A similar way of determining tangent vectors was developed by McConalogue [353], [354].

Finally, we mention a method created by H. Akima [4]. It sets

$$\mathbf{m}_i = (1 - c_i)\mathbf{a}_{i-1} + c_i\mathbf{a}_i,$$

where

$$\mathbf{a}_i = \frac{\Delta\mathbf{x}_i}{\Delta_i}$$

and

$$c_i = \frac{||\Delta\mathbf{a}_{i-2}||}{||\Delta\mathbf{a}_{i-2}|| + ||\Delta\mathbf{a}_i||}.$$

This interpolant appears fairly involved. It generates very good results, however, in situations where one needs curves that oscillate only minimally.

[3] They are also attributed to Ackland [2].

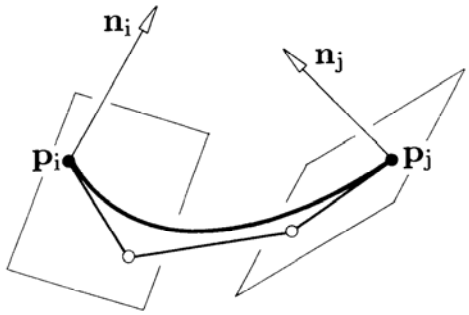

Figure 8.5: Finding cubic boundaries: while the endpoints of a boundary curve are fixed, its end tangents only have to lie in specified planes.

8.4 Point-Normal Interpolation

In a surface generation environment, one is often given a set of points $\mathbf{p}_i \in \mathbb{E}^3$ and a surface normal vector $\mathbf{n}_i$ at each data point, as illustrated in Figure 8.5. Thus we only know the tangent plane of the desired surface at each data point, not the actual endpoint derivatives of the patch boundary curves.

If we know that two points $\mathbf{p}_i$ and $\mathbf{p}_j$ have to be connected, then we must construct a curve leading from $\mathbf{p}_i$ to $\mathbf{p}_j$ that is normal to $\mathbf{n}_i$ at $\mathbf{p}_i$ and to $\mathbf{n}_j$ at $\mathbf{p}_j$. A cubic will suffice to solve this generalized Hermite interpolation problem. In Bézier form, we already have $\mathbf{b}_0 = \mathbf{p}_i$ and $\mathbf{b}_3 = \mathbf{p}_j$. We still need to find $\mathbf{b}_1$ and $\mathbf{b}_2$.

There are infinitely many solutions, so we may try to pick one that is both convenient to compute and of reasonable shape in most cases. Two approaches to this problem appear in Piper [402] and Nielson [376]. Both approaches, although formulated differently, yield the same result.

As a first approximation to $\mathbf{b}_1$, project $\mathbf{b}_3$ into the plane defined by $\mathbf{b}_0 = \mathbf{p}_i$ and $\mathbf{n}_i$. This defines a tangent at $\mathbf{b}_0$. Place the final $\mathbf{b}_1$ anywhere on this tangent, using some of the methods described in Section 8.2. The remaining point $\mathbf{b}_2$ is then obtained analogously.

8.5 Font Design

We conclude this chapter with an application of growing importance, namely *font design*. A graphics language such as PostScript has to generate characters for many different font sets—Arabic, Helvetica, boldface, just to name a few. These fonts must be scaleable, i.e., if a different font size is desired, the original fonts must be rescaled. Had the original fonts been stored as pixel maps, scaling would cause serious aliasing problems. It is common practice, therefore, not to store a given character as a pixel map, but rather to store its outline as a sequence of Bézier curves. These allow smooth

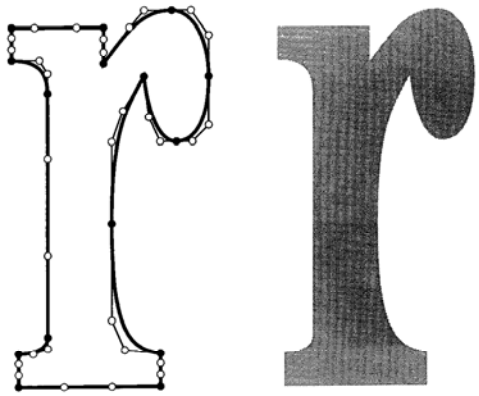

Figure 8.6: Font design: the characters in this book are stored as a sequence of cubic Bézier curves.

arcs where desired, and also allow for sharp corners, as shown in Figure 8.6.[4] This book was printed using PostScript, so all characters have been generated as piecewise cubic Bézier curves.

8.6 Exercises

1. Show that Akima's interpolant always passes a straight line segment through three subsequent points if they happen to lie on a straight line.

*2. Show that Overhauser splines are piecewise cubics with Bessel tangents at the junction points.

*3. One can generalize the quintic Hermite interpolants from Section 6.6 to piecewise quintic Hermite interpolants. These curves need first and second derivatives as input positions. Devise ways to generate second derivative information from data points and parameter values.

P1. Using piecewise cubic C^1 interpolation, approximate the semicircle with radius 1 to within a tolerance of $\epsilon = 0.001$. Use as few cubic segments as possible. Literature: [156], [228].

P2. Program the methods from Section 8.3. Apply to the semicircle from the previous problem and compare to the special-purpose interpolant developed there.

[4] This is my own rendition of the letter **r**.

Chapter 9

Cubic Spline Interpolation

In this chapter, we discuss what is probably *the* most popular curve scheme: C^2 cubic interpolatory splines. We have seen how polynomial Lagrange interpolation fails to produce acceptable results. On the other hand, we saw that cubic B-spline curves are a powerful modeling tool; they are able to model complex shapes easily. This "modeling" is carried out as an *approximation* process, manipulating the control polygon until a desired shape is achieved. We will see how cubic splines can also be used to fulfill the task of *interpolation*, the task of finding a spline curve passing through a given set of points. Cubic spline interpolation was introduced into the CAGD literature by J. Ferguson [202] in 1964, while the mathematical theory was studied in approximation theory (see de Boor [124] or Holladay [285]). For an outline of the history of splines, see Schumaker [452].

Because of the subject's importance, we present two entirely independent derivations of cubic interpolatory splines: the B-spline form and the Hermite form.

9.1 The B-spline Form

We are given a set of data points $\mathbf{x}_0, \ldots, \mathbf{x}_L$ and corresponding parameter values (or knots or breakpoints) $u_0, \ldots, u_L$.[1] We want a cubic B-spline curve $\mathbf{s}$, determined by the same knots and unknown control vertices $\mathbf{d}_{-1}, \ldots, \mathbf{d}_{L+1}$ such that $\mathbf{s}(u_i) = \mathbf{x}_i$: in other words, such that $\mathbf{s}$ *interpolates* to the data points.

The solution to this problem becomes obvious once one realizes the relationship between the data points $\mathbf{x}_i$ and the control vertices $\mathbf{d}_i$. Recall that we can write every B-spline curve as a piecewise Bézier curve (see Section 7.6). In that form, we have

$$\mathbf{x}_i = \mathbf{b}_{3i}; \quad i = 0, \ldots, L.$$

The inner Bézier points $\mathbf{b}_{3i\pm1}$ are related to the $\mathbf{x}_i$ by

$$\mathbf{x}_i = \frac{\Delta_i \mathbf{b}_{3i-1} + \Delta_{i-1} \mathbf{b}_{3i+1}}{\Delta_{i-1} + \Delta_i} \quad i = 1, \ldots, L-1, \tag{9.1}$$

[1] The knots are in general *not* given—see Section 9.4 on how to generate them.

where we have set $\Delta_i = \Delta u_i$. Finally, the $\mathbf{b}_{3i\pm1}$ are related to the control vertices $\mathbf{d}_i$ by

$$\mathbf{b}_{3i-1} = \frac{\Delta_i \mathbf{d}_{i-1} + (\Delta_{i-2} + \Delta_{i-1})\mathbf{d}_i}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}; \quad i = 2, \ldots L-1 \tag{9.2}$$

and

$$\mathbf{b}_{3i+1} = \frac{(\Delta_i + \Delta_{i+1})\mathbf{d}_i + \Delta_{i-1}\mathbf{d}_{i+1}}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}; \quad i = 1, \ldots L-2. \tag{9.3}$$

Near the endpoints of the curve, the situation is somewhat special:

$$\mathbf{b}_2 = \frac{\Delta_1 \mathbf{d}_0 + \Delta_0 \mathbf{d}_1}{\Delta_0 + \Delta_1}, \tag{9.4}$$

$$\mathbf{b}_{3L-2} = \frac{\Delta_{L-1}\mathbf{d}_{L-1} + \Delta_{L-2}\mathbf{d}_L}{\Delta_{L-2} + \Delta_{L-1}}. \tag{9.5}$$

We can now write down the relationships between the unknown $\mathbf{d}_i$ and the known $\mathbf{x}_i$, i.e., we can eliminate the $\mathbf{b}_i$:

$$(\Delta_{i-1} + \Delta_i)\mathbf{x}_i = \alpha_i \mathbf{d}_{i-1} + \beta_i \mathbf{d}_i + \gamma_i \mathbf{d}_{i+1}, \tag{9.6}$$

where we have set (with $\Delta_{-1} = \Delta_L = 0$):

$$\begin{aligned}
\alpha_i &= \frac{(\Delta_i)^2}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}, \\
\beta_i &= \frac{\Delta_i(\Delta_{i-2} + \Delta_{i-1})}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \frac{\Delta_{i-1}(\Delta_i + \Delta_{i+1})}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}, \\
\gamma_i &= \frac{(\Delta_{i-1})^2}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}.
\end{aligned}$$

If we choose the two Bézier points $\mathbf{b}_1$ and $\mathbf{b}_{3L-1}$ arbitrarily, we obtain a linear system of the form

$$\begin{bmatrix}
1 & & & & & & \\
\alpha_1 & \beta_1 & \gamma_1 & & & & \\
 & & \ddots & & & & \\
 & & & \alpha_{L-1} & \beta_{L-1} & \gamma_{L-1} & \\
 & & & & & 1
\end{bmatrix}
\begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{L-1} \\ \mathbf{d}_L \end{bmatrix}
=
\begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-1} \\ \mathbf{r}_L \end{bmatrix}. \tag{9.7}$$

Here we set

$$\begin{aligned}
\mathbf{r}_0 &= \mathbf{b}_1, \\
\mathbf{r}_i &= (\Delta_{i-1} + \Delta_i)\mathbf{x}_i, \\
\mathbf{r}_L &= \mathbf{b}_{3L-1}.
\end{aligned}$$

The first and last polygon vertices do not cause much of a problem:

$$\mathbf{d}_{-1} = \mathbf{x}_0, \quad \mathbf{d}_{L+1} = \mathbf{x}_L.$$

This linear system can be made *symmetric*: we can multiply each equation by a common factor. In particular, we can divide the i^{th} equation through by $\Delta_{i-1}^2 \Delta_i^2$. Also, we would have to delete the first and last rows and columns from the system and update the right-hand side accordingly. The resulting new matrix will now be symmetric; its entries will satisfy $\alpha_{i+1} = \gamma_i$.

The coefficient matrix will be *diagonally dominant* if $\Delta_{i-2} + \Delta_{i-1} > \Delta_i$, which is easily seen from the symmetric version of the linear system. If this condition holds for all i, the system has a unique solution. The fact that it *always* has a solution, as long as the u_i are increasing, is not that easily seen. It is a consequence of the Schoenberg-Whitney theorem, for which the reader is referred to Chapter XIII of de Boor's *Guide to Splines* [126].

Note that an affine parameter transformation does not affect the linear system. Therefore it would not matter if we rescaled our parameter values u_i.

In the special case of all Δ_i being equal, that is, for an equidistant parametrization, the system becomes even simpler:

$$\begin{bmatrix} 1 & & & & & & & & \\ \frac{3}{2} & \frac{7}{2} & 1 & & & & & & \\ & 1 & 4 & 1 & & & & & \\ & & & \ddots & & & & & \\ & & & & 1 & 4 & 1 & & \\ & & & & & 1 & \frac{7}{2} & \frac{3}{2} & \\ & & & & & & & 1 & \end{bmatrix} \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_{L-2} \\ \mathbf{d}_{L-1} \\ \mathbf{d}_L \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ 6\mathbf{x}_1 \\ 6\mathbf{x}_2 \\ \vdots \\ 6\mathbf{x}_{L-2} \\ 6\mathbf{x}_{L-1} \\ \mathbf{b}_{3L-1} \end{bmatrix}. \tag{9.8}$$

Frequently one must deal with *closed* curves; see Figure 9.1. The number of equations is reduced since the C^2 condition at $\mathbf{x}_0 = \mathbf{x}_L$ should not be listed twice in the linear system. It now takes the form:

$$\begin{bmatrix} \beta_0 & \gamma_0 & & & & & \alpha_0 \\ \alpha_1 & \beta_1 & \gamma_1 & & & & \\ & & \ddots & & & & \\ & & & \alpha_{L-2} & \beta_{L-2} & \gamma_{L-2} & \\ \gamma_{L-1} & & & & \alpha_{L-1} & \beta_{L-1} & \end{bmatrix} \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{L-2} \\ \mathbf{d}_{L-1} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-2} \\ \mathbf{r}_{L-1} \end{bmatrix}. \tag{9.9}$$

Here, the right-hand sides are of the form

$$\mathbf{r}_i = (\Delta_{i-1} + \Delta_i)\mathbf{x}_i.$$

For these equations to make sense, we define a *periodic continuation* of the knot sequence:

$$\Delta_{-1} = \Delta_{L-1}, \Delta_{-2} = \Delta_{L-2}.$$

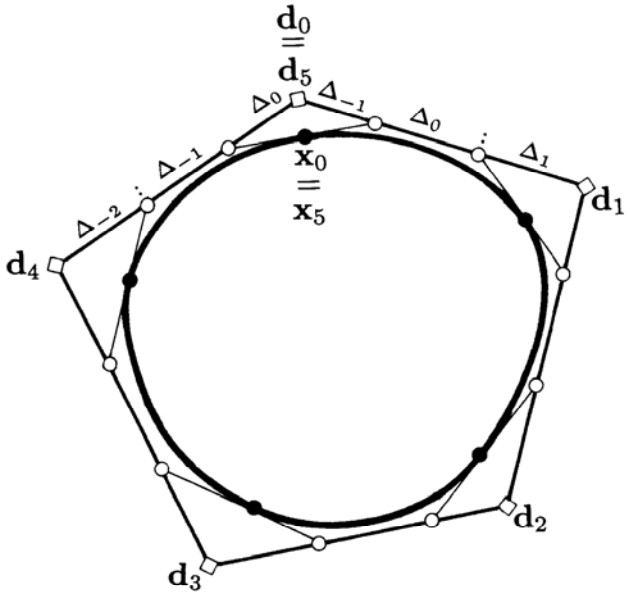

Figure 9.1: Closed curves: the interpolation problem becomes periodic.

The matrix of this system is no longer tridiagonal; yet one does not have to resort to solving a full linear system. For details, see Ahlberg *et al.* [3], p. 15.

We conclude with a method for B-spline interpolation that occasionally appears in the literature (e.g., in Yamaguchi [503]). It is possible to solve the interpolation problem without setting up a linear system! Just do the following: construct an initial control polygon—by setting $\mathbf{d}_i = \mathbf{x}_i$, for example. This initial polygon will not define an interpolating curve. So, for i from 0 to L, correct $\mathbf{d}_i$ such that the corresponding curve passes through $\mathbf{x}_i$.[2] Repeat until the solution is found.

This method will always converge, and will not need many steps in order to do so. So why bother with linear systems? The reason is that tridiagonal systems are most effectively solved by a direct method, whereas the above iterative method amounts to solving the system via Gauss-Seidel iteration. So while geometrically appealing, the iterative method needs more computation time than the direct method.

9.2 The Hermite Form

An interpolatory C^2 piecewise cubic spline may also be written in piecewise cubic Hermite form. For $u \in [u_i, u_{i+1}]$, the interpolant is of the form

$$\mathbf{x}(u) = \mathbf{x}_i H_0^3(r) + \mathbf{m}_i \Delta_i H_1^3(r) + \Delta_i \mathbf{m}_{i+1} H_2^3(r) + \mathbf{x}_{i+1} H_3^3(r), \tag{9.10}$$

[2] See Section 7.8 for details.

where the H_j^3 are cubic Hermite polynomials from (6.14) and $r = (u - u_i)/\Delta_i$ is the local parameter of the interval $[u_i, u_{i+1}]$. In (9.10), the $\mathbf{x}_i$ are the known data points, while the $\mathbf{m}_i = \dot{\mathbf{x}}(u_i)$ are the unknown tangent vectors. The interpolant is supposed to be C^2; therefore,

$$\ddot{\mathbf{x}}_+(u_i) - \ddot{\mathbf{x}}_-(u_i) = \mathbf{0}. \tag{9.11}$$

We insert (9.10) into (9.11) and obtain

$$\begin{aligned} \Delta_i \mathbf{m}_{i-1} + 2(\Delta_{i-1} + \Delta_i)\mathbf{m}_i + \Delta_{i-1}\mathbf{m}_{i+1} = 3\left(\tfrac{\Delta_i \Delta \mathbf{x}_{i-1}}{\Delta_{i-1}} + \tfrac{\Delta_{i-1}\Delta \mathbf{x}_i}{\Delta_i}\right); \\ i = 1, \ldots, L-1. \end{aligned} \tag{9.12}$$

Together with two end conditions, (9.12) can be used to compute the unknown tangent vectors $\mathbf{m}_i$. Note that this formulation of the spline interpolation problem depends on the scale of the u_i; it is not invariant under affine parameter transformations. This is a result of the use of the Hermite form.

The simplest end condition would be to prescribe $\mathbf{m}_0$ and $\mathbf{m}_L$, a method known as *clamped end condition*. In that case, the matrix of our linear system takes the form

$$\begin{bmatrix} 1 & & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & \\ & \ddots & & & & & \\ & & & \alpha_{L-1} & \beta_{L-1} & \gamma_{L-1} & \\ & & & & & 1 & \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_{L-1} \\ \mathbf{m}_L \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-1} \\ \mathbf{r}_L \end{bmatrix}, \tag{9.13}$$

where

$$\begin{aligned} \alpha_i &= \Delta_i, \\ \beta_i &= 2(\Delta_{i-1} + \Delta_i), \\ \gamma_i &= \Delta_{i-1} \end{aligned}$$

and

$$\begin{aligned} \mathbf{r}_0 &= \mathbf{m}_0, \\ \mathbf{r}_i &= 3\left(\frac{\Delta_i \Delta \mathbf{x}_{i-1}}{\Delta_{i-1}} + \frac{\Delta_{i-1}\Delta \mathbf{x}_i}{\Delta_i}\right); \quad i = 1, \ldots, L-1, \\ \mathbf{r}_L &= \mathbf{m}_L. \end{aligned}$$

Having found the $\mathbf{m}_i$, we can easily retrieve the piecewise Bézier form of the curve according to (8.2) and (8.3).

When dealing with linear systems, it is a good idea to make sure that a solution exists and that it is unique. In our case, the coefficient matrix is *diagonally dominant*, which means that the absolute value of any diagonal element is larger than the sum of the absolute values of the remaining elements on the same row:

$$|\beta_i| > |\alpha_i| + |\gamma_i|.$$

Such matrices are always invertible; moreover, they allow Gauss elimination without pivoting (see any advanced text on numerical analysis or the fundamental spline text

by Ahlberg *et al.* [3]). Thus the spline interpolation problem always possesses a unique solution (after the prescription of two consistent end conditions).

Since the coefficient matrix is *tridiagonal* (only the diagonal element and its two neighbors are nonzero), we do not have to solve a full $(L+1)\times(L+1)$ linear system. One forward substitution sweep and one for backward substitution is sufficient, as implemented in the programs `l_u_system` and `solve_system` that follow. All our remarks about the linear system hold for the B-spline form as well.

9.3 End Conditions

We may have the interpolation routine select the end tangents $\mathbf{m}_0$ and $\mathbf{m}_L$ automatically instead of prescribing it ourselves. One such selection is called the *Bessel end condition*. Here, the end tangent vector $\mathbf{m}_0$ is set equal to the tangent vector at $\mathbf{x}_0$ of the interpolating parabola through the first three data points. Similarly, $\mathbf{m}_L$ is set equal to the tangent vector at $\mathbf{x}_L$ of the interpolating parabola through the last three data points. Now the right-hand side changes to

$$\mathbf{r}_0 = -\frac{2(2\Delta_0+\Delta_1)}{\Delta_0\beta_1}\mathbf{x}_0 + \frac{\beta_1}{2\Delta_0\Delta_1}\mathbf{x}_1 - \frac{2\Delta_0}{\Delta_1\beta_1}\mathbf{x}_2 \tag{9.14}$$

and

$$\mathbf{r}_L = \frac{2\Delta_{L-1}}{\Delta_{L-2}\beta_{L-1}}\mathbf{x}_{L-2} - \frac{\beta_{L-1}}{2\Delta_{L-2}\Delta_{L-1}}\mathbf{x}_{L-1} + \frac{2(2\Delta_{L-1}+\Delta_{L-2})}{\beta_{L-1}\Delta_{L-1}}\mathbf{x}_L. \tag{9.15}$$

Of course, this condition may also be formulated in terms of the B-spline representation. This amounts to finding the control points $\mathbf{d}_0$ and $\mathbf{d}_L$, which are actually Bézier points. They were already determined in the context of C^1 piecewise cubic interpolation; see (8.15). C code for this version of Bessel end conditions is given in the routine `bessel_ends` described later.

Another possibility is the *quadratic end condition*, which sets $\ddot{\mathbf{x}}(u_0) = \ddot{\mathbf{x}}(u_1)$ and $\ddot{\mathbf{x}}(u_{L-1}) = \ddot{\mathbf{x}}(u_L)$. Now the linear system changes to

$$\begin{bmatrix} 1 & 1 & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & \\ & & \ddots & & & & \\ & & & \alpha_{L-1} & \beta_{L-1} & \gamma_{L-1} & \\ & & & & 1 & 1 & \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_{L-1} \\ \mathbf{m}_L \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-1} \\ \mathbf{r}_L \end{bmatrix} \tag{9.16}$$

and

$$\mathbf{r}_0 = \frac{2}{\Delta_0}\Delta\mathbf{x}_0, \quad \mathbf{r}_L = \frac{2}{\Delta_{L-1}}\Delta\mathbf{x}_{L-1}.$$

A slightly more complicated end condition is provided by the *not-a-knot condition*. Using it, we force the first two and the last two polynomial segments to merge into *one* cubic piece. This means that the third derivative of $\mathbf{x}(u)$ is continuous at u_1.

Writing down the conditions leads to a nontridiagonal system, which can, however, be transformed into a tridiagonal one. Its first equation is

$$\begin{aligned}&\Delta_1\beta_1\mathbf{m}_0 + \beta_1^2\mathbf{m}_1\\ &\quad = \frac{(\Delta_0)^2}{\Delta_1}\Delta\mathbf{x}_1 + \frac{\Delta_1}{\Delta_0}(3\Delta_0 + 2\Delta_1)\Delta\mathbf{x}_0;\end{aligned} \tag{9.17}$$

the last one is

$$\begin{aligned}&\beta_{L-1}^2\mathbf{m}_{L-1} + \Delta_{L-2}\beta_{L-1}\mathbf{m}_L\\ &\quad = \frac{(\Delta_{L-1})^2}{\Delta_{L-2}}\Delta\mathbf{x}_{L-2} + \frac{\Delta_{L-2}}{\Delta_{L-1}}(3\Delta_{L-1} + 2\Delta_{L-2})\Delta\mathbf{x}_{L-1}.\end{aligned} \tag{9.18}$$

Finally, we mention an end condition that bears the name "natural." The term stems from the fact that this condition arises "naturally" in the context of the minimum property for spline curves, as described later in this chapter. The natural end condition is defined by $\ddot{\mathbf{x}}(u_0) = \ddot{\mathbf{x}}(u_L) = \mathbf{0}$. The linear system becomes

$$\begin{bmatrix} 2 & 1 & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & \\ & & \ddots & & & & \\ & & & \alpha_{L-1} & \beta_{L-1} & \gamma_{L-1} & \\ & & & & 1 & 2 & \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_{L-1} \\ \mathbf{m}_L \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-1} \\ \mathbf{r}_L, \end{bmatrix} \tag{9.19}$$

and

$$\mathbf{r}_0 = \frac{3}{\Delta_0}\Delta\mathbf{x}_0, \quad \mathbf{r}_L = \frac{3}{\Delta_{L-1}}\Delta\mathbf{x}_{L-1}.$$

This end condition forces the curve to behave like a straight line near the endpoints; usually, this results in a poor shape of the spline curve.

The spline system becomes especially simple if the knots u_i are uniformly spaced; for example, the clamped end condition system becomes

$$\begin{bmatrix} 1 & & & & & & \\ 1 & 4 & 1 & & & & \\ & & \ddots & & & & \\ & & & 1 & 4 & 1 & \\ & & & & & 1 & \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_{L-1} \\ \mathbf{m}_L \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{L-1} \\ \mathbf{r}_L \end{bmatrix}, \tag{9.20}$$

where

$$\begin{aligned}\mathbf{r}_0 &= \mathbf{m}_0,\\ \mathbf{r}_i &= 3(\mathbf{x}_{i+1} - \mathbf{x}_{i-1}); \quad i = 1, \ldots, L-1,\\ \mathbf{r}_L &= \mathbf{m}_L.\end{aligned}$$

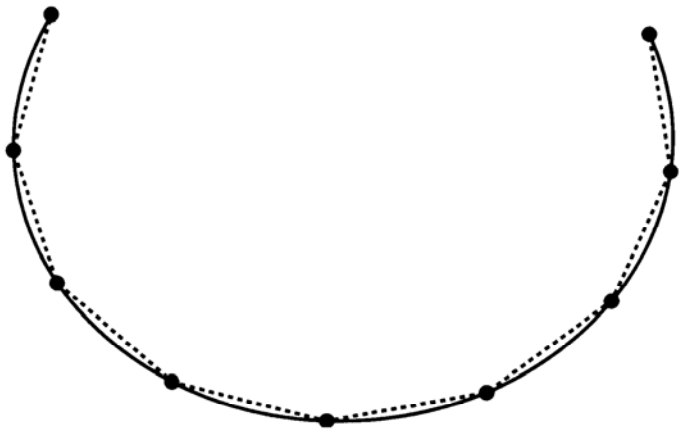

Figure 9.2: Exact clamped end condition spline.

We finish this section with a few examples, using uniform parameter values in all examples.[3] Figure 9.2 shows equally spaced data points read off from a circle of radius 1 and the cubic spline interpolant obtained with clamped end conditions, using the exact end derivatives of the circle. Figure 9.3 shows the curvature plot[4] of the spline curve. Ideally, the curvature should be constant, and the spline curvature is quite close to this ideal.

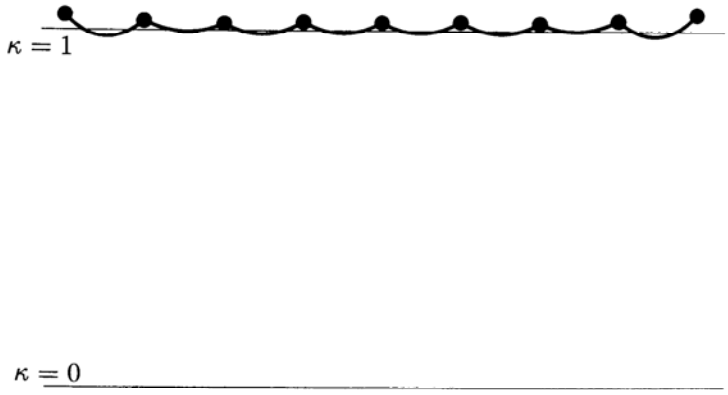

Figure 9.3: Curvature plot of exact clamped end condition spline.

[3]Because of the symmetry inherent in the data points, all parametrizations discussed later yield the same knot spacing. All circle plots are scaled down in the y-direction.

[4]The graph of curvature versus arc length; see also Chapter 23.

Figure 9.4: Bessel end condition spline.

Figure 9.4 shows the same data, but now using Bessel end conditions. Near the endpoints, the curvature deviates from the ideal value, as shown in Figure 9.5.

Finally, Figure 9.6 shows the curve that is obtained using natural end conditions. The end curvatures are forced to be zero, causing considerable deviation from the ideal value, as shown in Figure 9.7.

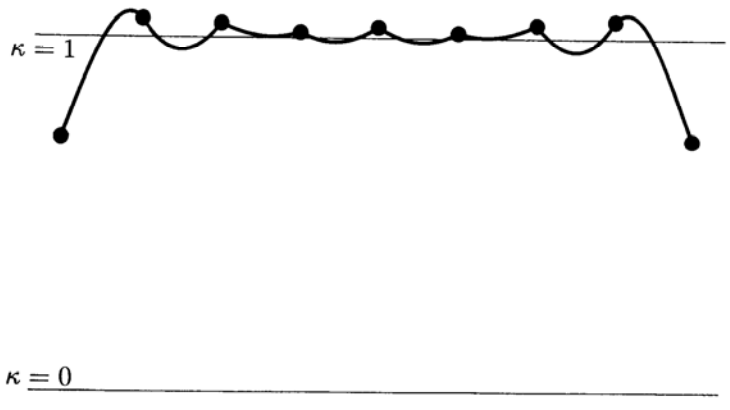

Figure 9.5: Curvature plot of Bessel end condition spline.

Figure 9.6: Natural end condition spline.

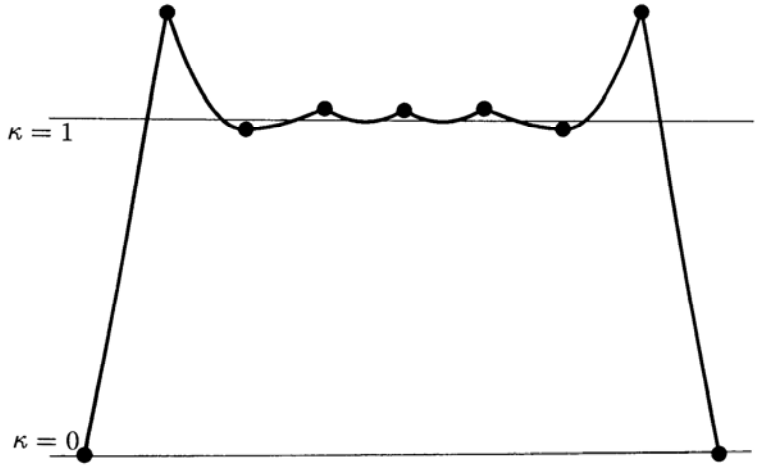

Figure 9.7: Curvature plot of natural end condition spline.

9.4 Finding a Knot Sequence

The spline interpolation problem is usually stated as "given data points $\mathbf{x}_i$ and parameter values $u_i, \ldots$." Of course, this is the mathematician's way of describing a problem. In practice, parameter values are rarely given and therefore must be made up somehow. The easiest way to determine the u_i is simply to set $u_i = i$. This is called *uniform* or *equidistant* parametrization. This method is too simplistic to cope

with most practical situations. The reason for the overall poor[5] performance of the uniform parametrization can be blamed on the fact that it "ignores" the geometry of the data points.

The following is a heuristic explanation of this fact. We can interpret the parameter u of the curve as time. As time passes from time u_0 to time u_L, the point $\mathbf{x}(u)$ traces the curve from point $\mathbf{x}(u_0)$ to point $\mathbf{x}(u_L)$. With uniform parametrization, $\mathbf{x}(u)$ spends the same amount of time between any two adjacent data points, irrespective of their relative distances. A good analogy is a car driving along the interpolating curve. We have to spend the same amount of time between any two data points. If the distance between two data points is large, we must move with a high speed. If the next two data points are close to each other, we will overshoot because we cannot abruptly change our speed—we are moving with continuous speed and acceleration, which are the physical counterparts of a C^2 parametrization of a curve. It would clearly be more reasonable to adjust speed to the distribution of the data points.

One way of achieving this is to have the knot spacing proportional to the distances of the data points:

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{||\Delta \mathbf{x}_i||}{||\Delta \mathbf{x}_{i+1}||}. \tag{9.21}$$

A knot sequence satisfying (9.21) is called *chord length parametrization*. Equation (9.21) does not uniquely define a knot sequence; rather, it defines a whole family of parametrizations that are related to each other by affine parameter transformations. In practice, the choices $u_0 = 0$ and $u_L = 1$ or $u_0 = 0$ and $u_L = L$ are reasonable options.

Chord length usually produces better results than uniform knot spacing, although not in all cases. It has been proven (Epstein [167]) that chord length parametrization (in connection with natural end conditions) cannot produce curves with corners[6] at the data points, which gives it some theoretical advantage over the uniform choice.

Another parametrization has been named "centripetal" by E. Lee [327]. It is derived from the physical heuristics presented above. If we set

$$\frac{\Delta_i}{\Delta_{i+1}} = \left[\frac{||\Delta \mathbf{x}_i||}{||\Delta \mathbf{x}_{i+1}||}\right]^{1/2}, \tag{9.22}$$

the resulting motion of a point on the curve will "smooth out" variations in the centripetal force acting on it.

Yet another parametrization was developed by G. Nielson and T. Foley [377]. It sets

$$\Delta_i = d_i\left[1 + \frac{3}{2}\frac{\hat{\Theta}_i d_{i-1}}{d_{i-1} + d_i} + \frac{3}{2}\frac{\hat{\Theta}_{i+1} d_{i+1}}{d_i + d_{i+1}}\right], \tag{9.23}$$

[5]There are cases in which uniform parametrization fares better than other methods. An interesting example is in Foley [210], p. 86.

[6]A corner is a point on a curve where the tangent (not necessarily the tangent vector!) changes in a discontinuous way. The special case of a change in 180 degrees is called a *cusp*; it may occur even with chord length parametrization.

where $d_i = ||\Delta x_i||$ and

$$\hat{\Theta}_i = \min\left(\pi - \Theta_i, \frac{\pi}{2}\right),$$

and Θ_i is the angle formed by $\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}$. Thus $\hat{\Theta}_i$ is the "adjusted" exterior angle formed by the vectors $\Delta\mathbf{x}_i$ and $\Delta\mathbf{x}_{i-1}$. As the exterior angle $\hat{\Theta}_i$ increases, the interval Δ_i increases from the minimum of its chord length value up to a maximum of four times its chord length value. This method was created to cope with "wild" data sets.

We note one property that distinguishes the uniform parametrization from its competitors: it is the only one that is invariant under affine transformations of the data points. Chord length, centripetal, and the Foley methods all involve length measurements, and lengths are not preserved under affine maps. One solution to this dilemma is the introduction of a modified length measure, as described in Nielson [375].[7]

For more literature on parametrizations, see Cohen and O'Dell [111], Hartley and Judd [273], [274], McConalogue [353] , and Foley [210].

Figures 9.8 to 9.15[8] show the performance of the discussed parametrization methods for one sample data set. For each method, the interpolant is shown together with its curvature plot. For all methods, Bessel end conditions were chosen.

While the figures are self-explanatory, some comments are in place. Note the very uneven spacing of the data points at the marked area of the curves. Of all methods, Foley's copes best with that situation (although we add that many examples exist where the simpler centripetal method wins out). The uniform spline curve seems to have no problems there, if one just inspects the plot of the curve itself. However, the curvature plot reveals a cusp in that region! The huge curvature at the cusp causes a scaling in the curvature plot that annihilates all other information. Also note how the chord length parametrization yields the "roundest" curve, having the smallest curvature values, but exhibiting the most marked inflection points.

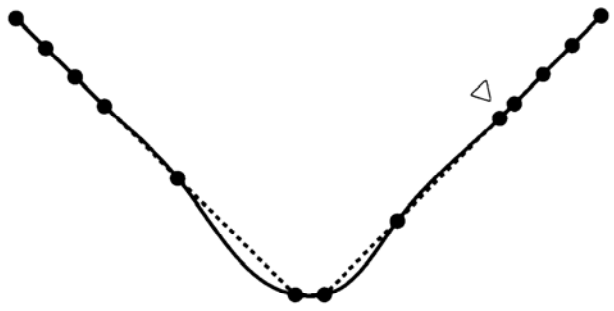

Figure 9.8: Chord length spline.

[7]The Foley parametrization was in fact first formulated in terms of that modified length measure.

[8]Kindly provided by T. Foley.

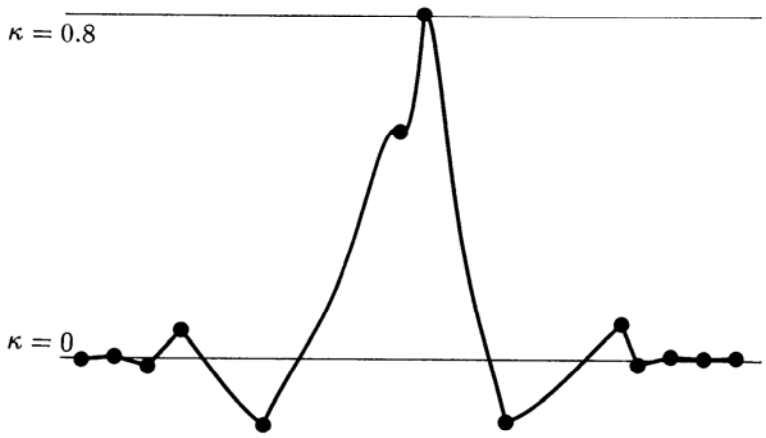

Figure 9.9: Curvature plot of chord length spline.

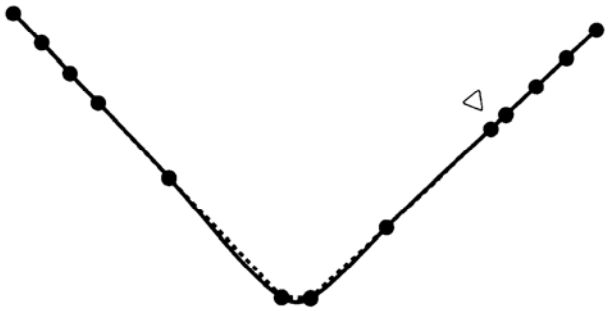

Figure 9.10: Foley spline.

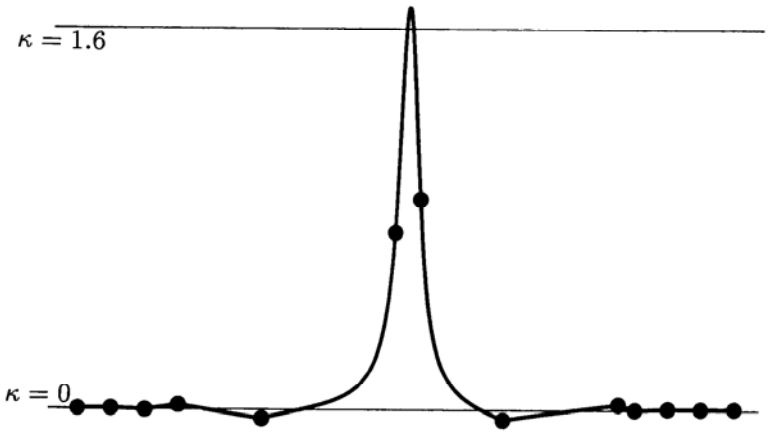

Figure 9.11: Curvature plot of Foley spline.

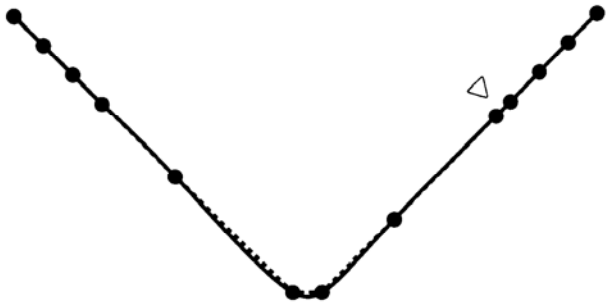

Figure 9.12: Centripetal spline.

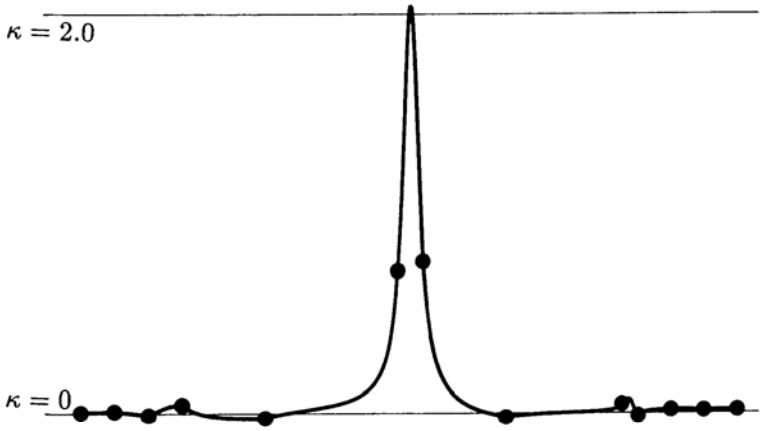

Figure 9.13: Curvature plot of centripetal spline.

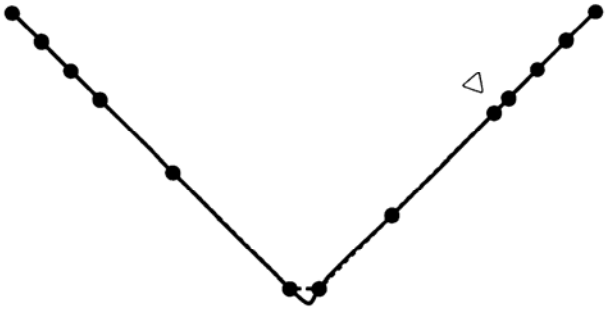

Figure 9.14: Uniform spline.

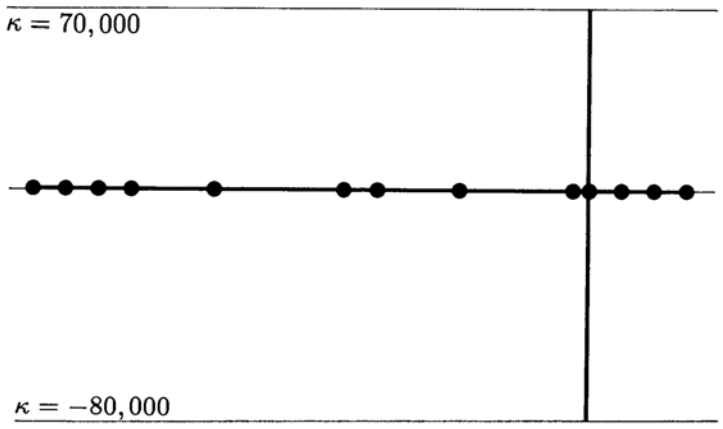

Figure 9.15: Curvature plot of uniform spline.

There is probably no "best" parametrization, since any method can be defeated by a suitably chosen data set. The following is a (personal) recommendation. You may improve the shape of the curve, at the cost of an increase of computation time, by the following hierarchy of methods: uniform, chord length, centripetal, Foley. The best compromise between cost and result is probably achieved by the centripetal method.

9.5 The Minimum Property

In the early days of design, say ship design in the 1800s, the problem had to be handled of how to draw (manually) a smooth curve through a given set of points. One way to obtain a solution was the following: place metal weights (called "ducks") at the data points, and then pass a thin, elastic wooden beam (called a "spline") between the ducks. The resulting curve is always very smooth and usually aesthetically pleasing. The same principle is used today when an appropriate design program is not available or for manual verification of a computer result; see Figure 9.16.

Figure 9.16: Spline interpolation: A plastic beam, the "spline," is forced to pass through data points, marked by metal weights, the "ducks."

The plastic or wooden beam assumes a position that minimizes its strain energy. The mathematical model of the beam is a curve $\mathbf{s}$, and its strain energy E is given by

$$E = \int (\kappa(s))^2 \mathrm{d}s,$$

where κ denotes the curvature of the curve. The curvature of most curves involves integrals and square roots and is cumbersome to handle; therefore, one often approximates the preceding integral by a simpler one:

$$\hat{\mathbf{E}} = \int \left[\frac{\mathrm{d}^2}{\mathrm{d}u^2}\mathbf{s}(u)\right]^2 \mathrm{du}. \tag{9.24}$$

Note that $\hat{\mathbf{E}}$ is a vector; it is obtained by performing the integration on each component of $\mathbf{s}$.

Equation (9.24) is more directly motivated by the following example: when an airplane is scheduled to fly from A to B, it will have to fly over a number of of intermediate "way points." The amount of fuel used by an airplane is mostly affected by its acceleration, which is essentially equivalent to the second derivative of its trajectory. Thus if the plane follows a cubic spline curve passing through all the way points, it will be guaranteed to use the least amount of fuel![9]

In a more general setting, we may word this as: among all C^2 curves interpolating the given data points at the given parameter values and satisfying the same end conditions, the cubic spline yields the smallest value for each component of $\hat{\mathbf{E}}$. For a proof, let $\mathbf{s}(u)$ be the C^2 cubic spline and let $\mathbf{y}(u)$ be another C^2 interpolating curve. We can write $\mathbf{y}$ as

$$\mathbf{y}(u) = \mathbf{s}(u) + [\mathbf{y}(u) - \mathbf{s}(u)].$$

The preceding integrals are defined componentwise; we will show the minimum property for one component only. Let $s(u)$ and $y(u)$ be the first component of $\mathbf{s}$ and $\mathbf{y}$, respectively. The "energy integral" $\hat{E}$ of $\mathbf{y}$'s first component becomes

$$\hat{E} = \int_{u_0}^{u_L} (\ddot{s})^2 \mathrm{d}u + 2\int_{u_0}^{u_L} \ddot{s}(\ddot{y} - \ddot{s})\,\mathrm{d}u + \int_{u_0}^{u_L} (\ddot{y} - \ddot{s})^2 \mathrm{d}u.$$

We may integrate the middle term by parts:

$$\int_{u_0}^{u_L} \ddot{s}(\ddot{y} - \ddot{s})\,\mathrm{d}u = \ddot{s}(\dot{y} - \dot{s})\Big|_{u_0}^{u_L} - \int_{u_0}^{u_L} \dddot{s}(\dot{y} - \dot{s})\,\mathrm{d}u.$$

The first expression vanishes because of the common end conditions. In the second expression, $\dddot{s}$ is piecewise constant:

$$\int_{u_0}^{u_L} \dddot{s}(\dot{y} - \dot{s})\,\mathrm{d}u = \sum_{j=0}^{L-1} \dddot{s}_j (y - s)\Big|_{u_j}^{u_{j+1}}.$$

[9] I am grateful to P. Crouch for bringing the airplane analogy to my attention.

All terms in the sum vanish because both **s** and **y** interpolate. Since

$$\int_{u_0}^{u_L} (\ddot{y} - \ddot{s})^2 \mathrm{d}u > 0$$

for continuous $\ddot{y} \neq \ddot{s}$,

$$\int_{u_0}^{u_L} (\ddot{y})^2 \mathrm{d}u \geq \int_{u_0}^{u_L} (\ddot{s})^2 \mathrm{d}u, \tag{9.25}$$

we have proved the claimed minimum property.

The minimum property of splines has spurred substantial research activity. The replacement of the actual strain energy measure E by $\hat{\mathbf{E}}$ is motivated by the desire for mathematical simplicity. The curvature of a curve is given by

$$\kappa(u) = \frac{||\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}||}{||\dot{\mathbf{x}}||^3}.$$

But we need $||\dot{\mathbf{x}}|| \approx 1$ in order for $||\ddot{\mathbf{x}}||$ to be a good approximation to κ. This means, however, that the curve must be parametrized according to arc length; see (11.7). This assumption is not very realistic for cubic splines in a design environment; see Exercises.

While the classical spline curve merely minimizes an approximation to (9.24), methods have been developed that produce interpolants which minimize the true energy (9.24), see [357], [86]. Moreton and Séquin have suggested to minimize the functional $\int [\kappa'(t)]^2 dt$ instead, see [363].

9.6 Implementation

The following routines produce the cubic B-spline polygon of an interpolating C^2 cubic spline curve. First, we set up the tridiagonal linear system:

```
void set_up_system(knot,l,alpha,beta,gamma)
/*     given the knot sequence, the linear system for clamped end
       condition B-spline interpolation is set up.
Input: knot:              knot sequence (all knots are simple; but,
                          in the terminology of Chapter 10, knot[0]
                          and knot[l] are of multiplicity three.)
       points:            points to be interpolated
       l:                 number of intervals
Output:alpha, beta,gamma: 1-D arrays that constitute
                          the elements of the interpolation matrix.
Note: no data points needed so far!
*/
```

The next routine performs the LU decomposition of the matrix from the previous routine. (Note that we do not generate a full matrix, but rather three linear arrays!)

```
void l_u_system(alpha,beta,gamma,l,up,low)
/*     perform LU decomposition of tridiagonal system with
     lower diagonal alpha, diagonal beta, upper diagonal gamma.

Input: alpha,beta,gamma: the coefficient matrix entries
       l:                matrix size [0,l]x[0,l]
Output:low:              L-matrix entries
        up:              U-matrix entries
*/
```

Finally, the routine that solves the system for the B-spline coefficients $\mathbf{d}_i$:

```
 solve_system(up,low,gamma,l,rhs,d)
/*   solve  tridiagonal linear system
     of size (l+1)(l+1) whose LU decomposition has entries up and low,
     and whose right hand side is rhs, and whose original matrix
     had gamma as  its upper diagonal. Solution is d[0],...,d[l+2].
Input: up,low,gamma:  as above.
       l:             size of system: l+1 eqs in l+1 unknowns.
       rhs:           right hand side, i.e, data points with end
                      'tangent Bezier points' in rhs[1] and rhs[l+1].
Output:d:             solution vector.
Note shift in indexing from text! Both rhs and d are from 0 to l+2.
*/
```

If Bessel ends are desired instead of clamped ends, this is the code:

```
void bessel_ends(data,knot,l)
/*     Computes B-spline points data[1] and data[l+1]
       according to bessel end condition.

Input: data:  sequence of data coordinates data[0] to data[l+2].
              Note that data[1] and data[l+1] are expected to
              be empty, as they will be filled by this routine.
              They correspond to the Bezier points bez[1] and bez[3l-1].
        knot: knot sequence
        l:    number of intervals
Output: data: now including ''tangent Bezier points'' data[1], data[l+1].
*/
```

The centripetal parametrization is achieved by the following routine:

```
void parameters(data_x,data_y,l,knot)
/*   Finds a centripetal parametrization for a given set
     of 2D data points.
Input:     data_x, data_y:  input points, numbered from 0 to l+2.
     l:                     number of intervals.
Output:    knot:            knot sequence. Note: not (knot[l]=1.0)!
Note:      data_x[1], data_x[l+1] are not used! Same for data_y.
*/
```

A calling sequence that utilizes the preceding programs might look like this:

```
parameters(data_x,data_y,l,knot);

set_up_system(knot,l,alpha,beta,gamma);

l_u_system(alpha,beta,gamma,l,up,low);

bessel_ends(data_x,knot,l);
bessel_ends(data_y,knot,l);

solve_system(up,low,gamma,l,data_x,bspl_x);
solve_system(up,low,gamma,l,data_y,bspl_y);
```

Here, we solved the 2D interpolation problem with given data points in `data_x`, `data_y`, a knot sequence `knot`, and the resulting B-spline polygon in `bspl_x`, `bspl_y`. This calling sequence is realized in the routine `c2_spline.c`.

9.7 Exercises

1. Formulate the quadratic and natural end conditions for the case of cubic B-spline interpolation.
2. Although this section is on cubic spline interpolants, we might also have considered quadratic ones. Yet there is a difference: for the case of closed curves, C^1 quadratic spline interpolation with uniform knots does not always have a solution. Why?[10]

*3. Show that interpolating splines reproduce cubic polynomial curves—that they have *cubic precision*. This means that if all data points $\mathbf{x}_i$ are read off from a cubic: $\mathbf{x}_i = \mathbf{c}(u_i)$, and the end tangent vectors are read off from the cubic, then the interpolating spline equals the original cubic.

*4. Any curve may be reparametrized in terms of its arc length s. Show that a polynomial curve of degree $n > 1$ cannot be polynomial in terms of its arc length s. See Chapter 11 for the arc length parametrization—the key condition is that $||\ddot{\mathbf{x}}(s)|| \equiv 1$ if s is the arc length parameter.

P1. Program the following: instead of prescribing end conditions at both ends, prescribe first and second derivatives at u_0. The interpolant can then be built segment by segment. Discuss the numerical aspects of this method (they will not be wonderful).

P2. Interpolate data points from a semicircle and compare your results with those from the corresponding exercises in Section 8.6.

P3. Compare C^2 cubic spline interpolation to the C^1 case from Chapter 8, using the data sets `outline_2D` and `outline_3D`.

[10] T. DeRose pointed this out to me.

Chapter 10

B-splines

B-splines were investigated as early as the nineteenth century by N. Lobachevsky (see Renyi [421], p. 165); they were constructed as convolutions of certain probability distributions.[1] In 1946, I. J. Schoenberg [449] used B-splines for statistical data smoothing, and his paper started the modern theory of spline approximation. For the purposes of this book, the discovery of the recurrence relations for B-splines by C. de Boor [125], M. Cox [118], and L. Mansfield was one of the most important developments in this theory. The recurrence relations were first used by Gordon and Riesenfeld [249] in the context of parametric B-spline curves.

This chapter presents a theory for arbitrary degree B-spline curves. The original development of these curves makes use of divided differences and is mathematically involved (see de Boor [126]). A different approach to B-splines was taken by de Boor and Hollig [131]; they used the recurrence relations for B-splines as the starting point for the theory. In this chapter, the theory of B-splines is based on an even more fundamental concept: the Boehm knot insertion algorithm [62]. Another interesting new approach to B-splines is the *blossoming* method proposed by L. Ramshaw [414] and, in a different form, by P. de Casteljau [135], which we will also discuss in this chapter.

Warning: *subscripts in this chapter differ from those in Chapter 7!* For the cubic and quadratic cases special subscripts are useful, but the general theory is easier to explain with the notation used here.[2]

10.1 Motivation

Figure 10.1 shows a C^2 cubic spline (nonparametric) with its B-spline polygon. The relationship between the polygon and the curve was discussed in Section 7.6. In

[1] However, those were only defined over a very special knot sequence.

[2] In terms of this chapter, we used end knots of multiplicity two (quadratic case) or three (cubic case) in Chapter 7. The coefficients there started with the subscript $i = -1$; here, they will start with $i = 0$.

that section, we were interested in the parametric case, whereas now we will restrict ourselves to nonparametric (functional) curves of the form $y = f(u)$. The reason is that much of the B-spline theory is explained more naturally in this setting.

In Section 5.5, we considered nonparametric Bézier curves. Recall that over the interval $[u_i, u_{i+1}]$, the abscissas of the Bézier points are $u_i + j\Delta u_i/n; j = 0, \ldots, n$. Two cubic Bézier functions that are defined over $[u_{i-1}, u_i]$ and $[u_i, u_{i+1}]$ are C^2 at u_i if an auxiliary point $\mathbf{d}_i = (\xi_i, d_i)$ can be constructed from both curve segments as discussed in Section 7.3. Some of the points $\mathbf{d}_i$ are shown in Figure 10.1. Section 7.3 tells us how to compute the y-values d_i of these points. Using the same reasoning for the u-coordinates ξ_i (see Exercises), we find

$$\xi_i = \frac{1}{3}(u_{i-1} + u_i + u_{i+1}); \quad i = 1, 2, 3.^3 \tag{10.1}$$

We can now give an algorithm for the "design" of a cubic B-spline function:

1. **Given** knots u_i.
2. **Find** abscissas $\xi_i = \frac{1}{3}(u_{i-1} + u_i + u_{i+1})$.
3. **Define** real numbers d_i to obtain a polygon with vertices (ξ_i, d_i).
4. **Evaluate** this polygon (= piecewise linear function) at the abscissas of the inner Bézier points. This produces a refined polygon, consisting of the inner Bézier points.
5. **Evaluate** the refined polygon at the knots u_i, the abscissas for the junction Bézier points. We now have the junction Bézier points.

After step 5, we have generated a C^2 piecewise cubic Bézier function. In a similar manner, we could generate a C^1 piecewise quadratic Bézier function. In this chapter, we will aim for a generalization of the preceding definition of piecewise polynomials to include arbitrary degrees and arbitrary differentiability classes.

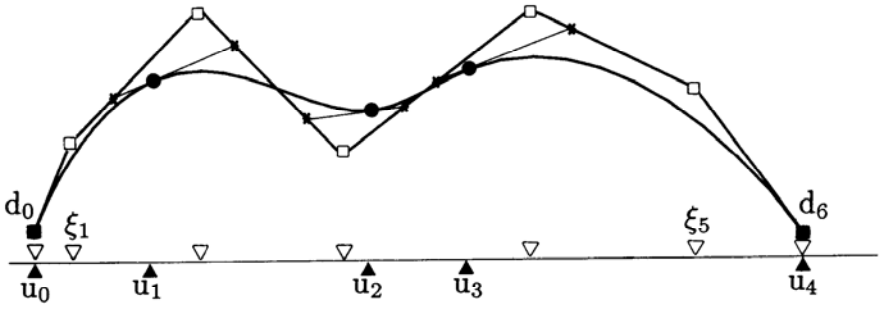

Figure 10.1: B-splines: a nonparametric C^2 cubic spline curve with its B-spline polygon. (In the context of this chapter, the end knots are of multiplicity three.)

[3]This notation is still in harmony with the cubic case of Section 7.3; we will change notation for the general case soon!

10.2 Knot Insertion

We will now define an algorithm to "refine" a piecewise linear function. Later, this piecewise linear function will be interpreted as a B-spline polygon, but at this point, we discuss only an algorithm that produces one piecewise linear function from another.

Suppose that we are given a number n (later the degree of the B-spline curve) and a number L (later related to the number of polynomial segments of the B-spline curve). Suppose also that we are given a nondecreasing *knot sequence*

$$u_0, \ldots, u_{L+2n-2}.$$

Not all of the u_i have to be distinct. If $u_i = \cdots = u_{i+r-1}$, i.e., if r successive knots coincide, we say that u_i has *multiplicity* r. If a knot does not coincide with any other knot, we say that it is *simple*, or that it has multiplicity one.

If knots have multiplicity higher than one, we have two alternatives: we may list them in our knot sequence repeatedly, or we may list them only once, keeping track of their multiplicity in a separate sequence.

When we define B-spline curves later, we will use only the interval $[u_{n-1}, \ldots, u_{L+n-1}]$ as their domain. These knots are the "domain knots." We call L the potential number of curve segments—if all domain knots are simple, L denotes the number of domain intervals. For each domain knot multiplicity, the number of domain intervals drops by one. If we list each knot only once and keep track of its multiplicity, the sum of all domain knot multiplicities is related to L by

$$\sum_{i=n-1}^{L+n-1} r_i = L + 1,$$

where r_i is the multiplicity of the domain knot u_i.

We shall illustrate the interplay between knot multiplicities and the number of domain intervals by means of the following examples:

Let $n = 3$, $L = 3$, and

$$\{u_0, \ldots, u_7\} = \{0, 1, 2, 3, 4, 5, 6, 7\}.$$

All knots are simple, so the number of domain intervals is three.

Leaving n and L unchanged, consider

$$\{u_0, \ldots, u_6\} = \{0, 1, 2, 3, 5, 6, 7\},$$

but now with a multiplicity sequence

$$\{m_0, \ldots, m_6\} = \{1, 1, 1, 2, 1, 1, 1\}.$$

In this knot sequence, the knot $u_3 = 3$ has multiplicity two, thus $r_3 = 2$, and we only have two domain intervals.

The multiplicities of the nondomain knots do not affect the number of domain intervals. If we set

$$\{u_0, \ldots, u_7\} = \{0, 0, 0, 3, 4, 7, 7, 7\},$$

then u_0 and u_5 both have multiplicity three; however, they are listed only once each in the domain knot sequence, which is

$$\{u_2, u_3, u_4, u_5\} = \{0, 3, 4, 7\}.$$

Thus we have three domain intervals.

We now define $n + L$ *Greville abscissas* ξ_i by

$$\xi_i = \frac{1}{n}(u_i + \cdots + u_{i+n-1}); \quad i = 0, \ldots, L + n - 1. \tag{10.2}$$

The Greville abscissas are averages of the knots. The number of Greville abscissas equals the number of successive n-tuples of knots in the knot sequence.

We next assume that we are given ordinates d_i, also called *de Boor ordinates*, over the Greville abscissas and hence a polygon P consisting of the points (ξ_i, d_i); $i = 0, \ldots, L + n - 1$. This polygon is a piecewise linear function with breakpoints at the Greville abscissas. Figure 10.2 shows some examples.

We now define our basic polygon manipulation technique, the knot insertion algorithm. As before, at this point we are only concerned with polygons, not with B-spline curves! Suppose a real number $u \in [u_{n-1}, \ldots, u_{L+n-1}]$ is given and we want to *insert* it into the knot sequence. We call the new knot sequence a *refined* knot sequence. It defines a new set of Greville abscissas, called ξ_i^u. Each ξ_i^u will be the abscissa for a vertex (ξ_i^u, d_i^u) of the new polygon P^u. The knot insertion algorithm is:

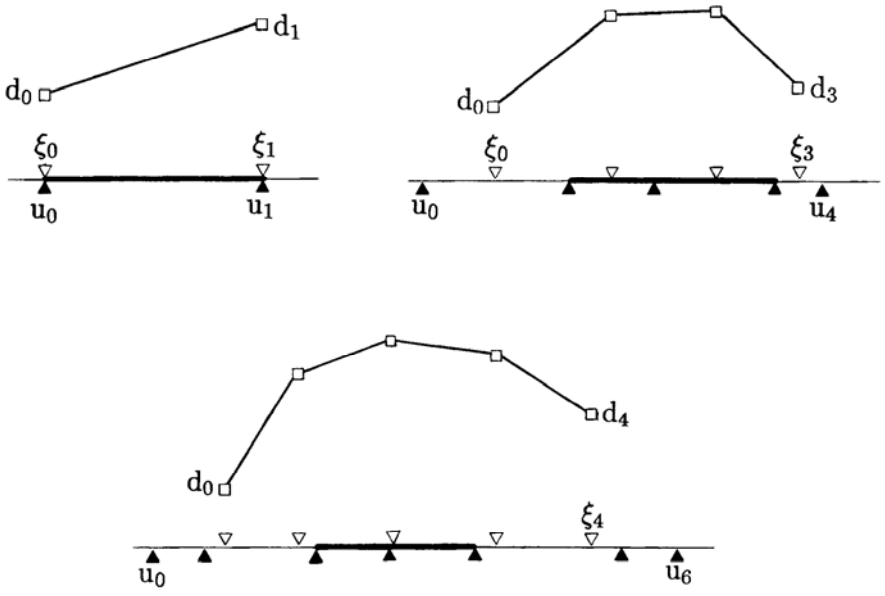

Figure 10.2: Greville abscissas: for various knot sequences and degrees, the corresponding Greville abscissas together with the polygon P are shown. Top left: $n = 1, L = 1$; top right, $n = 2, L = 2$; bottom: $n = 3, L = 2$.

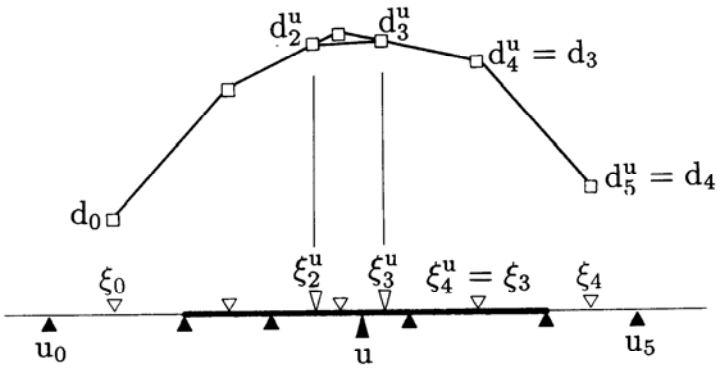

Figure 10.3: Knot insertion: the new knot is u; the new Greville abscissas are marked by larger icons. Old knot sequence: $u_0, u_1, u_2, u_3, u_4, u_5$. New sequence: $u_0, u_1, u_2, u, u_3, u_4, u_5$. In this example, $n = 2, L = 3$.

Knot insertion, informal: compute the Greville abscissas ξ_i^u for the refined knot sequence. Evaluate P there to obtain new ordinates $d_i^u = P(\xi_i^u)$. The *refined polygon* P^u is then formed by the points (ξ_i^u, d_i^u). The d_i^u are given by (10.4).

Figure 10.3 shows an example of the knot insertion procedure for the quadratic case. It is possible to insert the knot u again—it will then become a *double knot*, or a knot of multiplicity two, which simply means it is listed twice in the knot sequence. As another example of knot insertion, Figure 10.4 shows how the knot u is inserted again.

We shall formalize the knot insertion algorithm soon, but we can already deduce some properties:

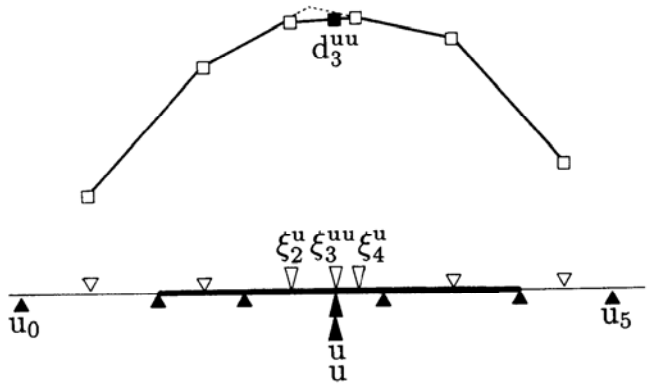

Figure 10.4: Knot insertion: the knot u is inserted again. Old knot sequence: $u_0, u_1, u_2, u, u_3, u_4, u_5$. New sequence: $u_0, u_1, u_2, u, u, u_3, u_4, u_5$.

- The polygon P^u is obtained from P by *piecewise linear interpolation* at the Greville abcissas ξ_i (see Section 2.4).
- As a consequence, knot insertion is *order independent*: if we insert two knots u and v into the knot sequence, the order in which we insert them does not matter. This follows from Menelaos' theorem of Section 2.5.
- As a further consequence, knot insertion is *variation diminishing*: no straight line intersects P^u more often than P.
- As yet another consequence, knot insertion is *convexity preserving*: if P is convex, so is P^u.
- Knot insertion is a *local* process: P differs from P^u only in the vicinity of u (the exact definition of vicinity being a function of the degree n).

We are now ready for an algorithmic definition of knot insertion. It is mostly intended for use in coding. The preceding informal description conveys the same information.

Knot insertion algorithm:

Given: $u \in [u_{n-1}, \ldots, u_{L+n-1}]$.

Find: refined polygon P^u, defined over the refined knot sequence that includes u.

1. Find the largest I with $u_I \leq u < u_{I+1}$. If $u = u_I$ and u_I is of multiplicity n : **stop**. Else:
2. For $i = 0, \ldots, I - n + 1$, set $\xi_i^u = \xi_i$.
3. For $i = I - n + 2, \ldots, I + 1$, set
$$\xi_i^u = \frac{1}{n}(u_i + \cdots + u_{i+n-2}) + \frac{1}{n}u.$$
4. For $i = I + 2, \ldots, L + n$, set $\xi_i^u = \xi_{i-1}$.
5. For $i = 0, \ldots, L + n$, set $d_i^u = P(\xi_i^u)$.
6. Renumber the knot sequence to include u as u_{I+1}.
7. Replace L by $L + 1$.

Step 5 only involves actual computation for $i = I - n + 2, \ldots, I + 1$. We now derive a formula for $P(\xi_i^u)$.

As before, let u be in the interval $[u_I, u_{I+1}]$. It is not hard to see that $\xi_{i-1} \leq \xi_i^u \leq \xi_i$. Thus d_i^u is obtained by linear interpolation:

$$d_i^u = \frac{\xi_i - \xi_i^u}{\xi_i - \xi_{i-1}} d_{i-1} + \frac{\xi_i^u - \xi_{i-1}}{\xi_i - \xi_{i-1}} d_i; \quad i = I - n + 2, \ldots, I + 1. \tag{10.3}$$

We invoke (10.2):

$$d_i^u = \frac{\sum_{j=i}^{i+n-1} u_j - \sum_{j=i}^{i+n-2} u_j - u}{u_{i+n-1} - u_{i-1}} d_{i-1} + \frac{\sum_{j=i}^{i+n-2} u_j + u - \sum_{j=i-1}^{i+n-2} u_j}{u_{i+n-1} - u_{i-1}} d_i$$

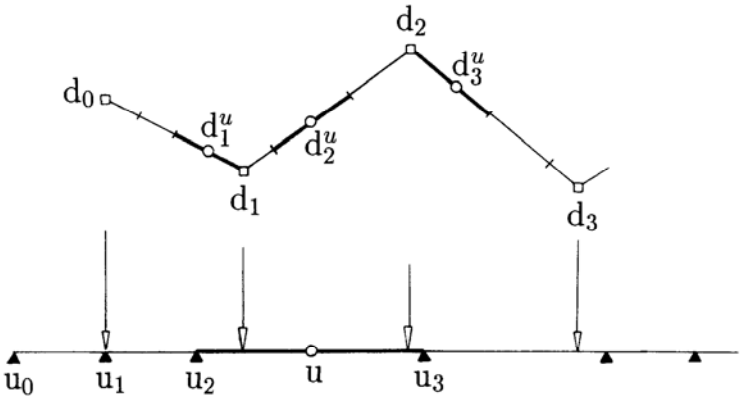

Figure 10.5: Knot insertion: this process may be interpreted as piecewise linear interpolation.

and simplify:

$$d_i^u = \frac{u_{i+n-1} - u}{u_{i+n-1} - u_{i-1}} d_{i-1} + \frac{u - u_{i-1}}{u_{i+n-1} - u_{i-1}} d_i; \quad i = I - n + 2, \ldots, I + 1. \quad (10.4)$$

This is our desired knot insertion formula.

It has the form of linear interpolation, and we recall from Section 2.3 that this may be interpreted as an affine map. In our case, we would map the interval $[u_{i-1}, u_{i+n-1}]$ onto the polygon leg defined by (ξ_{i-1}, d_{i-1}) and (ξ_i, d_i), as shown in Figure 10.5 for the cubic case.

10.3 The de Boor Algorithm

In the previous section, we described an operation to manipulate polygons. We shall now use this operation for the definition of B-spline curves. Recall Figure 10.4, in which a knot u was reinserted so that its multiplicity was raised to two. What happens if we reinsert u again? The answer: nothing. No new Greville abscissas are generated.

In general, for degree n, repeated insertion of a knot u no longer changes the polygon after the multiplicity of u has reached n. We use this fact in the algorithmic definition of a special function, called a B-spline curve.[4] The algorithm used in this definition is called the de Boor algorithm:

de Boor algorithm, informal: To evaluate an n^{th}-degree B-spline curve (given by its de Boor ordinates and knot sequence) at a parameter value u, insert u into the knot sequence until it has multiplicity n. The corresponding polygon vertex is the desired function value.

[4]We use the term "curve" loosely to emphasize that the theory developed here carries over easily to parametric curves.

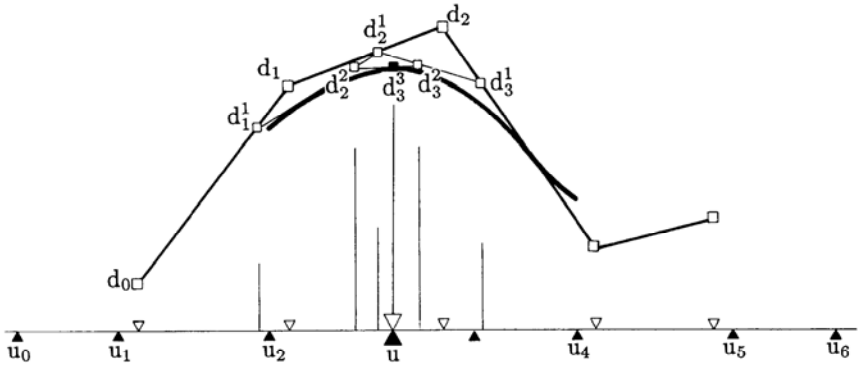

Figure 10.6: The de Boor algorithm: example with $n = 3, L = 2$.

Before we proceed further, one comment should be made. What is meant by "corresponding polygon vertex?" If a knot u_i is of multiplicity n, then one of the Greville abscissas coincides with u_i, namely, $\xi_i = \frac{1}{n}(u_i + \cdots + u_{i+n-1}) = u_i$. Consequently, the polygon has a vertex (u_i, d_i), and d_i is the function value of the B-spline curve at u_i. Figure 10.6 gives an illustration. We now realize that we have encountered an example of the de Boor algorithm earlier; see Figure 10.4 for the case $n = 2$.

Note that the de Boor algorithm needs fewer insertions if the parameter value u is already an element of the knot sequence. If it has multiplicity r, then only $n - r$ reinsertions are necessary to make u a knot of multiplicity n.

We are now ready for a formal definition. Let us denote a B-spline curve of degree n with control polygon P by B_nP, and its value at parameter value u by $[B_nP](u)$. We will only define the curve for values of u between u_{n-1} and u_{L+n-1}.

de Boor algorithm: Let $u \in [u_I, u_{I+1}) \subset [u_{n-1}, u_{L+n-1}]$. Define

$$d_i^k(u) = \frac{u_{i+n-k} - u}{u_{i+n-k} - u_{i-1}} d_{i-1}^{k-1}(u) + \frac{u - u_{i-1}}{u_{i+n-k} - u_{i-1}} d_i^{k-1}(u) \tag{10.5}$$

for $k = 1, \ldots, n - r$, and $i = I - n + k + 1, \ldots, I - r + 1$. Then

$$s(u) = [B_nP](u) = d_{I-r+1}^{n-r}(u) \tag{10.6}$$

is the value of the B-spline curve at parameter value u. Here, r denotes the multiplicity of u if it was already one of the knots. If it was not, set $r = 0$. As usual, we set $d_i^0(u) = d_i$.

C. de Boor [125] published this algorithm in 1972. It is the B-spline analogue of the de Casteljau algorithm. Figure 10.7 shows schematically which d_i are involved in (10.5).

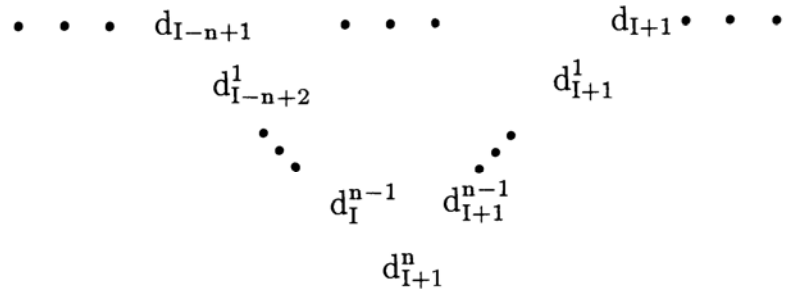

Figure 10.7: The de Boor algorithm: for $u \in [u_I, u_{I+1}]$, the scheme of generated intermediate points is shown, assuming that u was not one of the existing knots.

In our description of the de Boor algorithm, we did not renumber the knot sequence and the control points at each level, since our interest is only in the final result $d^{n-r}_{I-r+1}(u)$. Of course, at each level k, we generate a new control polygon that describes the same B-spline curve as did the previous control polygon. In particular, for $k = 1$, we obtain the knot insertion algorithm.

Figure 10.6 shows an example. We can also view that example as a case of multiple knot insertion. In that context, we have constructed several polygons that describe the same B-spline curve:

k=1: the de Boor ordinates $d_0, d^1_1, d^1_2, d^1_3, d_3, d_4$ corresponding to the knot sequence $u_0, u_1, u_2, u, u_3, u_4, u_5, u_6$;

k=2: the de Boor ordinates $d_0, d^1_1, d^2_2, d^2_3, d^1_3, d_3, d_4$ corresponding to the knot sequence $u_0, u_1, u_2, u, u, u_3, u_4, u_5, u_6$;

k=3: the de Boor ordinates $d_0, d^1_1, d^2_2, d^3_3, d^2_3, d^1_3, d_3, d_4$ corresponding to the knot sequence $u_0, u_1, u_2, u, u, u, u_3, u_4, u_5, u_6$.

Let us next examine an important special case. Consider the knot sequence

$$0 = u_0 = u_1 = \cdots = u_{n-1} < u_n = u_{n+1} = \cdots = u_{2n-1} = 1.$$

Here, both u_0 and u_n have multiplicity n. We note that the Greville abscissas are given by

$$\xi_i = \frac{1}{n} \sum_{j=i}^{i+n-1} u_j = \frac{i}{n}; \quad i = 0, \ldots, n.$$

For $0 \le u \le 1$, the de Boor algorithm sets $I = n - 1$ and

$$d^k_i(u) = \frac{u_{i+n-k} - u}{u_{i+n-k} - u_{i-1}} d^{k-1}_{i-1} + \frac{u - u_{i-1}}{u_{i+n-k} - u_{i-1}} d^{k-1}_i.$$

Since $n - k \ge i - k \ge 0$, we have $u_{i+n-k} = 1$, $u_{i-1} = 0$ for all i, k; thus

$$d^k_i(u) = (1 - u) d^{k-1}_{i-1} + u d^{k-1}_i; \quad k = 1, \ldots, n. \tag{10.7}$$

This is the de Casteljau algorithm![5] Schoenberg [451] first observed this in 1967, although in a different context. Riesenfeld [423] and Gordon and Riesenfeld [249] are more accessible references. We will be able to draw several important conclusions from this special case. First, we note that the restriction to the interval $[0, 1]$ is not essential: all our constructions are invariant under affine parameter transformations.

Thus, if two adjacent knots in any knot sequence both have multiplicity n, the corresponding B-spline curve is a Bézier curve between those two knots. The B-spline control polygon is the Bézier polygon; the Greville abscissas are equally spaced between the two knots.

After we inserted u until it was of multiplicity n, the initial de Boor polygon (or Bézier polygon, in this case) was transformed into two Bézier polygons, defining the same curve as did the initial polygon. Thus we have another proof for the fact that the de Casteljau algorithm subdivides Bézier curves.

For a B-spline curve over an arbitrary knot sequence, we can always reinsert the given knots until each knot is of multiplicity n. The B-spline polygon corresponding to that knot sequence is the *piecewise Bézier polygon* of the curve. We have thus shown that *B-spline curves are piecewise polynomial over* $[u_{n-1}, u_{L+n-1}]$. The method of constructing the piecewise Bézier polygon from the B-spline polygon via knot insertion was developed by W. Boehm [63]. A different method was created by P. Sablonnière [432]. We will give a concise algorithm later, in the context of blossoms; see (10.18).

10.4 Smoothness of B-spline Curves

Now that we know that B-spline curves are piecewise polynomials of degree n each, we shall investigate their smoothness: how often is a B-spline curve differentiable at a point u? Obviously, we need to consider only the knots u_i—the curve is infinitely often differentiable at all other points.

To answer this question, simply reconsider the preceding example of (10.7). Now, let u be an existing knot of multiplicity r. Our knot sequence is:

$$\begin{aligned} 0 &= u_0 = u_1 = \cdots = u_{n-1} \\ &< u_n = u_{n+1} = \cdots = u_{n+r-1} \\ &< u_{n+r} = u_{n+r+1} = \cdots = u_{2n+r-1} = 1; \end{aligned}$$

the knot to be reinserted is $u = u_n$. The de Boor algorithm only consists of $n - r$ levels. Taking into account the multiplicities of the end knots, we have

$$d_i^k(u) = (1 - u)d_{i-1}^{k-1} + ud_i^{k-1}; \quad k = 1, \ldots, n - r. \tag{10.8}$$

These are the $n - r$ last levels in a de Casteljau algorithm. Therefore the two polynomial curve segments meeting at u are at least $n - r$ times differentiable at that point (see Sections 4.5 and 4.6).

[5] The subscripts are different—but this is simply a matter of notation.

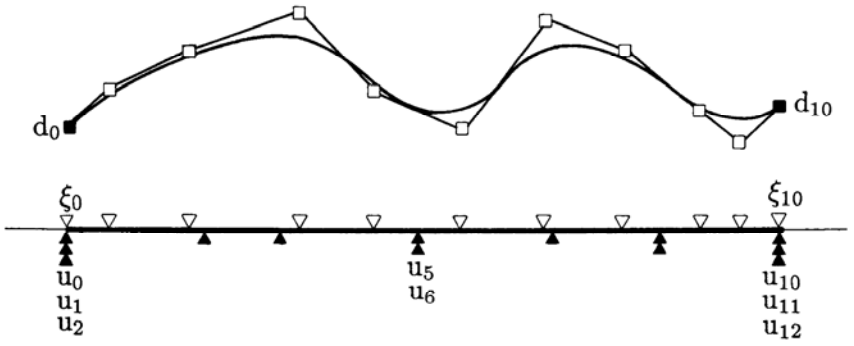

Figure 10.8: Multiple knots: the effects of multiple knots on the curve. Here, $n = 3, L = 8$.

As above, we note that the restriction to the interval [0, 1] is not essential. If we want to investigate the smoothness of an arbitrary B-spline curve at a knot, we can always force its two neighbors to be of multiplicity n (without changing the curve!) and apply our arguments.

Thus *a B-spline curve is (at least) C^{n-r} at knots with multiplicity r*. In particular, the curve is $n - 1$ times continuously differentiable if all knots are simple, i.e., of multiplicity one. Figure 10.8 shows a cubic ($n = 3$) B-spline curve over a knot sequence that has several multiple entries. The triple knots at the ends force d_0 and d_{10} to be on the curve.

10.5 The B-spline Basis

Consider a knot sequence $u_0, \ldots, u_K$ and the set of piecewise polynomials of degree n defined over it, where each function in that set is $n - r_i$ times continuously differentiable at knot u_i. All these piecewise polynomials form a linear space, with dimension

$$\dim = (n + 1) + \sum_{i=1}^{K-1} r_i. \tag{10.9}$$

For a proof, suppose we want to construct an element of our piecewise polynomial linear space. The number of independent constraints that we can impose on an arbitrary element, or its number of *degrees of freedom*, is equal to the dimension of the considered linear space. We may start by completely specifying the first polynomial segment, defined over $[u_0, u_1]$; we can do this in $n + 1$ ways, which is the number of coefficients that we can specify for a polynomial of degree n. The next polynomial segment, defined over $[u_1, u_2]$, must agree with the first segment in

position and $n - r_1$ derivatives at u_1, thus leaving only r_1 coefficients to be chosen for the second segment. Continuing further, we obtain (10.9).

We are interested in B-spline curves that are piecewise polynomials over the special knot sequence $[u_{n-1}, u_{L+n-1}]$. The dimension of the linear space that they form is $L + n$, which also happens to be the number of B-spline vertices for a curve in this space. If we can define $L + n$ linearly independent piecewise polynomials in our linear function space, we have found a basis for this space. We proceed as follows.

Define functions $N_i^n(u)$, called *B-splines*, by defining their de Boor ordinates to satisfy $d_i = 1$ and $d_j = 0$ for all $j \neq i$. The $N_i^n(u)$ are clearly elements of the linear space formed by all piecewise polynomials over $[u_{n-1}, u_{L+n-1}]$. They have *local support*:

$$N_i^n(u) \neq 0 \text{ only if } u \in [u_{i-1}, u_{i+n}].$$

This follows because knot insertion, and hence the de Boor algorithm, is a local operation; if a new knot is inserted, only those Greville abscissas that are "close" will be affected.

B-splines also have *minimal support:* if a piecewise polynomial with the same smoothness properties over the same knot vector has less support than N_i^n, it must be the zero function. All piecewise polynomials defined over $[u_{i-1}, u_{i+n}]$, the support region of N_i^n, are elements of a function space of dimension $2n + 1$, according to (10.9). A support region that is one interval "shorter" defines a function space of dimension $2n$. The requirement of vanishing $n - r_{i-1}$ derivatives at u_{i-1} and of vanishing $n - r_{i+n}$ derivatives at u_{i+n} imposes $2n$ conditions on any element in the linear space of functions over $[u_{i-1}, u_{i+n-1}]$. The additional requirement of assuming a nonzero value at some point in the support region raises the number of independent constraints to $2n + 1$, too many to be satisfied by an element of the function space with dimension $2n$.

Another important property of the N_i^n is their *linear independence.* To demonstrate this independence, we must verify that

$$\sum_{j=0}^{L+n-1} c_j N_j^n(u) \equiv 0 \tag{10.10}$$

implies $c_j = 0$ for all j. It is sufficient to concentrate on one interval $[u_I, u_{I+1}]$ with $u_I < u_{I+1}$. Because of the local support property of B-splines, (10.10) reduces to

$$\sum_{j=I-n+1}^{I+1} c_j N_j^n(u) \equiv 0 \text{ for } u \in [u_I, u_{I+1}].$$

We have completed our proof if we can show that the linear space of piecewise polynomials defined over $[u_{I-n}, u_{I+n+1}]$ does not contain a nonzero element that vanishes over $[u_I, u_{I+1}]$. Such a piecewise polynomial cannot exist: it would have to be a nonzero local support function over $[u_{I+1}, u_{I+n+1}]$. The existence of such a function would contradict the fact that B-splines are of *minimal* local support.

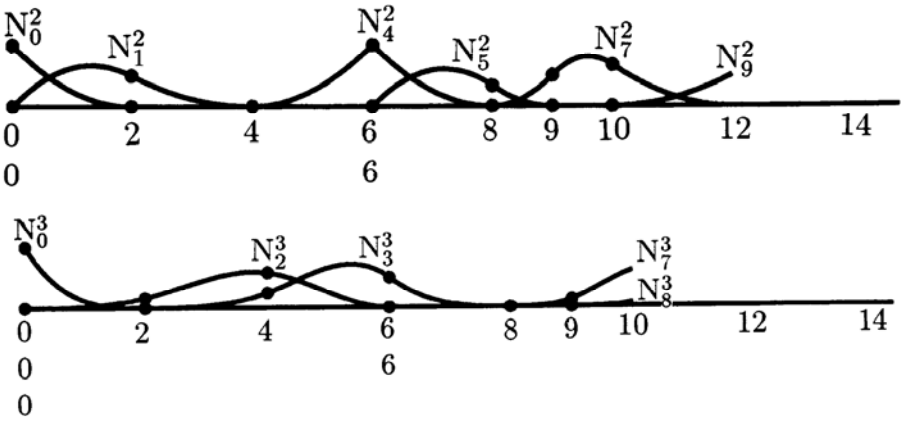

Figure 10.9: B-splines: top, some quadratic B-splines over the indicated knot sequence; bottom: some cubic ones. Note multiple knots at left end and simple knots at right end.

Because the B-splines N_i^n are linearly independent, every piecewise polynomial s over $[u_{n-1}, u_{L+n-1}]$ may be written uniquely in the form

$$s(u) = \sum_{j=0}^{L+n-1} d_j N_j^n(u). \tag{10.11}$$

The B-splines thus form a *basis* for this space. This reveals the origin of their name, which is short for *B*asis splines.

If we set all $d_i = 1$ in (10.11), the function $s(u)$ will be identically equal to one, thus asserting that B-splines form a *partition of unity*.

Figure 10.9 gives examples of quadratic and cubic basis functions.

10.6 Two Recursion Formulas

We have defined B-spline basis functions in a constructive way: the B-spline N_i^n is defined by the knot sequence and the Greville abscissa ξ_i. The function N_i^n is given by its B-spline control polygon with de Boor ordinates $d_j = \delta_{i,j}$; $j = 0, \ldots, L + n - 1$. From it, we can construct the piecewise Bézier polygon by inserting every knot until it is of multiplicity n. We can then compute values of $N_i^n(u)$ by applying the de Casteljau algorithm to the Bézier polygon corresponding to the interval that contains u. There is a more direct way, which we now discuss.

To further explore B-splines, let us investigate how they "react" to knot insertion. Let $\hat{u}$ be a new knot inserted into a given knot sequence. Denote the B-splines over the "old" knot sequence by N_i^n, and those over the "new" knot sequence by $\hat{N}_i^n$. Note that there is one more element in the set of $\hat{N}_i^n$ than in that of the N_i^n. In fact, the

linear space of all piecewise polynomials over the old knot sequence is a subspace of the linear space of all piecewise polynomials over the new sequence. Let N_l^n be an "old" basis function that has $\hat{u}$ in its support. Its B-spline polygon is defined by $d_j = \delta_{j,l}$, where j ranges from 0 to $L + n - 1$ and δ denotes the Kronecker delta. Its B-spline polygon with respect to the new knot sequence is obtained by the knot insertion process.

Only two of the new de Boor ordinates will be different from zero. Equation (10.4) yields

$$\hat{d}_l = \frac{u_{l+n-1} - \hat{u}}{u_{l+n-1} - u_{l-1}} \cdot 0 + \frac{\hat{u} - u_{l-1}}{u_{l+n-1} - u_{l-1}} \cdot 1,$$

$$\hat{d}_{l+1} = \frac{u_{l+n} - \hat{u}}{u_{l+n} - u_l} \cdot 1 + \frac{\hat{u} - u_l}{u_{l+n} - u_l} \cdot 0.$$

(Recall that $d_l = 1$, whereas all other $d_j = 0$.) Hence

$$\hat{d}_l = \frac{\hat{u} - u_{l-1}}{u_{l+n-1} - u_{l-1}},$$

$$\hat{d}_{l+1} = \frac{u_{l+n} - \hat{u}}{u_{l+n} - u_l}.$$

Thus we can write N_l^n in terms of $\hat{N}_l^n$ and $\hat{N}_{l+1}^n$:

$$N_l^n(u) = \frac{\hat{u} - u_{l-1}}{u_{l+n-1} - u_{l-1}} \hat{N}_l^n(u) + \frac{u_{l+n} - \hat{u}}{u_{l+n} - u_l} \hat{N}_{l+1}^n(u). \tag{10.12}$$

This result is due to W. Boehm [62]. It allows us to write B-splines as linear combinations of B-splines over a refined knot sequence.

For the second important recursion formula, we must define an additional B-spline function, N_i^0:

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_{i-1} \leq u < u_i \\ 0 & \text{else} \end{cases}. \tag{10.13}$$

The announced recursion formula relates B-splines of degree n to B-splines of degree $n - 1$:

$$N_l^n(u) = \frac{u - u_{l-1}}{u_{l+n-1} - u_{l-1}} N_l^{n-1}(u) + \frac{u_{l+n} - u}{u_{l+n} - u_l} N_{l+1}^{n-1}(u). \tag{10.14}$$

To prove (10.14), we shall prove the following more general statement:

$$s(u) = \sum_{j=i+r-n+1}^{i+1} d_j^r N_j^{n-r}(u) \tag{10.15}$$

for all $r \in [0, n]$. For its proof, we first check that it is true for $r = n$; this follows from (10.13). By the de Boor algorithm, (10.15) is equivalent to

$$s(u) = \sum_{j=i-n+1+r}^{i+1} (1 - \alpha_j^r) d_{j-1}^{r-1} N_j^{n-r}(u) + \sum_{j=i-n+1+r}^{i+1} \alpha_j^r d_j^{r-1} N_j^{n-r}(u),$$

where

$$\alpha_j^r = \frac{u - u_{i-1}}{u_{i+n-r} - u_{i-1}}.$$

An index transformation yields

$$s(u) = \sum_{j=i-n+r}^{i} (1 - \alpha_{j+1}^r) d_j^{r-1} N_{j+1}^{n-r}(u) + \sum_{j=i-n+1+r}^{i+1} \alpha_j^r d_j^{r-1} N_j^{n-r}(u).$$

Because of the local support of the N_j^{n-r}, this may be changed to

$$s(u) = \sum_{j=i-n+r}^{i+1} (1 - \alpha_{j+1}^r) d_j^{r-1} N_{j+1}^{n-r}(u) + \sum_{j=i-n+r}^{i+1} \alpha_j^r d_j^{r-1} N_j^{n-r}(u).$$

Hence, by the inductive hypothesis,

$$s(u) = \sum_{j=i-n+r}^{i+1} [\alpha_j^r N_j^{n-r}(u) + (1 - \alpha_{j+1}^r) N_{j+1}^{n-r}(u)] d_j^{r-1}.$$

This step completes the proof of (10.15), since we have now shown that (10.15) holds for $r - 1$ provided that it holds for r. The recurrence (10.14) now follows from comparing (10.15) and (10.6). The development of equation (10.14) is due to L. Mansfield, C. de Boor, and M. Cox; see de Boor [125] and Cox [118]. For an illustration of (10.14), see Figure 10.10.

The recursion formula (10.14) shows that a B-spline of degree n is a strictly convex combination of two lower degree ones; it is therefore a very stable formula from a numerical viewpoint. If B-spline curves must be evaluated repeatedly at the

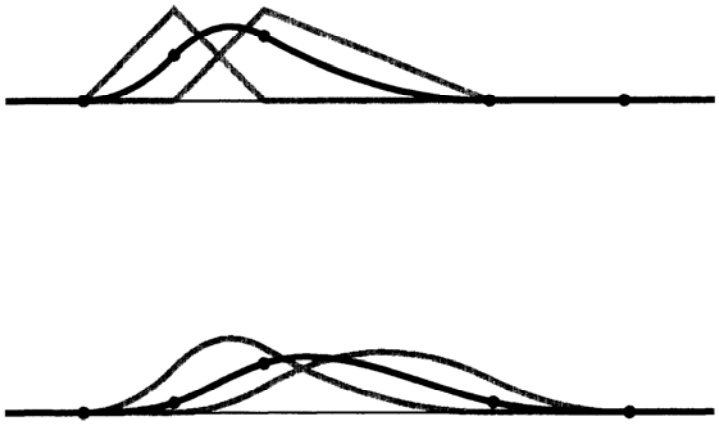

Figure 10.10: The B-spline recursion: top, two linear B-splines yield a quadratic one; bottom, two quadratic B-splines yield a cubic one.

same parameter values u_k, it is a good idea to compute the values for $N_i^n(u_k)$ using (10.14) and then to store them.

A comment on end knot multiplicities: the widespread data format IGES uses two additional knots at the ends of the knot sequence; in our terms, it adds knots u_{-1} and u_{L+2n-1}. The reason is that formulas such as (10.14) seemingly require the presence of these knots. As they only are multiplied by zero factors, their values have no influence on any computation. There is no reason to store completely inconsequential data; hence the "leaner" notation of this chapter.

10.7 Repeated Knot Insertion

We may insert more and more knots into the knot sequence; let us now investigate the effect of such a process. A B-spline curve of degree n is defined over $u_{n-1}, \ldots, u_{L+n-1}$. Let us set $a = u_{n-1}$, $b = u_{L+n-1}$. Now insert more knots u_i^r into $[a, b]$; here r counts the overall number of insertions and i denotes the number of u_i^r in the new knot sequence. After r knot insertions, we have a new polygon P^r that describes the same curve as did the original polygon P. As we insert more and more knots, so as to become dense in $[a, b]$, the sequence of polygons P^r converges to the curve that they all define:

$$\lim_{r\to\infty} P^r = [B_n P]. \tag{10.16}$$

To begin, we recall that a B-spline curve depends only on $d_k, \ldots, d_{k+n}$ over the interval $[u_k, u_{k+1}]$. Then for $u \in [u_k, u_{k+1}]$,

$$\min(d_k, \ldots, d_{k+n}) \le [B_n P](u) \le \max(d_k, \ldots, d_{k+n})$$

by the strong convex hull property.

We need to show that for any ϵ, we can find an r such that

$$|P^r(u) - [B_n P](u)| \le \epsilon \text{ for all } u.$$

We know that for any ϵ, we can find an r such that

$$|\xi_{i+1}^r - \xi_i^r| \le \delta$$

and

$$|P^r(\xi_{i+1}^r) - P^r(\xi_i^r)| \le \epsilon,$$

since each P^r is continuous. Thus,

$$\max[P^r(\xi_i^r), \ldots, P^r(\xi_{i+n}^r)] - \min[P^r(\xi_i^r), \ldots, P^r(\xi_{i+n}^r)] \le n\epsilon$$

for those i that are relevant to the interval $[u_k, u_{k+1}]$. But we also know that

$$\min(d_i^r) \le [B_n P](u) \le \max(d_i^r).$$

Thus,

$$|[B_n P](u) - P_j^r| \leq n\epsilon; \quad j \in [i, \ldots, i+n],$$

which finally yields

$$|[B_n P](u) - P^r(u)| \leq n\epsilon,$$

proving our convergence claim.

The use of repeated knot insertion lies in the *rendering* of B-spline curves. If sufficiently many knots have been inserted into the knot sequence, the resulting control polygon will be arbitrarily close to the curve. Then, instead of plotting the curve directly, one simply plots the refined polygon. To obtain an *adaptive* rendering method, one would control the knot insertion process by inserting more knots where the curve is of high curvature and fewer knots where it is flat.

Of course, B-spline curves may also be *parametric*. All we have to do is use functional B-spline curves (all over the same knot vector) for each component of the parametric curve:

$$\mathbf{x}(u) = \sum_{i=0}^{L+n-1} \mathbf{d}_i N_i^n(u) = \sum_{i=0}^{L+n-1} \begin{bmatrix} d_i^x \\ d_i^y \\ d_i^z \end{bmatrix} N_i^n(u).$$

The curves for $n = 2$ and $n = 3$ were already described in Chapter 7, although with a different notation that especially suited those cases. General degree B-spline curves enjoy all the properties of the lower degree ones, such as affine invariance and the convex hull property.

An interesting application of repeated knot insertion is due to G. Chaikin [95]. Although this scheme may be described in the context of functional curves, we prefer the more intuitive parametric version. Consider a quadratic B-spline curve over a uniform knot sequence. Insert a new knot at the midpoint of every interval of the knot sequence. If the "old" curve had control vertices $\mathbf{d}_i$ and those of the new one are $\mathbf{d}_i^{(1)}$, it is easy to show that

$$\mathbf{d}_{2i-1}^{(1)} = \frac{3}{4}\mathbf{d}_i + \frac{1}{4}\mathbf{d}_{i-1} \quad \text{and} \quad \mathbf{d}_{2i+1}^{(1)} = \frac{3}{4}\mathbf{d}_i + \frac{1}{4}\mathbf{d}_{i+1}.$$

If this procedure is repeated infinitely often, the resulting sequence of polygons will converge to the original curve, as follows from our previous considerations. Figure 10.11 shows the example of a closed quadratic B-spline curve; two levels of the iteration are shown.

Chaikin's algorithm may be described as *corner-cutting*: At each step, we chisel away the corners of the initial polygon. This process is, on a high level, similar to that of degree elevation for Bézier curves, which is also a convergent process (see Section 5.2). One may ask if corner-cutting processes will always converge to a smooth curve. The answer is "yes," with some mild stipulations on the corner-cutting process; this was first proved by de Boor [128]. One may thus use a corner-cutting procedure to *define* a curve—and only very few of the curves thus generated are piecewise

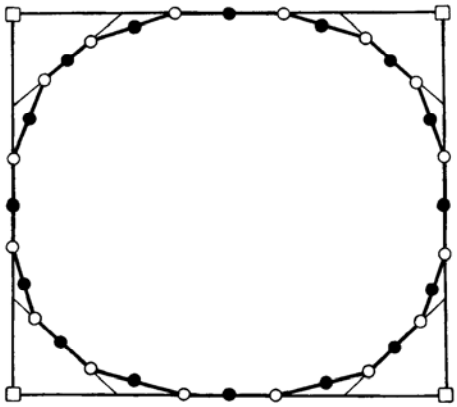

Figure 10.11: Chaikin's algorithm: starting with the (closed) control polygon of a quadratic B-spline curve, two levels of the Chaikin iteration are shown.

polynomial! Recent work has been carried out by Prautzsch and Micchelli [412] and [358], based on earlier results by de Rham [137], [138].

Corner-cutting may also be used for interpolation; see Dyn, Levin, and Gregory [160], [159].

R. Riesenfeld [424] realized that Chaikin's algorithm actually generates uniform quadratic B-spline curves. A general algorithm for the simultaneous insertion of several knots into a B-spline curve has been developed by Cohen, Lyche, and Riesenfeld [110]. This so-called "Oslo algorithm" needs a theory of discrete B-splines for its development (see Bartels *et al.* [42]). The knot insertion algorithm as developed in this chapter is more intuitive and equally powerful.

10.8 B-spline Properties

After the more theoretical developments of the previous two sections, let us examine some of the properties that we can now derive for B-spline curves.

Linear precision: If $l(u)$ is a straight line of the form $l = au + b$, and if we read off values at the Greville abscissas, the resulting B-spline curve reproduces the straight line:

$$\sum l(\xi_i) N_i^n(u) = l(u).$$

This property is a direct consequence of the de Boor algorithm. It was originally obtained by T. Greville [261], [260] in a different context. The original Greville result is the motivation for the term "Greville abscissas."

Strong convex hull property: Each point on the curve lies in the convex hull of no more than $n + 1$ nearby control points.

Variation diminishing property: The curve is not intersected by any straight line more often than is the polygon. This result has a very simple proof, presented by Lane and Riesenfeld [320]: we may insert every knot until it is of full multiplicity. This is a variation diminishing process, since it is piecewise linear interpolation. Once all knots are of full multiplicity, the B-spline control polygon is the piecewise Bézier polygon, for which we showed the variation diminishing property in Section 5.3.

The parametric case: In the parametric case, it is desirable to have u_0 and u_{L+n-1} both of full multiplicity n. This condition forces the first and last control points $\mathbf{d}_0$ and $\mathbf{d}_{L+n-1}$ to lie on the endpoints of the curve. In this way, one has better control of the behavior of the curve at the ends. The spline curves that we discussed in Chapter 7 are all described in this form, although we did not formally make use of knot multiplicities there. If the end knots are allowed to be of lower (even simple) multiplicity, the first and last control vertices do not lie on the curve, and are called "phantom vertices" by Barsky and Thomas [40]. Figure 10.12 gives several examples of B-spline curves over various knot sequences, all with $L = 7$.

The de Boor algorithm allows a nice geometric description in parametric form. Formally, we perform the algorithm for all components of the control polygon. Geometrically, we may "engrave" parts of the knot sequence on each polygon leg: map the first $n + 1$ subsequent knots (starting at u_0) onto $\mathbf{d}_0\mathbf{d}_1$, the next subsequent $n + 1$ knots onto $\mathbf{d}_1\mathbf{d}_2$, and so on, until the last subsequent $n + 1$ knots are mapped to the last polygon leg. For example, in Figure 10.13, with $n = 3$ and $L = 5$, the knots $[u_2, u_3, u_4, u_5]$ are mapped to $\mathbf{d}_2\mathbf{d}_3$, whereas $[u_0, u_1, u_2, u_3]$ are mapped to $\mathbf{d}_0\mathbf{d}_1$. The interval $[u_I, u_{I+1}]$, which contains the evaluation parameter u, is thus mapped to n polygon legs by n affine maps, which are equivalent to linear interpolation as outlined in Section 2.3. These affine maps take u to the $\mathbf{d}_j^1(u)$. The same procedure is then repeated: Consider all sets of subsequent n knots that contain u_I, u_{I+1}. Map them by affine maps to the polygon formed by the $\mathbf{d}_j^1$ and denote the images of u by $\mathbf{d}_j^2$, etc. The final step is to map the interval u_I, u_{I+1} onto $\mathbf{d}_I^{n-1}, \mathbf{d}_{I+1}^{n-1}$ to obtain the point $\mathbf{d}_{I+1}^n$ on the curve.

Finally, a note on how to *store* B-spline curves. It is not wise to store the knot sequence $\{u_i\}$ and simply list multiple knots as often as indicated by their multiplicity. Roundoff may produce knots that are a small distance apart, yet meant to be identical. Following the approach taken by Schumaker [452], it is wiser to store only distinct knots and to note their multiplicities in a second array. From these two arrays, one may compute the original knot sequence when required—for example, for the de Boor algorithm.

10.9 B-spline Blossoms

In Section 3.4, we generalized the de Casteljau algorithm by allowing evaluations at n different arguments $t_1, \ldots, t_n$, thus arriving at the blossom $\mathbf{b}[t_1, \ldots, t_n]$ of a

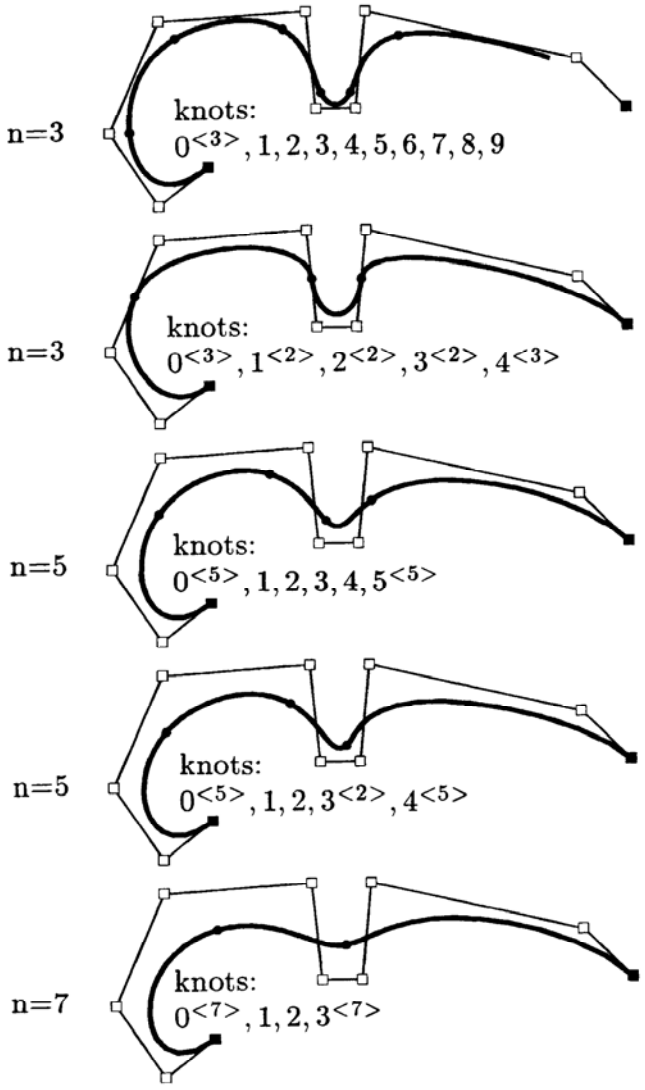

Figure 10.12: Parametric B-spline curves: several examples, all with the same control polygon but with different degrees and knot sequences.

polynomial curve $\mathbf{b}(t)$. The same principle may be applied to B-spline curves. For the sake of concreteness, let us begin with the case of a cubic B-spline curve, and also restrict ourselves to the parameter interval $[u_4, u_5]$.

We will now modify the standard de Boor algorithm: at each level $k; k = 1, 2, 3$, we will evaluate at a different argument $v_k \in [u_4, u_5]$. The relevant control points for

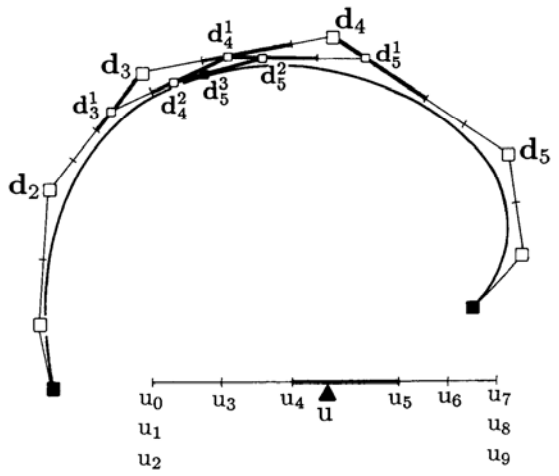

Figure 10.13: The de Boor algorithm in parametric form: all groups of $n + 1 - r$ intervals that contain $[u_4, u_5]$ are mapped onto polygon legs.

our interval are $\mathbf{d}_2, \ldots, \mathbf{d}_5$, and we obtain the following scheme:

$$\begin{array}{llll} \mathbf{d}_2 & & & \\ \mathbf{d}_3 & \mathbf{d}_3^1[v_1] & & \\ \mathbf{d}_4 & \mathbf{d}_4^1[v_1] & \mathbf{d}_4^2[v_1, v_2] & \\ \mathbf{d}_5 & \mathbf{d}_5^1[v_1] & \mathbf{d}_5^2[v_1, v_2] & \mathbf{d}_5^3[v_1, v_2, v_3]. \end{array}$$

Again, it is easy to see that it does not matter in which order we "feed" the the v_i into this scheme. Also, if all v_i agree, we recover the standard de Boor algorithm. We shall use the notation $\mathbf{d}_4[v_1, v_2, v_3]$ for the point $\mathbf{d}_5^3[v_1, v_2, v_3]$, indicating that we are dealing with the interval $[u_4, u_5]$.

In general, for a parameter interval $[u_I, u_{I+1}]$, we shall define as the *B-spline blossom*—or just blossom—the function $\mathbf{d}_I[v_1, \ldots, v_n]$, obtained by applying the de Boor algorithm for the interval $[u_I, u_{I+1}]$ to the control points $\mathbf{d}_{I-n+1}, \ldots, \mathbf{d}_{I+1}$, and using a (possibly) different argument v_k at level k of the algorithm. Thus the blossom corresponding to the interval $[u_I, u_{I+1}]$ is a function of n variables $v_1, \ldots, v_n$; in fact, it is a multivariate *polynomial* function. The whole B-spline curve possesses as many blossoms as it has domain intervals.

Note that we have not restricted the v_i to be in the interval $[u_I, u_{I+1}]$! In fact, a very interesting result arises if we evaluate for v_i outside that interval. Returning to our earlier cubic example, let us set $[v_1, v_2, v_3] = [u_2, u_3, u_4]$. The scheme becomes:

$$\begin{array}{llll} \mathbf{d}_2 & & & \\ \mathbf{d}_3 & \mathbf{d}_2 & & \\ \mathbf{d}_4 & \bullet & \mathbf{d}_2 & \\ \mathbf{d}_5 & \bullet & \bullet & \mathbf{d}_2 = \mathbf{d}_4[u_2, u_3, u_4]. \end{array}$$

The •-entries in the scheme were not computed, because their values do not contribute to the final result. We have, similarly to the Bézier case, recovered one of the control points! The algorithm no longer uses only convex combinations; instead, extrapolation is used several times.

To illustrate the principle of control point recovery, we try one more set of arguments, namely $[v_1, v_2, v_3] = [u_3, u_4, u_5]$. The scheme becomes:

$$\begin{array}{llll} \mathbf{d}_2 & & & \\ \mathbf{d}_3 & \bullet & & \\ \mathbf{d}_4 & \mathbf{d}_3 & \bullet & \\ \mathbf{d}_5 & \bullet & \mathbf{d}_3 & \mathbf{d}_3 = \mathbf{d}_4[u_3, u_4, u_5]. \end{array}$$

Again, we have recovered a control point. Continuing in this manner, we find that $\mathbf{d}_4 = \mathbf{d}_4[u_4, u_5, u_6]$ and $\mathbf{d}_5 = \mathbf{d}_4[u_5, u_6, u_7]$. Figure 10.14 illustrates these examples.

More generally, we have the following: the curve segment defined over $[u_I, u_{I+1}]$ needs $n + 1$ control points for its definition, namely $\mathbf{d}_{I-n+1}, \ldots, \mathbf{d}_{I+1}$. They can be obtained as blossom values:

$$\mathbf{d}_{I-n+1+k} = \mathbf{d}_I[u_{I-n+1+k}, \ldots, u_{I+k}]; \quad k = 0, \ldots, n. \tag{10.17}$$

The arguments of $\mathbf{d}_I$ on the right-hand side are all n-tuples of subsequent knots that contain either u_I or u_{I+1}.

As a spinoff, we can give a very compact formula for the *conversion of a B-spline curve into its piecewise Bézier form*: let the Bézier points corresponding to the interval $[u_I, u_{I+1}]$ be $\mathbf{b}_0^I, \ldots, \mathbf{b}_n^I$. They are given by

$$\mathbf{b}_j^I = \mathbf{d}_I[u_I^{<n-j>}, u_{I+1}^{<j>}]; \quad j = 0, \ldots, n. \tag{10.18}$$

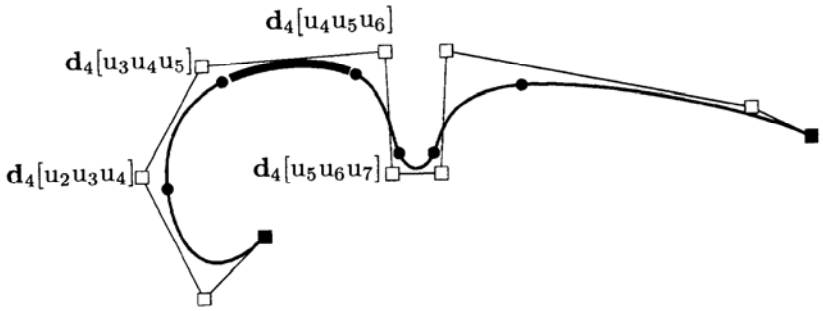

Figure 10.14: B-spline blossoms (cubic): The knot sequence is $u_0 = u_1 = u_2 < u_3 < u_4 < u_5 < u_6 < u_7 < u_8 = u_9 = u_{10}$. The control points corresponding to $[u_4, u_5]$ are shown as blossom values.

The simplicity of this formula is striking; in former days, involved papers were written on this conversion problem! That is not to say, however, that (10.18) is the most efficient way to solve the problem. But it does produce very readable code, which is equally important.

We will now use the blossoming principle to discuss *degree elevation*. Formally, we can write any n^{th}-degree piecewise polynomial curve over a given knot sequence as one of degree $n+1$. It will not be over the same knot sequence: instead, we will have to increase the multiplicity of each knot by one. We denote these new knots by $\hat{u}_j$. The task is then to find the B-spline control points of the degree elevated curve, similar to the process of degree elevation for Bézier curves as described in Section 5.1.

The degree elevated curve $\hat{\mathbf{d}}$ has $\hat{\mathbf{d}}_K[v_1, \ldots, v_{n+1}]$ as the blossom of the interval $[\hat{u}_K, \hat{u}_{K+1}] = [u_I, u_{I+1}]$. How can we write it as a combination of blossoms of n variables? We can try the following:

$$\hat{\mathbf{d}}_K[v_1, \ldots, v_{n+1}] = \frac{1}{n+1} \sum_{j=1}^{n+1} \mathbf{d}_I[v_1, \ldots, v_{n+1}|v_j], \tag{10.19}$$

where $[v_1, \ldots, v_{n+1}|v_j]$ is the argument sequence $[v_1, \ldots, v_{n+1}]$ with v_j removed from it. This simple attempt already yields the solution: $\hat{\mathbf{d}}_K$ is a blossom, being a barycentric combination of blossoms. Also, it is symmetric and multiaffine, and it yields a point on the given curve for the case of all v_j being equal.

To be more specific: we know that the control points $\hat{\mathbf{d}}_j$ are the blossom values

$$\hat{\mathbf{d}}_{K-n+r} = \hat{\mathbf{d}}_K[\hat{u}_{K-n+r}, \ldots, \hat{u}_{K+r}]; \quad r = 0, \ldots, n+1$$

by application of (10.17). Using (10.19), we now have the desired result:

$$\hat{\mathbf{d}}_{K-n+r} = \frac{1}{n+1} \sum_{j=1}^{n+1} \mathbf{d}_I[\hat{u}_{K-n+r}, \ldots, \hat{u}_{K+r}|\hat{u}_{K-n+r+j-1}]; \quad r = 0, \ldots, n+1. \tag{10.20}$$

Thus the new B-spline vertices can be found by evaluating blossoms at n arguments of the new knot sequence and then taking their average. The corresponding formula for the basis functions is given in Section 10.11. A specific case is discussed in Example 10.1.

Literature on B-spline blossoms: [469], [465], [464], [467], [415].

10.10 Approximation

The full generality of the theory of B-splines allows a broader class of curve construction algorithms. Curves are not always required to pass *through* a set of points; sometimes it may suffice to be *close* to the given points. In this case, we speak of *approximating* curves. Figure 10.15 illustrates.

Let a cubic B-spline curve be defined over $\{u_0 = u_1 = u_2, u_3, \ldots\}$. Then the interval $[u_4, u_5]$ corresponds to $[\hat{u}_7, \hat{u}_8]$. The new control point $\hat{\mathbf{d}}_4$ is computed as follows:

$$\begin{aligned}\hat{\mathbf{d}}_4 &= \hat{\mathbf{d}}_7[\hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7] \\ &= \frac{1}{4}\big(\mathbf{d}_4[\hat{u}_4, \hat{u}_5, \hat{u}_6] + \mathbf{d}_4[\hat{u}_4, \hat{u}_5, \hat{u}_7] + \mathbf{d}_4[\hat{u}_4, \hat{u}_6, \hat{u}_7] + \mathbf{d}_4[\hat{u}_5, \hat{u}_6, \hat{u}_7]\big) \\ &= \frac{1}{2}\big(\mathbf{d}_4[u_3, u_3, u_4] + \mathbf{d}_4[u_3, u_4, u_4]\big).\end{aligned}$$

For the last step, we have utilized $\hat{u}_4 = \hat{u}_5 = u_3$ and $\hat{u}_6 = \hat{u}_7 = u_4$.

Example 10.1: B-spline degree elevation and blossoms.

As an example, consider the generation of an airplane wing: its cross-sections (profiles) are defined by analytical means, optimizing some airflow characteristics, for example.[6] One can now compute many (100, say) points on the profile and then ask for a curve through them. A cubic spline interpolant would do the job, but it would have too many segments—for a typical profile, a curve with 15 segments might provide a perfect fit. One possibility is to simply discard data points until we are left with the desired number. We would then compute the interpolant to the reduced data set and check if the discarded points are within tolerance. This is expensive, and a more frequently encountered approach is one that makes sure that *all* data points are *as close as possible* to the curve, avoiding any iterations.

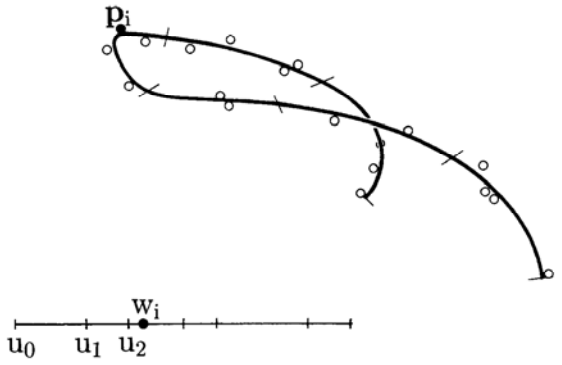

Figure 10.15: Least squares approximation: the curve should be *close* to the data points.

[6] Many explicit wing section equations are given by the so-called NACA profiles.

To make matters more precise, assume that we are given data points $\mathbf{p}_i$ with $i = 0, \ldots, P$. We wish to find an approximating B-spline curve $\mathbf{p}(u)$ of degree n with L domain knots, i.e., with a knot sequence $u_0, \ldots, u_{L+2n-2}$. We want the curve to be close to the data points in the following sense. Suppose the data point $\mathbf{p}_i$ is associated with a data parameter value w_i.[7] Then we would like the distance $||\mathbf{p}_i - \mathbf{p}(w_i)||$ to be small. Attempting to minimize all such distances then amounts to

$$\text{minimize} \sum_{i=0}^{P} ||\mathbf{p}_i - \mathbf{p}(w_i)||^2. \tag{10.21}$$

The squared distances are introduced to simplify our subsequent computations. They gave the name to this method: *least squares approximation*. We shall minimize (10.21) by finding suitable B-spline control vertices $\mathbf{d}_j$:

$$\text{minimize } f(\mathbf{d}_0, \ldots, \mathbf{d}_{L+n-1}) = \sum_{i=0}^{P} ||\mathbf{p}_i - \sum_{j=0}^{L+n-1} \mathbf{d}_j N_j^n(w_i)||^2. \tag{10.22}$$

Slightly rewritten, this becomes

$$\text{minimize } f(\mathbf{d}_0, \ldots, \mathbf{d}_{L+n-1}) = \sum_{i=0}^{P} \left[\mathbf{p}_i - \sum_{j=0}^{L+n-1} \mathbf{d}_j N_j^n(w_i)\right] \left[\mathbf{p}_i - \sum_{j=0}^{L+n-1} \mathbf{d}_j N_j^n(w_i)\right]. \tag{10.23}$$

Thus f is a quadratic form with $L + n$ independent variables $\mathbf{d}_j$. Such functions only have one minimum, and at its location, the partials with respect to the $\mathbf{d}_j$ must vanish: $\partial f / \partial \mathbf{d}_k = \mathbf{0}$.[8] Thus:

$$\mathbf{0} = \sum_{i=0}^{P} \left[\mathbf{p}_i - \sum_{j=0}^{L+n-1} \mathbf{d}_j N_j^n(w_i)\right] N_k^n(w_i); \quad k = 0, \ldots, L + n - 1 \tag{10.24}$$

or

$$\sum_{j=0}^{L+n-1} \mathbf{d}_j \sum_{i=0}^{P} N_j^n(w_i)) N_k^n(w_i) = \sum_{i=0}^{P} \mathbf{p}_i N_k^n(w_i); \quad k = 0, \ldots, L + n - 1 \tag{10.25}$$

This is a linear system of $L + n$ equations for the unknowns $\mathbf{d}_k$, with a coefficient matrix M whose elements $m_{j,k}$ are given by

$$m_{j,k} = \sum_{i=0}^{P} N_j^n(w_i) N_k^n(w_i); \quad 0 \le j, k \le L + n.$$

These equations are usually called *normal equations*. The symmetric matrix M, although containing many zero entries, is often ill-conditioned; special equation solvers, such as a Cholesky decomposition, should be employed. For more details on the numerical treatment of least squares problems, see [276] or [325].

[7] Note that w_i does not have to be one of the knots!

[8] This is shorthand for taking the partials for each of $\mathbf{d}_k$'s components.

The matrix M is nonsingular in all "standard" cases. It is obviously singular if the number of data points $P + 1$ is less than the number of domain knots $L + n + 1$. It is also singular if there is a span $[u_{j-1}, u_{j+n}]$ that contains no w_i. In that case, the basis function N_j^n would evaluate to zero for all w_i, resulting in a row of zeroes for M.

We have so far assumed much more than would be available in a practical situation. First, what should the degree n be? In most cases, $n = 3$ is a reasonable choice. The knot sequence poses a more serious problem.

Recall that the data points are typically given without assigned data parameter values w_i. The centripetal parametrization from Section 9.4 will give reasonable estimates, provided that there is not too much noise in the data. But how many knots u_j shall we use, and what values should they receive? A universal answer to this question does not exist—it will invariably depend on the application at hand. For example, if the data points come from a laser digitizer, there will be vastly more data points $\mathbf{p}_i$ than knots u_i.

After the curve $\mathbf{p}(u)$ has been computed, we will find that many distance vectors $\mathbf{p}_i - \mathbf{p}(w_i)$ are not perpendicular to $\dot{\mathbf{p}}(w_i)$. This means that the point $\mathbf{p}(w_i)$ on the curve is not the closest point to $\mathbf{p}_i$, and thus $||\mathbf{p}_i - \mathbf{p}(w_i)||$ does not measure the distance of $\mathbf{p}_i$ to the curve. This indicates that we could have chosen a better data parameter value w_i corresponding to $\mathbf{p}_i$. We may improve our estimate for w_i by finding the closest point to $\mathbf{p}_i$ on the computed curve and assigning its parameter value $\hat{w}_i$ to $\mathbf{p}_i$; see Figure 10.16. We do this for all i and then recompute the least squares curve with the new $\hat{w}_i$. This process typically converges after three or four iterations. It was named *parameter correction* by J. Hoschek [291].

The new parameter value $\hat{w}_i$ is found using a Newton iteration. We project $\mathbf{p}_i$ onto the tangent at $\mathbf{p}(w_i)$, yielding a point $\mathbf{q}_i$. Then the ratio of the lengths $||\mathbf{q}_i - \mathbf{p}_i||/||\dot{\mathbf{p}}(w_i)||$ is a measure for the adjustment of w_i. The actual Newton iteration step looks like this:

$$\hat{w}_i = w_i + [\mathbf{p}_i - \mathbf{p}(w_i)] \frac{\dot{\mathbf{p}}(w_i)}{||\dot{\mathbf{p}}(w_i)||} \frac{\Delta u_k}{s_k}. \tag{10.26}$$

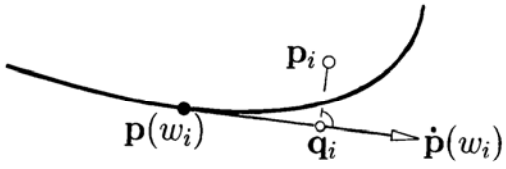

Figure 10.16: Parameter correction: the connection of $\mathbf{p}_i$ and $\mathbf{p}(w_i)$ is typically not perpendicular to the tangent at $\mathbf{p}(w_i)$. A better value for w_i is found by projecting $\mathbf{p}_i$ onto the tangent.

In this equation, s_k denotes the arc length of the segment that w_i is in, i.e., $u_k < w_i \leq u_{k+1}$. This length may safely (and cheaply) be overestimated by the length of the Bézier polygon of the k^{th} segment.[9]

We finally note that (10.26) should not be used to compute the point on a curve closest to an arbitrary point $\mathbf{p}_i$. It only works if $\mathbf{p}_i$ is close to the curve, and if a good estimate w_i is known for the closest point on the curve.

10.11 B-spline Basics

Here, we present a collection of the most important formulas and definitions of this chapter. As before, n is the (maximal) degree of each polynomial segment, L is the number of curve segments if all knots in the domain are simple, and, more generally, $L + 1$ is the sum of all domain knot multiplicities.

Knot sequence: $\{u_0, \ldots, u_{L+2n-2}\}$.

Domain: Curve is only defined over $[u_{n-1}, \ldots, u_{L+n-1}]$.

Greville abscissas: $\xi_i = \frac{1}{n}(u_i + \cdots + u_{i+n-1})$.

Support: N_i^n is nonnegative over $[u_{i-1}, u_{i+n}]$.

Control polygon P: (ξ_i, d_i); $i = 0, \ldots, L + n - 1$.

Knot insertion: To insert $u_I \leq u < u_{I+1}$: (1) Find new Greville abscissas $\hat{\xi}_i$. (2) Set new $d_i = P(\hat{\xi}_i)$.

de Boor algorithm: Given $u_I \leq u < u_{I+1}$, set

$$d_i^k(u) = \frac{u_{i+n-k} - u}{u_{i+n-k} - u_{i-1}} d_{i-1}^{k-1}(u) + \frac{u - u_{i-1}}{u_{i+n-k} - u_{i-1}} d_i^{k-1}(u)$$

for $k = 1, \ldots, n - r$, and $i = I - n + k + 1, \ldots, I + 1$. Here, r denotes the multiplicity of u. (Normally, u is not already in the knot sequence; then, $r = 0$.)

Boehm recursion: Let $\hat{u}$ be a new knot; then,

$$N_l^n(u) = \frac{\hat{u} - u_{l-1}}{u_{l+n-1} - u_{l-1}} \hat{N}_l^n(u) + \frac{u_{l+n} - \hat{u}}{u_{l+n} - u_l} \hat{N}_{l+1}^n(u).$$

Mansfield, de Boor, Cox recursion:

$$N_l^n(u) = \frac{u - u_{l-1}}{u_{l+n-1} - u_{l-1}} N_l^{n-1}(u) + \frac{u_{l+n} - u}{u_{l+n} - u_l} N_{l+1}^{n-1}(u).$$

Derivative:

$$\frac{\mathrm{d}}{\mathrm{d}u} N_l^n(u) = \frac{n}{u_{n+l-1} - u_{l-1}} N_l^{n-1}(u) - \frac{n}{u_{l+n} - u_l} N_{l+1}^{n-1}(u).$$

[9] Hoschek's original development uses $u_{L+n-1} - u_{n-1}$ instead of Δu_k and the length of the total curve instead of s_k. Our formula is cheaper and worked well in our applications.

Derivative of B-spline curve:

$$\frac{\mathrm{d}}{\mathrm{d}u}s(u) = n \sum_{i=1}^{L+n-1} \frac{\Delta d_{i-1}}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u).$$

Degree elevation:

$$N_i^n(u) = \frac{1}{n+1} \sum_{j=i-1}^{n+i} N_i^{n+1}(u; u_j),$$

where $N_i^{n+1}(u; u_j)$ is defined over the original knot sequence except that the knot u_j has its multiplicity increased by one. This identity was discovered by H. Prautzsch in 1984 [410]. Another reference is Barry and Goldman [33].

10.12 Implementation

Here is the header for the de Boor algorithm code:

```
float deboor(degree,coeff,knot,u,i)
/*   uses de Boor algorithm to compute one
     coordinate on B-spline curve for param. value u in interval i.
Input: degree:        polynomial degree of each piece of curve
       coeff:         B-spline control points
       knot:          knot sequence
       u:             evaluation abscissa
       i:             u's interval: u[i]<= u < u[i+1]
Output:               coordinate value.
*/
```

This program does not need to know about L. The next program generates a set of points on a whole B-spline curve—for one coordinate, to be honest—so it has to be called twice for a 2D curve and three times for a 3D curve.

```
bspl_to_points(degree,l,coeff,knot,dense,points,point_num)
/*     generates points on B-spline curve. (one coordinate)
Input: degree:         polynomial degree of each piece of curve
       l:              number of active  intervals
       coeff:          B-spline control points
       knot:           knot sequence: knot[0]...knot[l+2*degree-2]
       dense:          how many points per segment
Output:points:         output array with function values.
       point_num:      how many points are generated. That number is
                       easier computed here than in the calling program:
                       no points are generated between multiple knots.
*/
```

The main program `deboormain.c` generates a postscript plot of a B-spline curve. A sample input file is in `bspl.dat`; it creates the outline of the letter **r** from Figure 8.6.

As a second example, the input data for the y-values of the curve in Figure 10.1 are:

`degree = 3; l = 4; coeff = 0.8, 2.8, 5.7, 2.6, 5.7, 4.0, 0.6; knot = 0.0, 0.0, 0.0, 2.6, 7.7, 9.9, 17.8, 17.8, 17.8; dense = 10.`

Next, we include a B-spline blossom routine:

```
deboor_blossom(control,degree,deboor,deboor_wts,
               knot,uvec,interval,point,point_wt)

/*

  FUNCTION: deBoor algorithm to evaluate a B-spline curve blossom.
            For polynomial or rational curves.

  INPUT:    control[] ........... [0]: indicates type of input curve
                                       0 = polynomial
                                       1 = rational
                                  [1]: indicates if input/output is
                                       in R3 or R4;
                                       3 = R3
                                       4 = R4
            degree .............. polynomial degree of each piece
                                  of the input curve, must be <=20
            deboor[][3] ......... deboor control points
            deboor_wts[] ........ rational weights associated with
                                  the control points if control[0]=1;
                                  otherwise weights not used
            knot[] .............. knot sequence with multiplicities
                                  entered explicitly
            uvec[]  ............. blossom (parameter) values
                                  to evaluate
            interval ............ interval within knot sequence
                                  with which to evaluate wrt u
                                  (typically: i=interval then
                                  knot[i]<= u < knot[i+1])

  OUTPUT:   point[3] ............ evaluation point;
                                  depending on control[] values,
                                  this point will be in R3 or R4
            point_wt ............ if control[0]=1 then this is the
                                  rational weight associated with
                                  the point
```

10.13 Exercises

1. For the case of a planar parametric B-spline curve, does symmetry of the polygon with respect to the y-axis imply that same symmetry for the curve?
2. Prove (10.1). Hint: use similar triangles.

*3. Find the Bézier points of the closed B-spline curves of degree four whose control polygons consist of the edges of a square and have (a) uniform knot spacing and simple knots, (b) uniform knot spacing, and knots all with multiplicity two.

*4. Work out the conditions under which the least squares approximation of Section 10.10 yields an interpolating curve.

P1. Use `de_boor_blossom` to program degree elevation for B-spline curves.

P2. Take the data from the file `outline_3D` and approximate by a least squares cubic spline with fewer segments than you did for the corresponding problem in Chapter 9. Compare.

Chapter 11

W. Boehm: Differential Geometry I

Differential geometry is based largely on the pioneering work of L. Euler (1707–1783), C. Monge (1746–1818), and C. F. Gauss (1777–1855). One of their concerns was the description of local curve and surface properties such as curvature. These concepts are also of interest in modern computer-aided geometric design. The main tool for the development of general results is the use of local coordinate systems, in terms of which geometric properties are easily described and studied. This introduction discusses local properties of curves independent of a possible imbedding into a surface.

11.1 Parametric Curves and Arc Length

A curve in $\mathbb{E}^3$ is given by the parametric representation

$$\mathbf{x} = \mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}, \quad t \in [a, b] \subset \mathbb{R}, \tag{11.1}$$

where its cartesian coordinates x, y, z are differentiable functions of t (see Figure 11.1). (We have encountered a variety of such curves already, among them Bézier and B-spline curves.) To avoid potential problems concerning the parametrization of the curve, we shall assume that

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} \neq \mathbf{0}, \quad t \in [a, b], \tag{11.2}$$

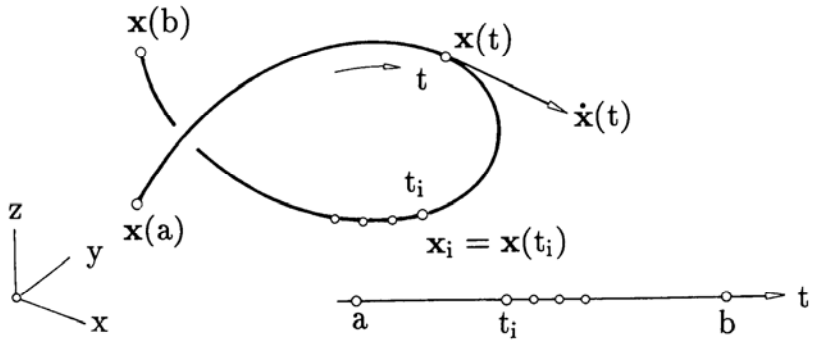

Figure 11.1: Parametric curve in space.

where dots denote derivatives with respect to t. Such a parametrization is called *regular*.

A change $\tau = \tau(t)$ of the parameter t, where τ is a differentiable function of t, will not change the shape of the curve. This *reparametrization* will be regular if $\dot{\tau} \neq 0$ for all $t \in [a, b]$, i.e., we can find the inverse $t = t(\tau)$. Let

$$s = s(t) = \int_a^t ||\dot{\mathbf{x}}|| \mathrm{d}t \tag{11.3}$$

be such a parametrization. Because

$$\dot{\mathbf{x}}\mathrm{d}t = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\tau}\frac{\mathrm{d}\tau}{\mathrm{d}t}\mathrm{d}t = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\tau}\mathrm{d}\tau,$$

s is independent of any regular reparametrization. It is an invariant parameter and is called *arc length* parametrization of the curve. One also calls $\mathrm{d}s = ||\dot{\mathbf{x}}||\mathrm{d}t$ the *arc element* of the curve.

Remark 1 Arc length may be introduced more intuitively as follows: let $t_i = a + i\Delta t$ and let $\Delta t > 0$ be an equidistant partition of the t-axis. Let $\mathbf{x}_i = \mathbf{x}(t_i)$ be the corresponding sequence of points on the curve. *Chord length* is then defined by

$$S = \sum_i ||\Delta \mathbf{x}_i|| = \sum_i \left|\left|\frac{\Delta \mathbf{x}_i}{\Delta t}\right|\right| \Delta t, \tag{11.4}$$

where $\Delta \mathbf{x}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$. It is easy to check that for $\Delta t \to 0$, chord length S converges to arc length s, while $\Delta \mathbf{x}_i / \Delta t$ converges to the tangent vector $\dot{\mathbf{x}}_i$ at $\mathbf{x}_i$.

Remark 2 Although arc length is an important concept, it is primarily used for theoretical considerations and for the development of curve algorithms. If, for some application, computation of the arc length is unavoidable, it may be approximated by the chord length (11.4).

11.2 The Frenet Frame

We will now introduce a special local coordinate system, linked to a point $\mathbf{x}(t)$ on the curve, that will significantly facilitate the description of local curve properties at that point. Let us assume that all derivatives needed later do exist. The first terms of the Taylor expansion of $\mathbf{x}(t + \Delta t)$ at t are given by

$$\mathbf{x}(t + \Delta t) = \mathbf{x} + \dot{\mathbf{x}}\Delta t + \ddot{\mathbf{x}}\frac{1}{2}\Delta t^2 + \dddot{\mathbf{x}}\frac{1}{6}\Delta t^3 + \ldots.^{1}$$

Let us assume that the first three derivatives are linearly independent. Then $\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}}$ form a local affine coordinate system with origin $\mathbf{x}$. In this system, $\mathbf{x}(t)$ is represented by its *canonical coordinates*

$$\begin{bmatrix} \Delta t + \ldots \\ \frac{1}{2}\Delta t^2 + \ldots \\ \frac{1}{6}\Delta t^3 + \ldots \end{bmatrix},$$

where "..." denotes terms of degree four and higher in Δt.

From this local affine coordinate system, one easily obtains a local cartesian (orthonormal) system with origin $\mathbf{x}$ and axes $\mathbf{t}$, $\mathbf{m}$, $\mathbf{b}$ by the Gram–Schmidt process of orthonormalization, as shown in Figure 11.2:

$$\mathbf{t} = \frac{\dot{\mathbf{x}}}{||\dot{\mathbf{x}}||}, \quad \mathbf{m} = \mathbf{b} \wedge \mathbf{t}, \quad \mathbf{b} = \frac{\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}}{||\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}||}, \tag{11.5}$$

where "$\wedge$" denotes the cross product.

The vector $\mathbf{t}$ is called *tangent vector* (see Remark 1), $\mathbf{m}$ is called *main normal vector*,[2] and $\mathbf{b}$ is called *binormal vector*. The frame (or trihedron) $\mathbf{t}$, $\mathbf{m}$, $\mathbf{b}$ is called the *Frenet frame*; it varies its orientation as t traces out the curve.

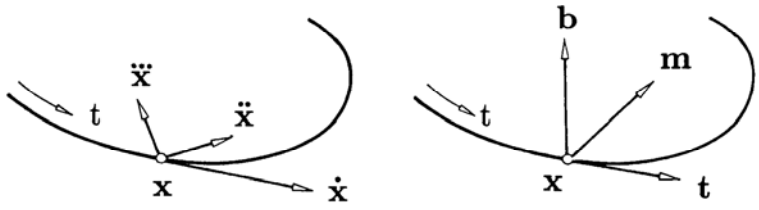

Figure 11.2: Local affine system (left) and Frenet frame (right).

[1] We use the abbreviation $\Delta t^2 = (\Delta t)^2$.

[2] Warning: one often sees the notation $\mathbf{n}$ for this vector. We use $\mathbf{m}$ to avoid confusion with surface normals, which are discussed later.

The plane spanned by the point $\mathbf{x}$ and the two vectors $\mathbf{t}, \mathbf{m}$ is called the *osculating plane* $\mathbf{O}$. Its equation is

$$\det \begin{bmatrix} \mathbf{y} & \mathbf{x} & \dot{\mathbf{x}} & \ddot{\mathbf{x}} \\ 1 & 1 & 0 & 0 \end{bmatrix} = \det[\mathbf{y} - \mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}] = 0,$$

where $\mathbf{y}$ denotes any point on $\mathbf{O}$. Its parametric form is

$$\mathbf{O}(u, v) = \mathbf{x} + u\dot{\mathbf{x}} + v\ddot{\mathbf{x}}.$$

Remark 3 The process of orthonormalization yields

$$\mathbf{m} = \frac{\dot{\mathbf{x}}\dot{\mathbf{x}} \cdot \ddot{\mathbf{x}} - \dot{\mathbf{x}}\ddot{\mathbf{x}} \cdot \dot{\mathbf{x}}}{||\dot{\mathbf{x}}\dot{\mathbf{x}} \cdot \ddot{\mathbf{x}} - \dot{\mathbf{x}}\ddot{\mathbf{x}} \cdot \dot{\mathbf{x}}||}.$$

This equation may also be used for planar curves, where the binormal vector $\mathbf{b} = \mathbf{t} \wedge \mathbf{m}$ agrees with the normal vector of the plane.

11.3 Moving the Frame

Letting the Frenet frame vary with t provides a good idea of the curve's behavior in space. It is a fundamental idea in differential geometry to express the local change of the frame in terms of the frame itself. The resulting formulas are particularly simple if one uses arc length parametrization. We denote differentiation with respect to arc length by a prime. Since $\mathbf{x}' = \mathbf{t}$ is a unit vector, one finds the following two identities:

$$\mathbf{x}' \cdot \mathbf{x}' = 1 \quad \text{and} \quad \mathbf{x}' \cdot \mathbf{x}'' = 0.$$

The first identity states that the curve is traversed with *unit speed*; the second one states that the tangent vector is perpendicular to the second derivative vector, provided the curve is parametrized with respect to arc length.

Some simple calculations yield the so-called *Frenet–Serret* formulas:

$$\begin{array}{rcrcr} \mathbf{t}' & = & & +\kappa\mathbf{m} & \\ \mathbf{m}' & = & -\kappa\mathbf{t} & & +\tau\mathbf{b}, \\ \mathbf{b}' & = & & -\tau\mathbf{m} & \end{array} \tag{11.6}$$

where the terms κ and τ, called *curvature* and *torsion*, may be defined both in terms of arc length s and in terms of the actual parameter t. We give both definitions:

$$\kappa = \kappa(s) = ||\mathbf{x}''||,$$

$$\kappa = \kappa(t) = \frac{||\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}||}{||\dot{\mathbf{x}}||^3}, \tag{11.7}$$

$$\tau = \tau(s) = \frac{1}{\kappa^2}\det[\mathbf{x}', \mathbf{x}'', \mathbf{x}'''],$$

$$\tau = \tau(t) = \frac{\det[\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}}]}{||\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}||^2}. \tag{11.8}$$

Figure 11.3 illustrates the formulas of Eqs. (11.6).

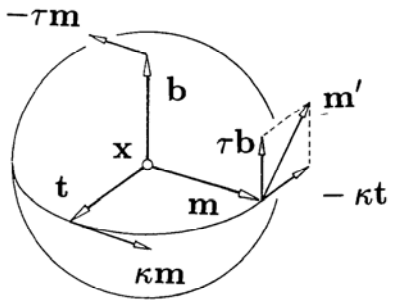

Figure 11.3: The geometric meaning of the Frenet-Serret formulas.

Curvature and torsion have an intuitive geometric meaning: consider a point $\mathbf{x}(s)$ on the curve and a "consecutive" point $\mathbf{x}(s + \Delta s)$. Let $\Delta\alpha$ denote the angle between the two tangent vectors $\mathbf{t}$ and $\mathbf{t}(s + \Delta s)$ and let $\Delta\beta$ denote the angle between the two binormal vectors $\mathbf{b}$ and $\mathbf{b}(s + \Delta s)$, both angles measured in radians. It is easy to verify that $\Delta\alpha = \kappa\Delta s + \ldots$ and $\Delta\beta = -\tau\Delta s + \ldots$, where "..." denotes terms of higher degree in Δs. Thus, when $\Delta s \longrightarrow \mathrm{d}s$, one finds that

$$\kappa = \frac{\mathrm{d}\alpha}{\mathrm{d}s}, \quad \tau = -\frac{\mathrm{d}\beta}{\mathrm{d}s}.$$

In other words, κ and $-\tau$ are the angular velocities of $\mathbf{t}$ and $\mathbf{b}$, respectively, because the frame is moved according to the parameter s.

Remark 4 Note that κ and τ are independent of the current parametrization of the curve. They are euclidean invariants of the curve, i.e., they are not changed by a rigid body motion of the curve. Moreover, any two continuous functions $\kappa = \kappa(s) > 0$ and $\tau = \tau(s)$ define uniquely (except for rigid body motions) a curve that has curvature κ and torsion τ.

Remark 5 The curve may be written in canonical form in terms of the Frenet frame. Then it has the form

$$\mathbf{x}(s + \Delta s) = \begin{bmatrix} \Delta s & & -\frac{1}{6}\kappa^2\Delta s^3 + \ldots \\ & \frac{1}{2}\kappa\Delta s^2 & +\frac{1}{6}\kappa'\Delta s^3 + \ldots \\ & & \frac{1}{6}\kappa\tau\Delta s^3 + \ldots \end{bmatrix},$$

where "..." again denotes terms of higher degree in Δs.

11.4 The Osculating Circle

The circle that has second-order contact with the curve at $\mathbf{x}$ is called the *osculating circle* (Figure 11.4). Its center is $\mathbf{c} = \mathbf{x} + \rho\mathbf{m}$, and its radius $\rho = \frac{1}{\kappa}$ is called the *radius of curvature*. We shall provide a brief development of these facts. Using the

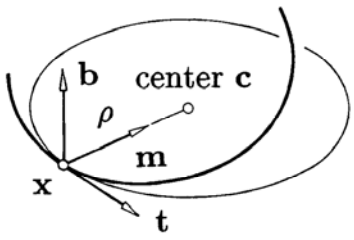

Figure 11.4: The osculating circle.

Frenet–Serret formulas of Eqs. (11.6), the Taylor expansion of $\mathbf{x}(s + \Delta s)$ can be written as

$$\mathbf{x}(s + \Delta s) = \mathbf{x}(s) + \mathbf{t}\Delta s + \frac{1}{2}\kappa\mathbf{m}\Delta s^2 + \dots.$$

Let ρ^* be the radius of the circle that is tangent to $\mathbf{t}$ at $\mathbf{x}$ and passes through the point $\mathbf{y} = \mathbf{x} + \Delta\mathbf{x}$, where $\Delta\mathbf{x} = \mathbf{t}\Delta s + \frac{1}{2}\kappa\mathbf{m}\Delta s^2$ (see Figure 11.5). Note that $\mathbf{y}$ lies in the osculating plane $\mathbf{O}$. Inspection of the figure reveals that $(\frac{1}{2}\Delta\mathbf{x} - \rho^*\mathbf{m})\Delta\mathbf{x} = 0$, i.e., one obtains

$$\rho^* = \frac{1}{2}\frac{(\Delta\mathbf{x})^2}{\mathbf{m}\Delta\mathbf{x}}.$$

From the definition of $\Delta\mathbf{x}$ one obtains $(\Delta\mathbf{x})^2 = \Delta s^2 + \dots$ and $\mathbf{m}\Delta\mathbf{x} = \frac{1}{2}\kappa(\Delta s)^2$. Thus $\rho^* = \frac{1}{\kappa} + \dots$. In particular, $\rho = \frac{1}{\kappa}$ as $\Delta s \to 0$. Obviously, this circle lies in the osculating plane.

Remark 6 Let $\mathbf{x}$ be a rational Bézier curve of degree n as defined in Chapter 14. Its curvature and torsion at $\mathbf{b}_0$ are given by

$$\kappa = \frac{n-1}{n}\frac{w_0 w_2}{w_1^2}\frac{b}{a^2}, \quad \tau = \frac{n-2}{n}\frac{w_0 w_3}{w_1 w_2}\frac{c}{ab}, \tag{11.9}$$

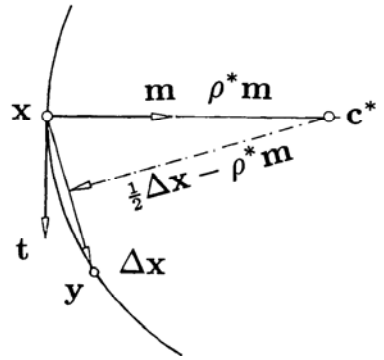

Figure 11.5: Construction of the osculating circle.

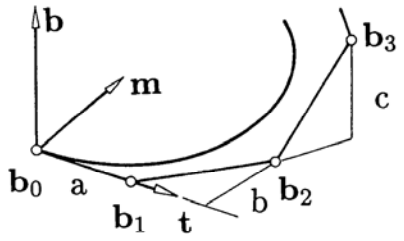

Figure 11.6: Frenet frame and geometric meaning of a, b, c.

where a is the distance between $\mathbf{b}_0$ and $\mathbf{b}_1$, b is the distance of $\mathbf{b}_2$ to the tangent spanned by $\mathbf{b}_0$ and $\mathbf{b}_1$, and c is the distance of $\mathbf{b}_3$ from the osculating plane spanned by $\mathbf{b}_0, \mathbf{b}_1$, and $\mathbf{b}_2$ (Figure 11.6). Note that these formulas can be used to calculate curvature and torsion at arbitrary points $\mathbf{x}(t)$ of a Bézier curve after subdividing it there (see Section 14.2).

Remark 7 An immediate application of (11.9) is the following: Let $\mathbf{x}$ be a point on an integral quadratic Bézier curve, i.e., a parabola. Let 2δ denote the length of a chord parallel to the tangent at $\mathbf{x}$, and let ϵ be the distance between the chord and the tangent. The radius of curvature at $\mathbf{x}$ is then $\rho = \frac{\delta^2}{2\epsilon}$ (see Figure 11.7).

Remark 8 An equivalent way to formulate (11.9) is given by

$$\kappa = 2\frac{n-1}{n}\frac{w_0 w_2}{w_1^2}\frac{\text{area}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}{\text{dist}^3[\mathbf{b}_0, \mathbf{b}_1]} \tag{11.10}$$

and

$$\tau = \frac{3}{2}\frac{n-2}{n}\frac{w_0 w_3}{w_1 w_2}\frac{\text{volume}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]}{\text{area}^2[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}. \tag{11.11}$$

The advantage of this formulation is that it can be generalized to "higher order curvatures" of curves that span $\mathbb{R}^d$, $3 < d \leq n$ (see Remark 12). An application of this possible generalization is addressed in Remark 13.

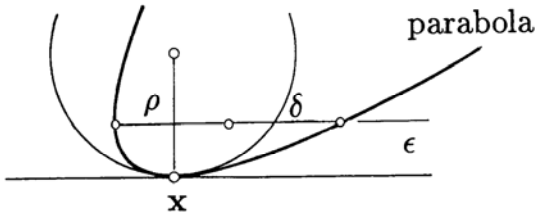

Figure 11.7: Curvature of a parabola.

11.5 Nonparametric Curves

Let $y = y(t);\ t \in [a, b]$ be a function. The planar curve $\begin{bmatrix} t \\ y(t) \end{bmatrix}$ is called the graph of $y(t)$ or a *nonparametric curve*. From the preceding, one derives the following:
The arc element:

$$\mathrm{d}s = \sqrt{1 + \dot{y}^2}\mathrm{d}t.$$

The tangent vector:

$$\mathbf{t} = \frac{1}{\sqrt{1 + \dot{y}^2}} \begin{bmatrix} 1 \\ \dot{y} \end{bmatrix}.$$

The curvature:

$$\kappa = \frac{\ddot{y}}{[1 + \dot{y}^2]^{\frac{3}{2}}}.$$

The center of curvature:

$$\mathbf{c} = \mathbf{x} + \frac{1 + \dot{y}^2}{\ddot{y}} \begin{bmatrix} -\dot{y} \\ 1 \end{bmatrix}.$$

Remark 9 Note that κ has a sign here. Any planar parametric curve can be given a *signed curvature*, for instance, by using the sign of $\det(\dot{\mathbf{x}}, \ddot{\mathbf{x}})$ [see also Eq. (23.1)].

Remark 10 For a nonparametric Bézier curve (see Section 5.5),

$$y(u) = b_0 B_0^n(t) + \cdots + b_n B_n^n(t).$$

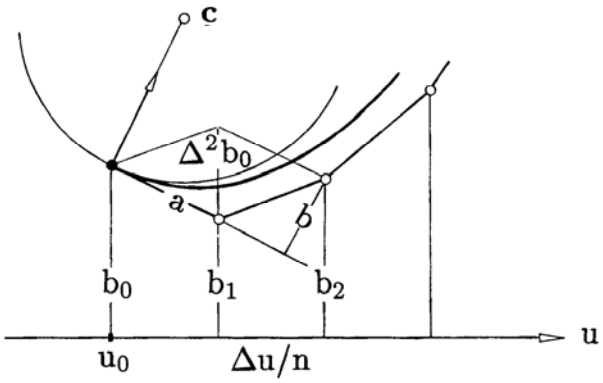

Figure 11.8: Curvature of nonparametric Bézier curve.

Where $u = u_0 + t\Delta u$ is a global parameter, we obtain

$$a = \frac{1}{n}\sqrt{\Delta u^2 + n^2(\Delta b_0)^2}, \quad b = -\frac{\Delta u}{n}\frac{\Delta^2 b_0}{a},$$

as illustrated in Figure 11.8.

11.6 Composite Curves

A curve can be composed of several segments; we have seen spline curves as an example. Let $\mathbf{x}_-$ denote the right endpoint of a segment and $\mathbf{x}_+$ the left endpoint of the adjacent segment. (We will consider only continuous curves, so that $\mathbf{x}_- = \mathbf{x}_+$ always.) Let t be a global parameter of the composite curve and let dots denote derivatives with respect to t. Obviously, the curve is tangent continuous if

$$\dot{\mathbf{x}}_+ = \alpha\dot{\mathbf{x}}_-. \tag{11.12}$$

Moreover, it is curvature and osculating plane continuous if in addition

$$\ddot{\mathbf{x}}_+ = \alpha^2\ddot{\mathbf{x}}_- + \alpha_{21}\dot{\mathbf{x}}_-, \tag{11.13}$$

and it is torsion continuous if in addition

$$\dddot{\mathbf{x}}_+ = \alpha^3\dddot{\mathbf{x}}_- + \alpha_{32}\ddot{\mathbf{x}}_- + \alpha_{31}\dot{\mathbf{x}}_- \tag{11.14}$$

and vice versa. Since we require the parametrization to be regular, it follows that $\alpha > 0$, while the α_{ij} are arbitrary parameters.

It is interesting to note that curvature and torsion continuous curves exist that are not κ' continuous[3] (see Remark 4). Conversely,

$$\mathbf{x}''' = \mathbf{t}'' = \kappa'\mathbf{m} + \kappa(-\kappa\mathbf{t} + \tau\mathbf{b})$$

implies that $\mathbf{x}'''$ is continuous if κ' is and vice versa. To ensure $\mathbf{x}'''_- = \mathbf{x}'''_+$, the coefficients α and α_{ij} must be the result of the application of the chain rule; i.e., with $\alpha_{21} = \beta$ and $\alpha_{31} = \gamma$, one finds that $\alpha_{32} = 3\alpha\beta$. Now, as before, the curve is tangent continuous if

$$\dot{\mathbf{x}}_+ = \alpha\dot{\mathbf{x}}_-, \quad \alpha > 0,$$

it is curvature and osculating plane continuous if in addition

$$\ddot{\mathbf{x}}_+ = \alpha^2\ddot{\mathbf{x}}_- + \beta\dot{\mathbf{x}}_-,$$

[3]Recall that $\kappa' = \mathrm{d}\kappa(s)/\mathrm{d}s$, where the prime denotes differentiation with respect to arc length s of the (composite) curve. A formula for κ' is provided by Eq. (23.2).

but it is κ' continuous if in addition

$$\dddot{\mathbf{x}}_+ = \alpha^3 \dddot{\mathbf{x}}_- + 3\alpha\beta\ddot{\mathbf{x}}_- + \gamma\dot{\mathbf{x}}_-$$

and vice versa.

Remark 11 For planar curves, torsion continuity is a vacuous condition, but κ' continuity is meaningful.

Remark 12 The preceding results may be used for the definition of higher order *geometric continuity*. A curve is said to be G^r, or r^{th}-order geometrically continuous, if a regular reparametrization exists after which it is C^r. This definition is obviously equivalent to the requirement of C^{r-2} continuity of κ and C^{r-3} continuity of τ. As a consequence, geometric continuity may be defined by using the chain rule, as in the earlier example for $r = 3$.

Remark 13 The geometric invariants curvature and torsion may be generalized for higher dimensional curves. Continuing the process mentioned in Remark 8, one finds that a d-dimensional curve has $d - 1$ geometric invariants. Continuity of these invariants only makes sense in $\mathbb{E}^d$, as was demonstrated for $d = 2$ in Remark 11.

Remark 14 Note that although curvature and torsion are euclidean invariants, curvature and torsion continuity (as well as the generalizations discussed in Remarks 12 and 13) are affinely invariant properties of a curve. Both are also projectively invariant properties; see Boehm [69] and Goldman and Micchelli [233].

Remark 15 If two curve segments meet with a continuous tangent and have (possibly different) curvatures κ_- and κ_+ at the common point, then the ratio κ_- / κ_+ is also a projectively invariant quantity. This is known as Memke's theorem; see Bol [78].

Chapter 12

Geometric Continuity

12.1 Motivation

Before we explain in detail the concept of geometric continuity, we will give an example of a curve that is *curvature continuous* yet *not twice differentiable*. Such curves (and, later, surfaces) are the objects that we will label *geometrically continuous*.

Figure 12.1 shows three parabolas with junction points at the midpoints of an equilateral triangle. According to (11.10), where we have to set all w_i equal to 1, all three parabolas have the same curvature at the junction points. We thus have a closed, curvature continuous curve. It is C^1 over a uniform knot sequence. But it is not C^2 according to the C^2 test of Figure 7.4.

Differential geometry teaches us that our closed curve can be *reparametrized* such that the new parameter is arc length. With that new parametrization, the curve will actually be C^2. Details are explained in Chapter 11. We shall adopt the term G^2 curves (second-order geometrically continuous) for curves that are *twice differentiable with respect to arc length but not necessarily twice differentiable with respect to their current parametrization.* Note that curves with a zero tangent vector cannot be G^2 under this definition. Planar G^2 curves have continuously varying signed curvature; G^2 space curves have continously varying binormal vectors and continuously varying curvature.

The concept of geometric continuity is more appropriate when dealing with shape; parametric continuity is appropriate when speed of traversal is an issue.[1]

Historically, several methods have been developed to deal with G^2 continuity. In the following, we present a unified treatment for most of these.

[1] Speed of traversal is important, for example, when the given curve is a vertical straight line and we consider the motion of an elevator: higher orders of continuity of its path ensure smoother rides.

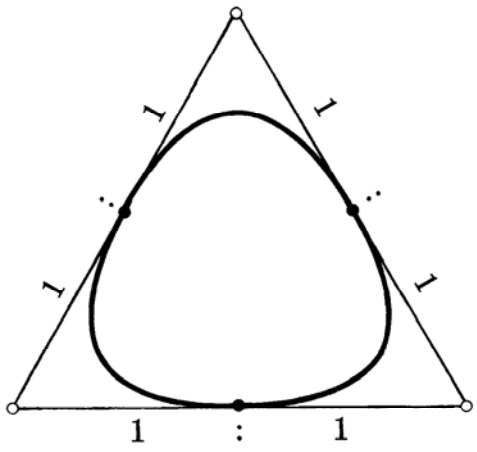

Figure 12.1: G^2 continuity: a closed quadratic G^2 spline curve.

12.2 The Direct Formulation

Let $\mathbf{b}_0, \ldots, \mathbf{b}_3$ and $\mathbf{c}_0, \ldots, \mathbf{c}_3$ be the control polygons of two cubic Bézier curves.[2] Since we are interested in G^2 continuity here, we need only consider the control points $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 = \mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$, all of which we assume to be coplanar. Referring to Figure 12.2, let $\mathbf{d}$ be the intersection of the lines $\overline{\mathbf{b}_1\mathbf{b}_2}$ and $\overline{\mathbf{c}_1\mathbf{c}_2}$.

We set

$$r_- = \text{ratio}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{d}), \tag{12.1}$$

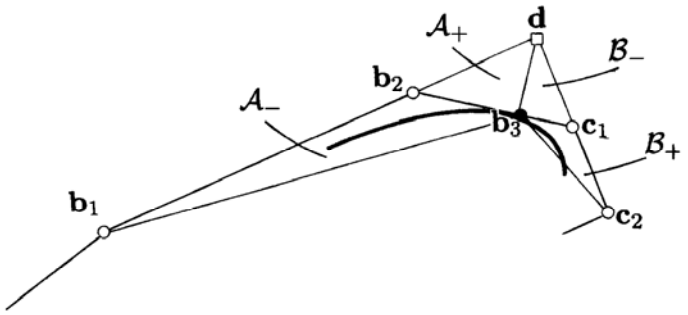

Figure 12.2: G^2 continuity: using the direct formulation.

[2]The G^2 conditions for general degrees will be identical, and so nothing is lost by concentrating on the cubic case.

$$r_+ = \text{ratio}(\mathbf{d}, \mathbf{c}_1, \mathbf{c}_2), \tag{12.2}$$

$$r = \text{ratio}(\mathbf{b}_2, \mathbf{b}_3, \mathbf{c}_1). \tag{12.3}$$

Letting $\mathcal{A}_-$, $\mathcal{A}_+$, $\mathcal{B}_-$, $\mathcal{B}_+$ denote the triangle areas in Figure 12.2, we can invoke (11.10) in order to to express the curvatures κ_- and κ_+ of the left and right segments at $\mathbf{b}_3$:

$$\kappa_- = \frac{4}{3}\frac{\mathcal{A}_-}{\|\mathbf{b}_3 - \mathbf{b}_2\|^3}, \quad \kappa_+ = \frac{4}{3}\frac{\mathcal{A}_+}{\|\mathbf{c}_1 - \mathbf{c}_0\|^3}.$$

If these two curvatures agree, we have that

$$\frac{\mathcal{A}_-}{\mathcal{A}_+} = r^3. \tag{12.4}$$

Referring to the figure again, we see that

$$\frac{\mathcal{A}_-}{\mathcal{B}_-} = r_-, \quad \frac{\mathcal{B}_+}{\mathcal{A}_+} = r_+, \quad \frac{\mathcal{B}_-}{\mathcal{B}_+} = r.$$

Inserting this into (12.4) yields our desired G^2 condition:

$$r^2 = r_- r_+. \tag{12.5}$$

12.3 The γ Formulation

Using the setting of the previous section, we observe that our composite curve could be made C^1 if we introduced a knot sequence with interval lengths Δ_-, Δ_+ satisfying $\Delta_-/\Delta_+ = r$. See Section 7.4 for a justification. Using (12.5), we define

$$\gamma := \frac{r}{r_-} = \frac{r_+}{r}.$$

We then have

$$\text{ratio}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{d}) = \frac{\Delta_-}{\gamma\Delta_+} \quad \text{and} \quad \text{ratio}(\mathbf{d}, \mathbf{c}_1, \mathbf{c}_2) = \frac{\gamma\Delta_-}{\Delta_+}.$$

In the case that $\gamma = 1$, we have the special case of a C^2 piecewise cubic curve. See also Figure 12.3.

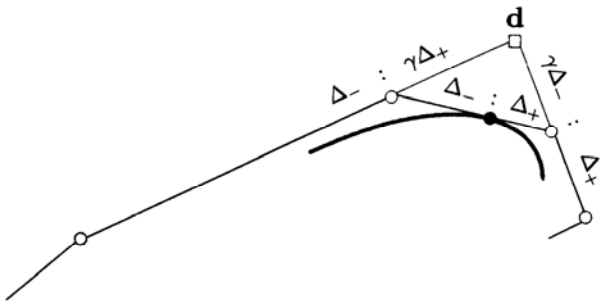

Figure 12.3: G^2 continuity: using the γ formulation.

W. Boehm used this framework for his development of G^2 cubic splines; see [65].

12.4 The ν and β Formulation

Using the knot sequence from the γ formulation, let us introduce two points $\mathbf{d}_-$ and $\mathbf{d}_+$ such that

$$\text{ratio}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{d}_-) = \text{ratio}(\mathbf{d}_+, \mathbf{c}_1, \mathbf{c}_2) = \frac{\Delta_-}{\Delta_+}.$$

We note that $\mathbf{d}_+ - \mathbf{d}_-$ is parallel to the tangent at $\mathbf{b}_3$. We start with two fairly trivial identities:

$$\frac{\Delta_+}{\Delta_-}\Delta\mathbf{b}_2 - \frac{\Delta_+}{\Delta_-}\Delta\mathbf{b}_1 = \frac{\Delta_+}{\Delta_-}\Delta^2\mathbf{b}_1,$$

$$\frac{\Delta_-}{\Delta_+}\Delta\mathbf{c}_1 - \frac{\Delta_-}{\Delta_+}\Delta\mathbf{c}_0 = \frac{\Delta_-}{\Delta_+}\Delta^2\mathbf{c}_0.$$

We may rewrite these as

$$\frac{1}{3}\Delta_+\dot{\mathbf{x}}_- - [\mathbf{d}_- - \mathbf{b}_2] = \frac{1}{6}\Delta_-\Delta_+\ddot{\mathbf{x}}_-,$$

$$[\mathbf{c}_1 - \mathbf{d}_+] - \frac{1}{3}\Delta_-\dot{\mathbf{x}}_+ = \frac{1}{6}\Delta_-\Delta_+\ddot{\mathbf{x}}_+.$$

Since our curve is C^1, we have that $\dot{\mathbf{x}}_- = \dot{\mathbf{x}}_+ = \dot{\mathbf{x}}$ and $\mathbf{c}_1 - \mathbf{b}_2 = [\Delta_- + \Delta_+]\dot{\mathbf{x}}/3$. If we now subtract the last two equations, we have

$$\mathbf{d}_- - \mathbf{d}_+ = \frac{1}{6}\Delta_-\Delta_+[\ddot{\mathbf{x}}_+ - \ddot{\mathbf{x}}_-]. \tag{12.6}$$

Since $\mathbf{d}_- - \mathbf{d}_+$ is parallel to the tangent at $\mathbf{b}_3 = \mathbf{c}_0$, so is $\ddot{\mathbf{x}}_+ - \ddot{\mathbf{x}}_-$. Hence a number ν exists such that

$$\ddot{\mathbf{x}}_+ - \ddot{\mathbf{x}}_- = \nu\dot{\mathbf{x}}. \tag{12.7}$$

We gave a geometric derivation of (12.7), but it also follows from (11.13) by setting $\alpha = 1, \alpha_{21} = \nu$.

The ν−formulation of G^2 continuity is due to G. Nielson; it was originally developed in the context of interpolatory ν-splines (see Section 12.7). A similar approach was taken by B. Barsky [34]; he uses $\beta_1 = \Delta_-/\Delta_+$ and $\beta_2 = \nu$ as the descriptors of G^2 continuity and calls them "bias" and "tension," respectively.

While ν depends on the parametrization and thus is not entirely geometric, we could use

$$\mathbf{d}_- - \mathbf{d}_+ = 3\nu\frac{\mathbf{c}_1 - \mathbf{b}_2}{[\Delta_- + \Delta_+]}$$

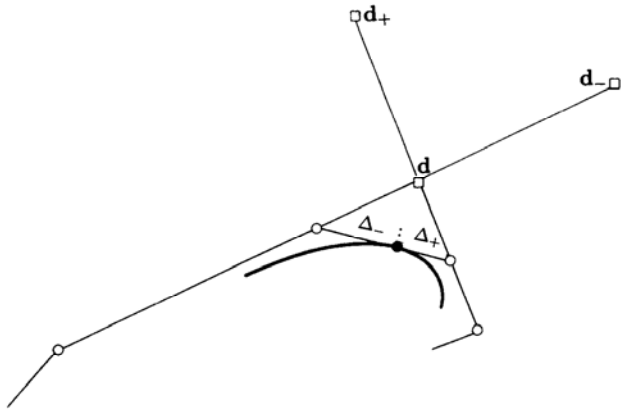

Figure 12.4: G^2 continuity: using the ν formulation.

to define a shape measure $N = 3\nu/(\Delta_- + \Delta_+)$ as the (signed) ratio between $\mathbf{d}_-$, $\mathbf{d}_+$ and $\mathbf{c}_1 - \mathbf{b}_2$. The more negative N becomes, the "rounder" the curve is at $\mathbf{b}_3$, and the more positive it is, the more "pointed" the curve is. As an example, in Figure 12.4 we have a negative value for ν.

12.5 Comparison

Why three or four different formulations for G^2 continuity of piecewise cubic curves? The reason is partly historical, and partly depends on applications. In fact, the preceding formulations are by no means the only ones—the discussion of G^2 continuity goes back as far as Bär [10], Bézier [53], Geise [226], and Manning [348].

Applications that aim at constructing surfaces will be better served by β, γ, or ν splines. These involve a knot sequence and thus lend themselves to the framework of tensor product surfaces; see Chapter 16.

Free-form curve design, on the other hand, will benefit more from the direct formulation because it is linked the most closely to the curve geometry. The direct approach is the most geometric, followed by the γ formulation, which needs a knot sequence. The least geometric are the ν and β formulations; their defining quantities are not invariant under scaling of the knot sequence.

Using the fact that the triangles $\mathbf{d}$, $\mathbf{b}_2$, $\mathbf{c}_1$ and $\mathbf{d}$, $\mathbf{d}_-$, $\mathbf{d}_+$ in Figure 12.4 are similar, we may derive the relationship

$$\nu = 2\frac{\Delta_- + \Delta_+}{\Delta_-\Delta_+}\ \frac{1-\gamma}{\gamma}, \tag{12.8}$$

first found by Boehm [65].

12.6 G^2 Cubic Splines

The spline curves in this section will be a generalization of the C^2 cubic B-splines from Chapter 7. We will follow the notation of that chapter.

We start with a control polygon $\mathbf{d}_{-1}, \ldots, \mathbf{d}_{L+1}$. In the context of C^2 cubic B-splines, we now needed a knot sequence in order to place the inner Bézier points on the control polygon legs; the junction points then were fixed by the C^2 conditions. In our case, we have more freedom: we may place the inner Bézier points *anywhere* on the control polygon legs; the junction points are then fixed by the G^2 conditions.

To be more precise, consider Figure 12.5. Placing $\mathbf{b}_{3i-2}$ on the polygon leg $\overline{\mathbf{d}_{i-1}, \mathbf{d}_i}$ amounts to picking a number α_{i-1} (between 0 and 1) and then setting

$$\mathbf{b}_{3i-2} = (1 - \alpha_{i-1})\mathbf{d}_{i-1} + \alpha_{i-1}\mathbf{d}_i. \tag{12.9}$$

Similarly, we place $\mathbf{b}_{3i-1}$ by picking a number ω_{i-1} and setting

$$\mathbf{b}_{3i-1} = (1 - \omega_{i-1})\mathbf{d}_{i-1} + \omega_{i-1}\mathbf{d}_i. \tag{12.10}$$

In the same manner, by choosing numbers α_i and ω_i, we determine $\mathbf{b}_{3i+1}$ and $\mathbf{b}_{3i+2}$.

We still have to determine the junction point $\mathbf{b}_{3i}$. Upon comparing Figures 12.5 and 12.2, we see that we need the quantities $\lambda_i = \text{ratio}(\mathbf{b}_{3i-2}, \mathbf{b}_{3i-1}, \mathbf{d}_i)$ and $\rho_i = \text{ratio}(\mathbf{d}_i, \mathbf{b}_{3i+1}, \mathbf{b}_{3i+2})$. Since

$$\mathbf{b}_{3i-1} = \frac{1 - \omega_{i-1}}{1 - \alpha_{i-1}}\mathbf{b}_{3i-2} + \frac{\omega_{i-1} - \alpha_{i-1}}{1 - \alpha_{i-1}}\mathbf{d}_i \tag{12.11}$$

and

$$\mathbf{b}_{3i+1} = \frac{\omega_i - \alpha_i}{\omega_i}\mathbf{d}_i + \frac{\alpha_i}{\omega_i}\mathbf{b}_{3i+2}, \tag{12.12}$$

we have

$$\lambda_i = \frac{\omega_{i-1} - \alpha_{i-1}}{1 - \omega_{i-1}}, \quad \rho_i = \frac{\alpha_i}{\omega_i - \alpha_i}. \tag{12.13}$$

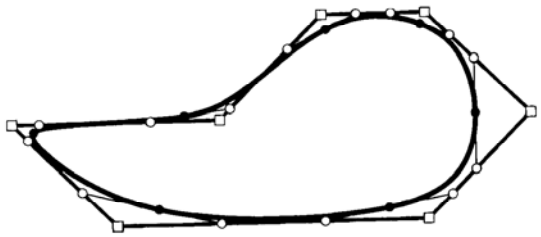

Figure 12.5: G^2 conditions: inner Bézier points may be placed on the control polygon legs. The junction points then may be found using the G^2 condition.

Setting $r_i = \sqrt{\lambda_i}/(\sqrt{\lambda_i} + \sqrt{\rho_i})$, we find the desired junction point to be

$$\mathbf{b}_{3i} = (1 - r_i)\mathbf{b}_{3i-1} + r_i\mathbf{b}_{3i+1}. \tag{12.14}$$

Continuing in this manner for all i, we have completed the definition of a G^2 spline curve. We note that it is advisable to restrict all α_i and ω_i to be between 0 and 1. It is possible, however, to violate that condition: we only have to ensure that λ_i and ρ_i have the same sign. As long as they do, $\mathbf{b}_{3i}$ is computable from (12.14).

For an open polygon, we set $\alpha_1 = 0$ and $\omega_{L-1} = 1$. This ensures the usual $\mathbf{b}_1 = \mathbf{d}_0$ and $\mathbf{b}_{3L-1} = \mathbf{d}_L$.

Our development of G^2 splines is solely based upon ratios; hence G^2 spline curves will be mapped to G^2 spline curves by affine maps. We may also say that G^2 continuity is affinely invariant.

There is one interesting difference between the preceding construction for a G^2 spline and the corresponding construction for a C^2 spline: every C^2 piecewise cubic possesses a B-spline control polygon—but not every G^2 piecewise cubic curve possesses a G^2 control polygon. The two cubics in Figure 12.6 are curvature continuous, yet they cannot be obtained with the foregoing construction: the control point $\mathbf{d}_1$ would have to be at infinity.

In interactive design, one would utilize G^2 cubic splines in a two-step procedure. The design of the G^2 control polygon may be viewed as a rough sketch. The program would estimate the inner Bézier points automatically, and the designer could fine-tune the curve shape by readjusting them where necessary. For this fine-tuning, it is

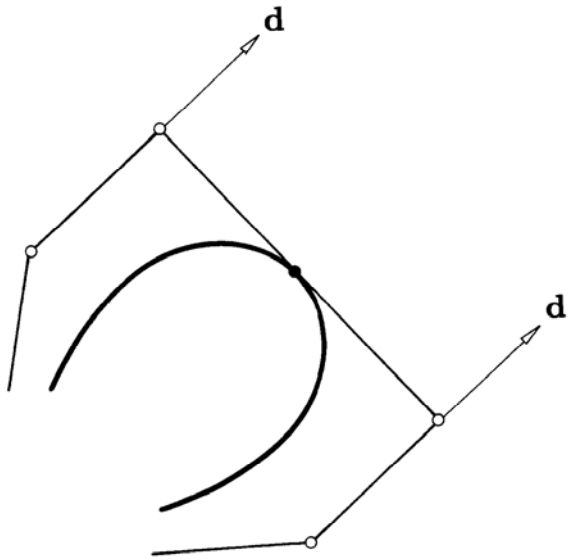

Figure 12.6: G^2 splines: these two cubics are a G^2 spline but do not possess a G^2 control polygon.

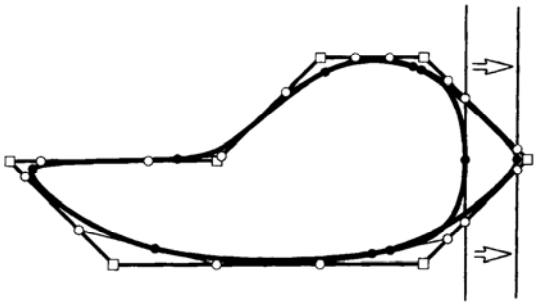

Figure 12.7: G^2 splines: a shape may be varied by prescribing tangents in addition to the control polygon.

important to observe that $\overline{\mathbf{b}_{3i-1}, \mathbf{b}_{3i+1}}$ is tangent to the curve. Instead of prescribing numbers α_i and ω_i—not very intuitive!—a designer may thus specify tangents to the curve, and the α_i, ω_i can be computed. Figure 12.7 gives examples.

We have just described G^2 splines using the direct G^2 formulation. Using the γ formulation, we arrive at γ-splines, which use a set of γ_i and a knot sequence, employing the principles of Section 12.3. We then have

$$\alpha_i = \frac{\Delta_{i-1} + \gamma_i \Delta_i}{\gamma_{i-1}\Delta_{i-2} + \Delta_{i-1} + \gamma_i \Delta_i} \tag{12.15}$$

and

$$\omega_i = \frac{\gamma_{i-1}\Delta_{i-2}}{\gamma_{i-1}\Delta_{i-2} + \Delta_{i-1} + \gamma_i \Delta_i}. \tag{12.16}$$

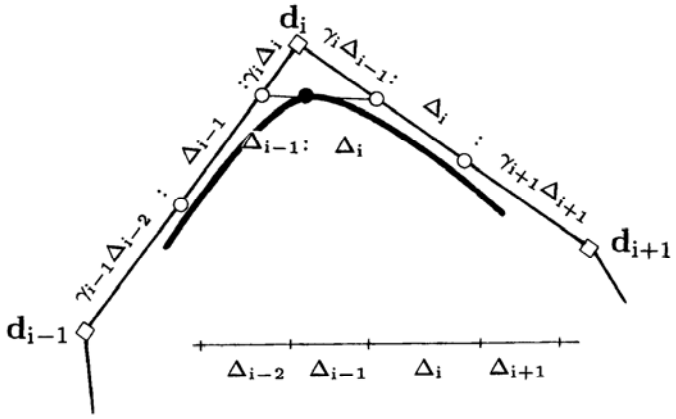

Figure 12.8: γ-splines: the Bézier points are connected to the G^2 control polygon by the ratios shown.

The geometry of a $\gamma-$spline curve is shown in Figure 12.8. Note that for all $\gamma_i \rightarrow 0$, the curve will tend toward its control polygon.

12.7 Interpolating G^2 Cubic Splines

We may also use G^2 cubics to *interpolate* to given data points $\mathbf{x}_i; i = 0, \ldots, L$. In the C^2 case, we had to supply a knot sequence in addition to the data points. Now, we have to specify a sequence of pairs α_i, ω_i. How to do this effectively is still an unsolved problem, so let us assume for now that a reasonable sequence of α_i, ω_i is given. Setting $\mathbf{b}_{3i} = \mathbf{x}_i$, we insert (12.10) and (8.2) into (12.14) and obtain:

$$(\sqrt{\lambda_i} + \sqrt{\rho_i})\mathbf{x}_i = \sqrt{\rho_i}(1 - \omega_{i-1})\mathbf{d}_{i-1} + [\sqrt{\rho_i}\omega_{i-1} + \sqrt{\lambda_i}(1 - \alpha_i)]\mathbf{d}_i + \sqrt{\lambda_i}\alpha_i\mathbf{d}_{i+1}, i = 1, \ldots, L-1. \tag{12.17}$$

Together with two end conditions, we then have $L + 1$ equations for the $L + 1$ unknowns $\mathbf{d}_i$. A suitable end condition is to make $\mathbf{d}_0$ a linear combination of the first three data points: $\mathbf{d}_0 = u\mathbf{x}_0 + v\mathbf{x}_1 + w\mathbf{x}_2$. In our experience, $(u, v, w) = (\frac{5}{6}, \frac{1}{2}, -\frac{1}{3})$ has worked well. A similar equation then holds for $\mathbf{d}_L$. For the limiting case of $\alpha_i \rightarrow 0$ and $\omega_i \rightarrow 1$, the interpolating curve will approach the polygon formed by the data points. In terms of the γ formulation, this spline type was investigated in [186].

Nielson [371] derived the G^2 interpolating spline from the ν formulation. We now assume that the data points $\mathbf{x}_i$ have parameter values u_i assigned to them. Using the piecewise cubic Hermite form, the interpolant becomes

$$\mathbf{x}(u) = \mathbf{x}_i H_0^3(r) + \mathbf{m}_i \Delta_i H_1^3(r) + \Delta_i \mathbf{m}_{i+1} H_2^3(r) + \mathbf{x}_{i+1} H_3^3(r), \tag{12.18}$$

where the H_j^3 are cubic Hermite polynomials from (6.14) and $r = (u - u_i)/\Delta_i$ is the local parameter of the interval (u_i, u_{i+1}). In (12.18), the $\mathbf{x}_i$ are the known data points, while the $\mathbf{m}_i$ are as yet unknown tangent vectors. The interpolant is supposed to be G^2; it is therefore characterized by (12.7), more specifically,

$$\ddot{\mathbf{x}}_+(u_i) - \ddot{\mathbf{x}}_-(u_i) = \nu_i \mathbf{m}_i \tag{12.19}$$

for some constants ν_i, where $\mathbf{m}_i = \dot{\mathbf{x}}(u_i)$. The ν_i are constants that can be used to manipulate the shape of the interpolant; they will be discussed soon. We insert (12.18) into (12.19) and obtain the linear system

$$3\left(\frac{\Delta_i \Delta \mathbf{x}_{i-1}}{\Delta_{i-1}} + \frac{\Delta_{i-1} \Delta \mathbf{x}_i}{\Delta_i}\right) = \Delta_i \mathbf{m}_{i-1} + (2\Delta_{i-1} + 2\Delta_i + \tfrac{1}{2}\Delta_{i-1}\Delta_i \nu_i)\mathbf{m}_i + \Delta_{i-1}\mathbf{m}_{i+1}; i = 1, \ldots, L-1. \tag{12.20}$$

Together with two end conditions, (12.20) can be used to compute the unknown tangent vectors $\mathbf{m}_i$. The simplest end condition is prescribing $\mathbf{m}_0$ and $\mathbf{m}_L$, but any other end condition from Chapter 9 may be used as well. Note that this formulation of the ν-spline interpolation problem depends on the scale of the u_i; it is not invariant under affine parameter transformations as pointed out in Section 12.5.

If the ν_i are chosen to be nonnegative, the linear system (12.20) is solvable; in the special case of all $\nu_i = 0$, it results in the standard C^2 cubic spline. For the case of all $\nu_i \to \infty$, the interpolant approaches the polygon formed by the data points.

12.8 Local Basis Functions for G^2 Splines

C^2 cubic splines form a linear space over a fixed knot sequence. G^2 have the same property, best illustrated in terms of the γ formulation. Consider two γ-spline curves $\mathbf{g}$ and $\hat{\mathbf{g}}$ over the same knot sequence and with the same γ_i. Denote the G^2 control vertices of $\mathbf{g}$ by $\mathbf{d}_i$, those of $\hat{\mathbf{g}}$ by $\hat{\mathbf{d}}_i$. We observe that the barycentric combination

$$\mathbf{h}(u) = (1 - \alpha)\mathbf{g}(u) + \alpha\hat{\mathbf{g}}(u)$$

is again a γ-spline curve. Moreover, the G^2 control polygon for $\mathbf{h}$ consists of the points $(1 - \alpha)\mathbf{d}_i + \alpha\hat{\mathbf{d}}_i$. A glance at Figure 12.9 reveals the truth of this statement: the points $\mathbf{d}_{i-1}$, $\mathbf{d}_i$, $\hat{\mathbf{d}}_{i-1}$, $\hat{\mathbf{d}}_i$ form a bilinear surface. Thus the Bézier points and the G^2 control vertices of $\mathbf{h}$ are related to each other in the same ratios as those of $\mathbf{g}$ and $\hat{\mathbf{g}}$, ensuring that $\mathbf{h}$ is again a γ-spline curve.

A consequence of this linearity property is that all γ-splines over the same knot sequence and with the same γ_i form a linear space whose dimension, $L + 3$, equals the number of control vertices of each γ-spline in that space. Each element of that space then has a basis representation

$$\mathbf{x}(u) = \sum_{i=-1}^{L+1} \mathbf{d}_i M_i(u). \tag{12.21}$$

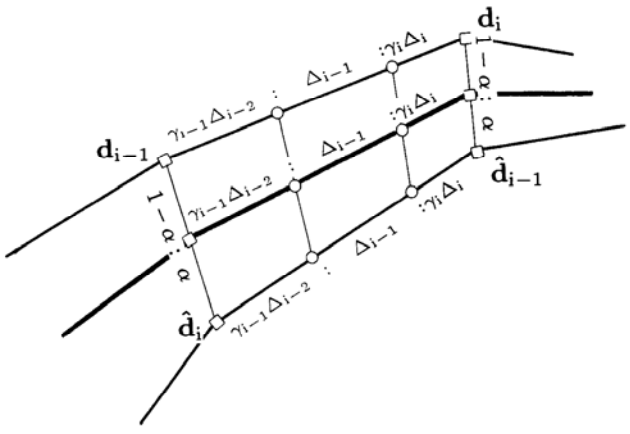

Figure 12.9: γ-splines: a barycentric combination of two γ-splines is obtained by forming the barycentric combination of their G^2 control polygons.

We are slightly negligent here: actually, the M_i depend not only on u, but also on the u_i and the γ_i.

We shall now develop several properties of the M_i until we are finally able to give an explicit form for them. As the geometry of the γ-spline construction reveals, they have the following properties:

Partition of unity: This follows since the affine invariance of the γ-spline construction implies that (12.21) is a barycentric combination:

$$\sum_{i=-1}^{L+1} M_i(u) \equiv 1. \tag{12.22}$$

Positivity: For $\gamma_i \geq 0$, the γ-spline curve lies in the convex hull of the control polygon. Thus (12.21) is a convex combination:

$$M_i(u) \geq 0. \tag{12.23}$$

Local support: If we change one $\mathbf{d}_i$, the curve is only changed over the four intervals $(u_{i-2}, \ldots, u_{i+2})$. This is illustrated in Figure 7.14 in the context of C^2 B-spline curves. Thus the corresponding basis function $M_i(u)$ must vanish outside this region:

$$M_i(u) = 0 \text{ for } u \notin [u_{i-2}, u_{i+2}]. \tag{12.24}$$

Equation (12.24) is a consequence of the fact that a change in $\mathbf{d}_i$ does not affect $\mathbf{b}_j$ with $j \leq 3i - 6$ or with $j \geq 3i + 6$. That change does not affect $\mathbf{b}_{3i\pm5}$ and $\mathbf{b}_{3i\pm4}$, either—therefore, the first and second derivatives of the curve at u_{i-2} and u_{i+2} remain unchanged. As a consequence,

$$\frac{\mathrm{d}}{\mathrm{d}u} M_i(u_{i\pm2}) = \frac{\mathrm{d}^2}{\mathrm{d}u^2} M_i(u_{i\pm2}) = 0. \tag{12.25}$$

With these properties at hand, we can now construct M_i. Consider the control polygon that is obtained by setting $\mathbf{d}_i = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ while setting all other vertices $\mathbf{d}_j = \mathbf{0}$. The graph of this polygon is quite degenerate—only one control point is nonzero. Its usefulness stems from the fact that the cross plot of the corresponding γ-spline curve consists of $\begin{bmatrix} M_i(u) \\ M_i(u) \end{bmatrix}$; in other words, it singles out exactly one basis function. We can therefore construct the Bézier points of M_i by the use of a cross plot (see Figure 12.10); if necessary, consult Sections 5.5 and 5.6. The Bézier ordinates of M_i are now a simple consequence of (12.14), (12.15), and (12.16):

$$b_{3i-2} = \frac{\gamma_{i-1}\Delta_{i-2}}{\Gamma_1}, \tag{12.26}$$

$$b_{3i-1} = \frac{\gamma_{i-1}\Delta_{i-2} + \Delta_{i-1}}{\Gamma_1}, \tag{12.27}$$

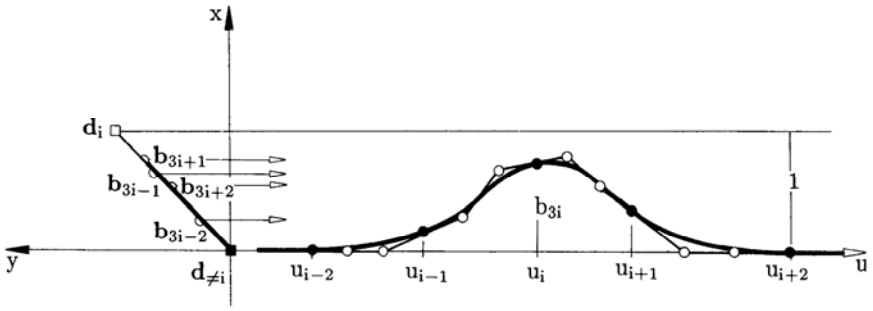

Figure 12.10: Local basis for G^2 splines: a basis function M_i is obtained through the cross plot technique. Only the plot for $x(u)$ is shown, the one for $y(u)$ being identical.

$$b_{3i+1} = \frac{\Delta_i + \gamma_{i+1}\Delta_{i+1}}{\Gamma_2}, \tag{12.28}$$

$$b_{3i+2} = \frac{\gamma_{i+1}\Delta_{i+1}}{\Gamma_2}, \tag{12.29}$$

where

$$\Gamma_1 = \gamma_{i-1}\Delta_{i-2} + \Delta_{i-1} + \gamma_i\Delta_i$$

and

$$\Gamma_2 = \gamma_i\Delta_{i-1} + \Delta_i + \gamma_{i+1}\Delta_{i+1}.$$

For the junction ordinate b_{3i} we find

$$b_{3i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i} b_{3i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i} b_{3i+1}. \tag{12.30}$$

All remaining Bézier ordinates of M_i are zero.

Historically, he first local basis for G^2 splines was developed by G. Nielson and J. Lewis [331] in 1975. In 1981, B. Barsky [34] developed a local basis for so-called β-splines, which are, in the context of this chapter, γ-splines with constant $\gamma_i = \gamma$ and a distorted uniform knot sequence with $\Delta_i = \beta\Delta_{i-1}$. Later, local bases were developed for β-spline curves that are equivalent to γ-splines (Bartels and Beatty [41]).

12.9 Higher Order Geometric Continuity

Just as we can define higher order parametric continuity C^r, we may also define higher order geometric continuity. We say that a curve is r^{th}-order geometrically continuous, or G^r, at a given point, if it can be reparametrized such that it will become C^r (see Remark 12 in Section 11.6). In particular, the new parameter might be arc length.

Figure 12.11: G^r continuity: a segment of a C^r may be reparametrized. The resulting curve is not C^r any more, but still G^r.

To derive conditions for G^r continuity, we start with a composite C^r curve $\mathbf{x}(u)$ with a global parameter u. At a given parameter value u, derivatives from the left and from the right agree:

$$\frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_- = \frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_+; \quad i = 0, \ldots, r. \tag{12.31}$$

Now let us reparametrize the right segment by introducing a new parameter $t = t(u)$; see Figure 12.11. By our earlier definition, the resulting composite curve will be G^r, while it is clearly not C^r any more. We will now study the conditions for G^r continuity using this composite G^r curve.

Modifying (12.31) so as to incorporate the new parametrization yields:

$$\frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_- = \frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}(t)_+; \quad i = 0, \ldots, r. \tag{12.32}$$

The terms on the right-hand side of this equation may be expanded using the chain rule. For $i = 1$, we obtain

$$\mathbf{x}'_- = \dot{\mathbf{x}}_+ \frac{\mathrm{d}t}{\mathrm{d}u}, \tag{12.33}$$

where a prime denotes differentiation with respect to u, and a dot denotes differentiation with respect to t. For $i = 2$, we have to apply both the chain and the product rule to the right-hand side of (12.33):

$$\mathbf{x}_-{}'' = \ddot{\mathbf{x}}_+ \left(\frac{\mathrm{d}t}{\mathrm{d}u}\right)^2 + \dot{\mathbf{x}}_+ \frac{\mathrm{d}^2 t}{\mathrm{d}u^2}. \tag{12.34}$$

For the case $i = 3$:

$$\mathbf{x}_-{}''' = \dddot{\mathbf{x}}_+ \left(\frac{\mathrm{d}t}{\mathrm{d}u}\right)^3 + 3\ddot{\mathbf{x}}_+ \frac{\mathrm{d}t}{\mathrm{d}u}\frac{\mathrm{d}^2 t}{\mathrm{d}u^2} + \dot{\mathbf{x}}_+ \frac{\mathrm{d}^3 t}{\mathrm{d}u^3}. \tag{12.35}$$

Let us define $\alpha_i = \mathrm{d}^i t/\mathrm{d}u^i$. Then the preceding equations may be written in matrix form:

$$\begin{bmatrix} \mathbf{x}'_- \\ \mathbf{x}_-{}'' \\ \mathbf{x}_-{}''' \end{bmatrix} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & \alpha_1^2 & 0 \\ \alpha_3 & 3\alpha_1\alpha_2 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_+ \\ \ddot{\mathbf{x}}_+ \\ \dddot{\mathbf{x}}_+ \end{bmatrix}. \tag{12.36}$$

The lower triangular matrix in (12.36) is called a *connection matrix*; it connects the derivatives of one segment to that of the other. For r^{th}-order geometric continuity, the connection matrix is a lower triangular $r \times r$ matrix; for more details see Gregory [255] or Goodman [235]. See also the related discussion in Section 11.6. The connection matrix is a powerful theoretical tool, and has been used to derive variation diminishing properties of geometrically continuous curves (Dyn and Micchelli [164]), to show the projective invariance of torsion continuity (Boehm [69]), and for other theoretical pursuits (Goldman and Micchelli [233]).

The above definition of geometric continuity has been used by Manning [348], Barsky [34], Barsky and DeRose [37], Degen [139], Pottmann [406], [407], and Farin [173]. In terms of classical differential geometry, the concept of "G^2" is called "order two of contact"; see do Carmo [155]. It was used in a constructive context by G. Geise [226] as early as 1962.

An interesting phenomenon arises if we consider geometric continuity of order higher than two. Consider a G^3 space curve. It is easy to verify that it possesses continuous curvature and torsion. But the converse is not true: there are space curves with continuous curvature and torsion that are not G^3 (Farin [173]). This more general class of curves, called *Frenet frame continuous*, has been studied by Boehm [67]; see also Section 11.6 and Hagen [263], [264]. They are characterized by a more general connection matrix than that for G^3 continuity; it is given by

$$\begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & \alpha_1^2 & 0 \\ \alpha_3 & \beta & \alpha_1^3 \end{bmatrix},$$

where β is an arbitrary constant. For higher order Frenet frame continuity, one has to resort to higher dimensional spaces; this has been carried out by Dyn and Micchelli [164], Goodman [235], Goldman and Micchelli [233], and Pottmann [405]; see also the survey by Gregory [255]. An even more general concept than that of Frenet frame continuity has been discussed by H. Pottmann [406].

A condition for torsion continuity of two adjacent Bézier curves with polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{c}_0, \ldots, \mathbf{c}_n$ is given by

$$\frac{\text{volume}[\mathbf{b}_{n-3}, \ldots, \mathbf{b}_n]}{||\Delta \mathbf{b}_{n-1}||^6} = \frac{\text{volume}[\mathbf{c}_0, \ldots, \mathbf{c}_3]}{||\Delta \mathbf{c}_0||^6}. \tag{12.37}$$

See Boehm [66], Farin [173], or Hagen [263].

A nice geometric interpretation of the fact that torsion continuity is more general than G^3 continuity is due to W. Boehm [66]. If $\mathbf{b}_{n-3}, \ldots, \mathbf{b}_n$ and $\mathbf{c}_0, \ldots, \mathbf{c}_3$ are given such that the two curves are G^3, can we vary $\mathbf{c}_3$ and still maintain G^3 continuity? The answer is yes, and $\mathbf{c}_3$ may be displaced by any vector parallel to the tangent spanned by $\mathbf{b}_{n-1}$ and $\mathbf{c}_1$. But we may displace $\mathbf{c}_3$ by any vector parallel to the osculating plane spanned by $\mathbf{b}_{n-2}$, $\mathbf{b}_n$, $\mathbf{c}_2$ and still maintain torsion continuity!

12.10 Implementation

We include a direct G^2 spline program. It assumes that the piecewise Bézier polygon has been determined except for the junction points $\mathbf{b}_{3i}$, which will be computed:

```
void direct_gspline(l,bez_x,bez_y)
/* From given interior Bezier points,
   the junction Bezier points b3i are found from the G2 conditions.
Input: l:            no of cubic pieces.
       bez_x,bez_y:  interior Bezier points b_{3i+1}, b_{3i-1}.
Output:bez_x,bez_y:  completed piecewise Bezier polygon.
Note:                b_0 and b_{3l+3} should be provided, too!
*/
```

12.11 Exercises

1. Figure 12.1 shows a triangle and an inscribed piecewise quadratic curve. Find the ratio of the areas enclosed by the curve and the triangle.
2. Show that the average of two G^2 piecewise cubics is in general not G^2.
3. Find an example of a G^2 torsion continuous curve that is not G^3.

*4. Let a G^3 curve consist of two cubic Bézier curves. The derivatives of the two curves at the junction point are related by a connection matrix. Work out the corresponding connection matrix for the Bézier points.

*5. Show that a nonplanar cubic cannot have zero curvature or torsion anywhere.

*6. The G^2 piecewise cubic from Figure 12.6 cannot be represented as a direct G^2 spline. Can it be obtained from a ν-spline interpolation problem?

P1. Change the programs for interpolating C^2 cubics so that they compute interpolating G^2 splines.

Chapter 13

Conic Sections

Conic sections (short: conics) have received the most attention throughout the centuries of any known curve type. Today, they are an important design tool in the aircraft industry; they are also used in areas such as font design. A great many algorithms for the use of conics in design were developed in the 1940s; two books by Liming, [334] and [335], contain detailed descriptions of those methods. A thorough development of conics can also be found in [76] and [183].

The first person to consider conics in a CAD environment was S. Coons [113]. Later, A. Forrest [211] further investigated conics and also rational cubics. We shall treat conics in the rational Bézier form; a good reference for this approach is Lee [326]. We present conics partly as a subject in its own right, but also as a first instance of rational Bézier and B-spline curves (NURBS), to be discussed later.

13.1 Projective Maps of the Real Line

Polynomial curves, as studied before, bear a close relationship to affine geometry. Consequently, the de Casteljau algorithm makes use of ratios, which are the fundamental invariant of affine maps. Thus the class of polynomial curves is invariant under affine transformations: an affine map maps a polynomial curve onto another polynomial curve.

Conic sections, and later rational polynomials, are invariant under a more general type of map: the so-called *projective maps*. These maps are studied in *projective geometry*. This is not the place to outline the ideas of that kind of geometry; the interested reader is referred to the text by Penna and Patterson [385] or to [76] and [183]. All we need here is the concept of a projective map.

We start with a map that is familiar to everybody with a background in computer graphics: the *projection*. Consider a plane (called image plane) $\mathbf{P}$ and a point $\mathbf{o}$ (called center or origin of projection) in $\mathbb{E}^3$. A point $\mathbf{p}$ is projected onto $\mathbf{P}$ through $\mathbf{o}$ by finding

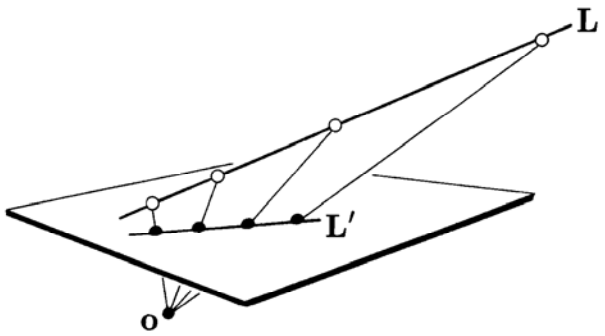

Figure 13.1: Projections: a straight line **L** is mapped onto another straight line **L**′ by a projection. Note how ratios of corresponding triples of points are distorted.

the intersection $\hat{\mathbf{p}}$ between the straight line through **o** and **p** with **P**. For a projection to be well-defined it is necessary that **o** is not in **P**. Any object in $\mathbb{E}^3$ can be projected into **P** in this manner.

In particular, we can project a straight line, **L**, say, onto **P**, as shown in Figure 13.1. We clearly see that our projection is not an affine map: the ratios of corresponding points on **L** and **L**′ are not the same. But a projection leaves another geometric property unchanged: the *cross ratio* of four collinear points.

The cross ratio, cr, of four collinear points is defined as a ratio of ratios [ratios are defined by (2.7)]:

$$\mathrm{cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \frac{\mathrm{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{d})}{\mathrm{ratio}(\mathbf{a}, \mathbf{c}, \mathbf{d})}. \tag{13.1}$$

This particular definition is only one of several equivalent ones; any permutation of the four points gives rise to a valid definition. Our convention (13.1) has the advantage of being symmetric: $\mathrm{cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \mathrm{cr}(\mathbf{d}, \mathbf{c}, \mathbf{b}, \mathbf{a})$. Cross ratios were first studied by C. Brianchon and F. Moebius, who proved their invariance under projective maps in 1827; see [361].

Let us now prove this invariance claim. We have to show, with the notation from Figure 13.2, that

$$\mathrm{cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \mathrm{cr}(\hat{\mathbf{a}}, \hat{\mathbf{b}}, \hat{\mathbf{c}}, \hat{\mathbf{d}}). \tag{13.2}$$

This fact is called the *cross ratio theorem*.

For a proof, consider Figure 13.2. Denote the area of a triangle with vertices **p**, **q**, **r** by $\Delta(\mathbf{p}, \mathbf{q}, \mathbf{r})$. We note that for instance

$$\mathrm{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \Delta(\mathbf{a}, \mathbf{b}, \mathbf{o})/\Delta(\mathbf{b}, \mathbf{c}, \mathbf{o}).$$

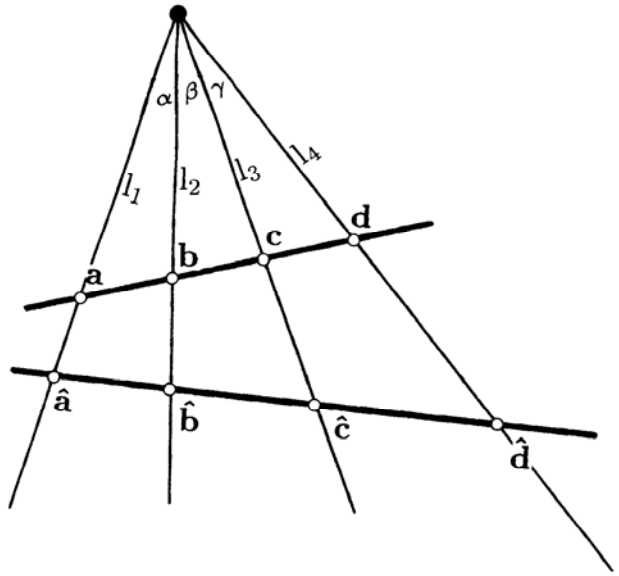

Figure 13.2: Cross ratios: the cross ratios of **a**, **b**, **c**, **d** and $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, $\hat{\mathbf{c}}$, $\hat{\mathbf{d}}$ only depend on the angles shown and are thus equal.

This gives

$$\begin{aligned}\mathrm{cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) &= \frac{\Delta(\mathbf{a}, \mathbf{b}, \mathbf{o})/\Delta(\mathbf{b}, \mathbf{d}, \mathbf{o})}{\Delta(\mathbf{a}, \mathbf{c}, \mathbf{o})/\Delta(\mathbf{c}, \mathbf{d}, \mathbf{o})} \\ &= \frac{l_1 l_2 \sin\alpha / l_2 l_4 \sin(\beta + \gamma)}{l_1 l_3 \sin(\alpha + \beta)/l_3 l_4 \sin\gamma} \\ &= \frac{\sin\alpha / \sin(\beta + \gamma)}{\sin(\alpha + \beta)/\sin\gamma}.\end{aligned}$$

Thus the cross ratio of the four points **a**, **b**, **c**, **d** only depends on the angles at **o**. The four rays emanating from **o** may therefore be intersected by any straight line; the four points of intersection will have the same cross ratio, regardless of the choice of the straight line. All such straight lines are related by projections, and we can therefore say that projections leave the cross ratio of four collinear points invariant. Since the cross ratio is the same for any straight line intersecting the given four straight lines, one also calls it the cross ratio of the four given lines.

A concept that is slightly more abstract than that of projections is that of *projective maps*. Going back to Figure 13.1, we can interpret both **L** and $\mathbf{L}'$ as copies of the real line. Then the projection of **L** onto $\mathbf{L}'$ can be viewed as a map of the real line onto itself. With this interpretation, a projection defines a projective map of the real line onto itself. On the real line, a point is given by a real number, so we can assume a correspondence between the point **a** and a real number a.

An important observation about projective maps of the real line to itself is that they are defined by three preimage and three image points. To observe this, we inspect Figure 13.2. The claim is that $\mathbf{a}$, $\mathbf{b}$, $\mathbf{d}$ and their images $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, $\hat{\mathbf{d}}$ determine a projective map. It is true since if we pick an arbitrary fourth point $\mathbf{c}$ on $\mathbf{L}$, its image $\hat{\mathbf{c}}$ on $\mathbf{L}'$ is determined by the cross ratio theorem.

A projective map of the real line onto itself is thus determined by three preimage numbers a, b, c and three image numbers $\hat{a}, \hat{b}, \hat{c}$. The projective image $\hat{t}$ of a point t can then be computed from

$$\mathrm{cr}(a, b, t, c) = \mathrm{cr}(\hat{a}, \hat{b}, \hat{t}, \hat{c}).$$

Setting $\rho = (b - a)/(c - b)$ and $\hat{\rho} = (\hat{b} - \hat{a})/(\hat{c} - \hat{b})$, this is equivalent to

$$\frac{\rho}{(t - a)/(c - t)} = \frac{\hat{\rho}}{(\hat{t} - \hat{a})/(\hat{c} - \hat{t})}.$$

Solving for $\hat{t}$:

$$\hat{t} = \frac{(t - a)\hat{\rho}\hat{c} + (c - t)\hat{a}\rho}{\rho(c - t) + \hat{\rho}(t - a)}. \tag{13.3}$$

A convenient choice for the image and preimage points is $a = \hat{a} = 0, c = \hat{c} = 1$. Equation (13.3) then takes on the simpler form

$$\hat{t} = \frac{t\hat{\rho}}{\rho(1 - t) + \hat{\rho}t}. \tag{13.4}$$

Thus a projective map of the real line onto itself corresponds to a *rational linear transformation*. It is left for the reader to verify that the projective map becomes an affine map in the special case that $\rho = \hat{\rho}$.

13.2 Conics as Rational Quadratics

We will use the following definition for conic sections: *A conic section in* $\mathbb{E}^2$ *is the projection of a parabola in* $\mathbb{E}^3$ *into a plane.* We take this plane to be the plane $z = 1$. Figure 13.3 gives an example of how to obtain a conic as the projection of a 3D parabola. Since we will study planar curves in this section, we may think of this plane as a copy of $\mathbb{E}^2$, thus identifying points $[\ x \quad y\ ]^{\mathrm{T}}$ with $[\ x \quad y \quad 1\ ]^{\mathrm{T}}$. Our special projection is characterized by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}.$$

Note that a point $[\ x \quad y\ ]^{\mathrm{T}}$ is the projection of a whole family of points: every point on the straight line $[\ wx \quad wy \quad w\ ]^{\mathrm{T}}$ projects to $[\ x \quad y\ ]^{\mathrm{T}}$. In the following, we

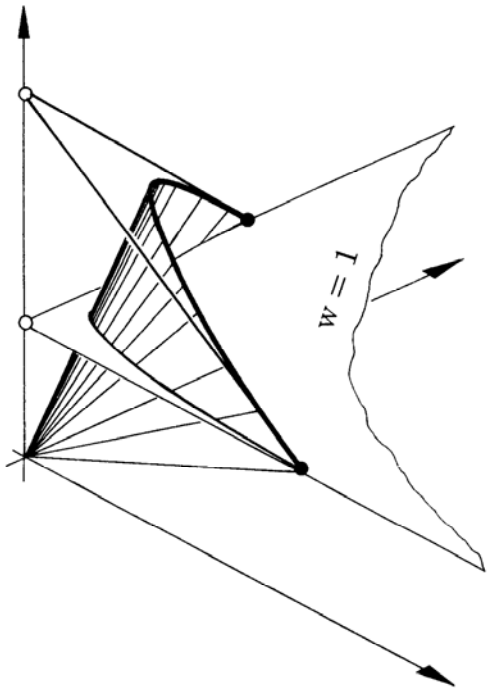

Figure 13.3: Conic sections: a parabolic arc in three-space is projected into the plane $z = 1$; the result, in this example, is part of a hyperbola.

will use the shorthand notation $[\ w\mathbf{x}\quad w\ ]^{\mathrm{T}}$ with $\mathbf{x} \in \mathbb{E}^2$ for $[\ wx\quad wy\quad w\ ]^{\mathrm{T}}$.[1] An illustration of this special projection is given in Figure 13.4.

Let $\mathbf{c}(t) \in \mathbb{E}^2$ be a point on a conic. Then there exist real numbers w_0, w_1, w_2 and points $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2 \in \mathbb{E}^2$ such that

$$\mathbf{c}(t) = \frac{w_0\mathbf{b}_0B_0^2(t) + w_1\mathbf{b}_1B_1^2(t) + w_2\mathbf{b}_2B_2^2(t)}{w_0B_0^2(t) + w_1B_1^2(t) + w_2B_2^2(t)}. \tag{13.5}$$

Let us prove (13.5). We may identify $\mathbf{c}(t) \in \mathbb{E}^2$ with $[\ \mathbf{c}(t)\quad 1\ ]^{\mathrm{T}} \in \mathbb{E}^3$. This point is the projection of a point $[\ w(t)\mathbf{c}(t)\quad w(t)\ ]^{\mathrm{T}}$, which lies on a 3D parabola. The third component $w(t)$ of this 3D point must be a quadratic function in t and may be expressed in Bernstein form:

$$w(t) = w_0B_0^2(t) + w_1B_1^2(t) + w_2B_2^2(t).$$

[1] The set of all points $[\ wx\quad wy\quad w\ ]^{\mathrm{T}}$ is called the *homogeneous form* or *homogeneous coordinates* of $[\ x\quad y\ ]^{\mathrm{T}}$.

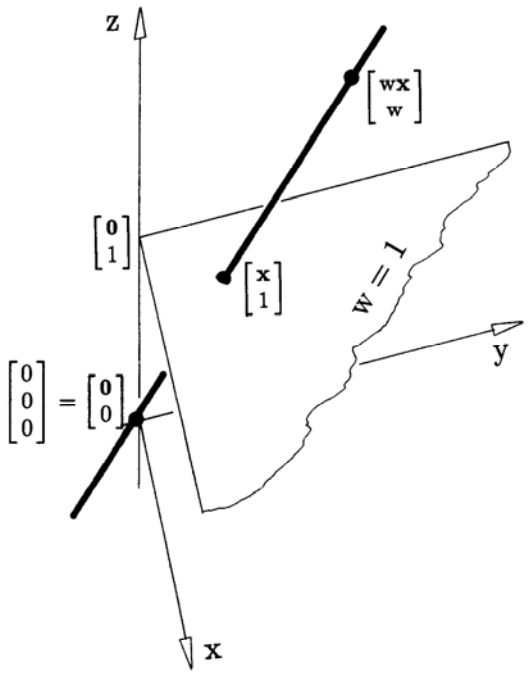

Figure 13.4: Projections: the special projection that is used to write objects in the plane $z = 1$ as projections of objects in $\mathbb{E}^3$.

Having determined $w(t)$, we may now write

$$w(t)\begin{bmatrix} \mathbf{c}(t) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{c}(t)\sum w_i B_i^2(t) \\ \sum w_i B_i^2(t) \end{bmatrix}.$$

Since the left-hand side of this equation denotes a parabola, we may write

$$\sum_{i=0}^{2} \begin{bmatrix} \mathbf{p}_i \\ w_i \end{bmatrix} B_i^2(t) = \begin{bmatrix} \mathbf{c}(t)\sum w_i B_i^2(t) \\ \sum w_i B_i^2(t) \end{bmatrix}$$

with some points $\mathbf{p}_i \in \mathbb{E}^2$. Thus

$$\sum_{i=0}^{2} \mathbf{p}_i B_i^2(t) = \mathbf{c}(t) \sum_{i=0}^{2} w_i B_i^2(t), \tag{13.6}$$

and hence

$$\mathbf{c}(t) = \frac{\mathbf{p}_0 B_0^2(t) + \mathbf{p}_1 B_1^2(t) + \mathbf{p}_2 B_2^2(t)}{w_0 B_0^2(t) + w_1 B_1^2(t) + w_2 B_2^2(t)}.$$

Setting $\mathbf{p}_i = w_i \mathbf{b}_i$ now proves (13.5).

We call the points $\mathbf{b}_i$ the *control polygon* of the conic $\mathbf{c}$; the numbers w_i are called *weights* of the corresponding control polygon vertices. Thus the conic control polygon is the projection of the control polygon with vertices $[\ w_i\mathbf{b}_i \quad w_i\ ]^{\mathrm{T}}$, which is the control polygon of the 3D parabola that we projected onto $\mathbf{c}$.

The form (13.5) is called the *rational quadratic form* of a conic section. If all weights are equal, we recover nonrational quadratics, i.e., parabolas. The influence of the weights on the shape of the conic is illustrated in Figure 13.5. In that figure, we have chosen

$$\mathbf{b}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{b}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{b}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Note that a common nonzero factor in the w_i does not affect the conic at all. If $w_0 \neq 0$, one may therefore always achieve $w_0 = 1$ by a simple scaling of all w_i. There are other changes of the weights that leave the curve shape unchanged: these correspond to *rational linear parameter transformations*. Let us set

$$t = \frac{\hat{t}}{\hat{\rho}(1-\hat{t}) + \hat{t}}, \quad (1-t) = \frac{\hat{\rho}(1-\hat{t})}{\hat{\rho}(1-\hat{t}) + \hat{t}}$$

[corresponding to the choice $\rho = 1$ in (13.4)]. We may insert this into (13.5) and obtain

$$\mathbf{c}(\hat{t}) = \frac{\hat{\rho}^2 w_0 \mathbf{b}_0 B_0^2(\hat{t}) + \hat{\rho} w_1 \mathbf{b}_1 B_1^2(\hat{t}) + w_2 \mathbf{b}_2 B_2^2(\hat{t})}{\hat{\rho}^2 w_0 B_0^2(\hat{t}) + \hat{\rho} w_1 B_1^2(\hat{t}) + w_2 B_2^2(\hat{t})}. \tag{13.7}$$

Thus, the curve shape is not changed if each weight w_i is replaced by $\hat{w}_i = \hat{\rho}^{2-i} w_i$ (for an early reference, see Forrest [211]). If, for a given set of weights w_i, we select

$$\hat{\rho} = \sqrt{\frac{w_2}{w_0}},$$

we obtain $\hat{w}_0 = w_2$, and, after dividing all three weights through by w_2, we have $\hat{w}_0 = \hat{w}_2 = 1$. A conic that satisfies this condition is said to be in *standard form*. All conics with $w_0, w_2 \neq 0$ may be rewritten in standard form with the above choice of $\hat{\rho}$, provided, of course, that $w_2/w_0 \geq 0$.

If in standard form, i.e., $w_0 = w_2 = 1$, the point $\mathbf{s} = \mathbf{c}(\frac{1}{2})$ is called the *shoulder point*. The shoulder point tangent is parallel to $\mathbf{b}_0\mathbf{b}_2$. If we set $\mathbf{m} = (\mathbf{b}_0 + \mathbf{b}_2)/2$, then the ratio of the three collinear points $\mathbf{m}, \mathbf{s}, \mathbf{b}_1$ is given by

$$\text{ratio}(\mathbf{m}, \mathbf{s}, \mathbf{b}_1) = w_1. \tag{13.8}$$

We finish this section with a theorem that will be useful in the later development of rational curves: *Any four tangents to a conic intersect each other in the same cross ratio.* The theorem is illustrated in Figure 13.6. The proof of this four tangent theorem is simple: one shows that it is true for parabolas (see Exercises). It then follows for all conics by their definition as a projection of a parabola and by the fact that cross ratios are invariant under projections. This theorem is due to J. Steiner. It is a projective version of the three-tangent theorem from Section 3.1.

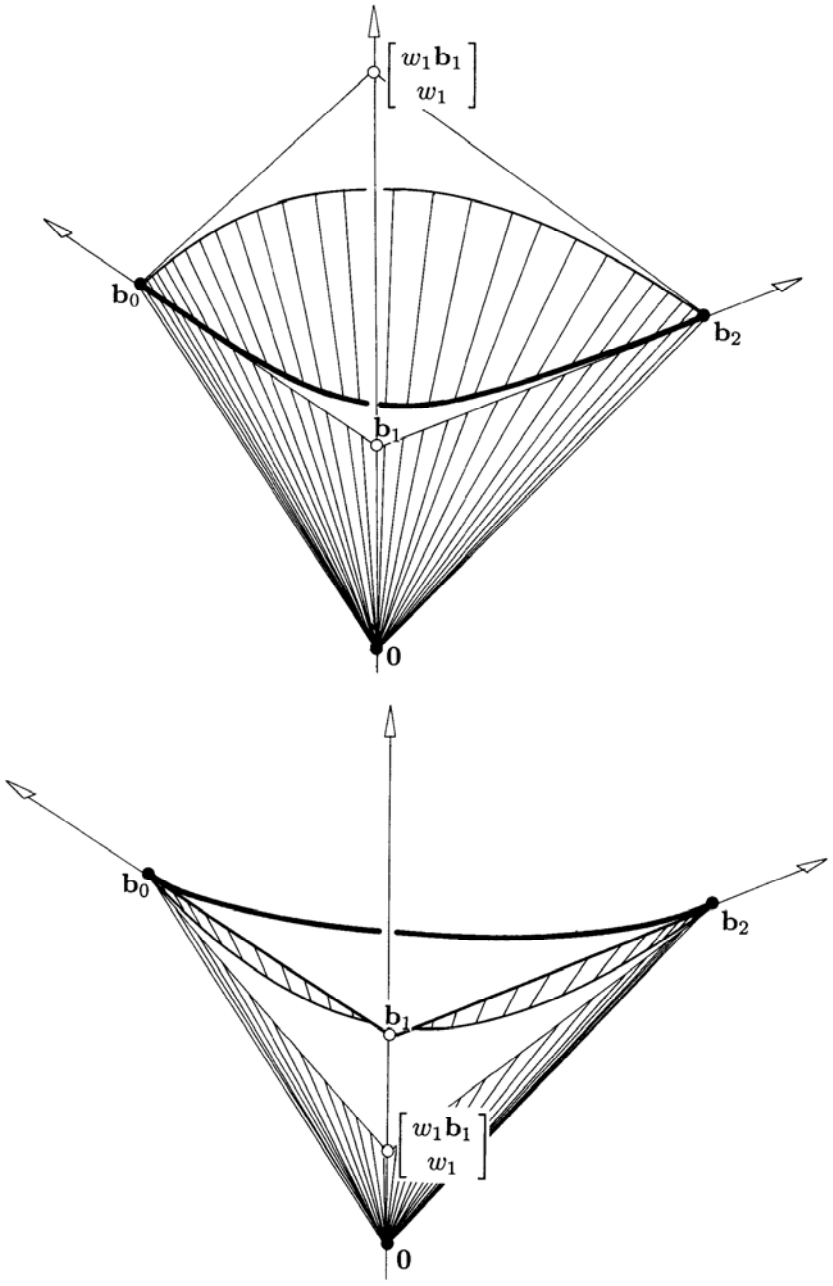

Figure 13.5: Conic sections: in the two examples shown, $w_0 = w_2 = 1$. As w_1 becomes larger, i.e., as $[w_1\mathbf{b}_1, w_1]$ moves "up" on the z-axis, the conic is "pulled" toward $\mathbf{b}_1$.

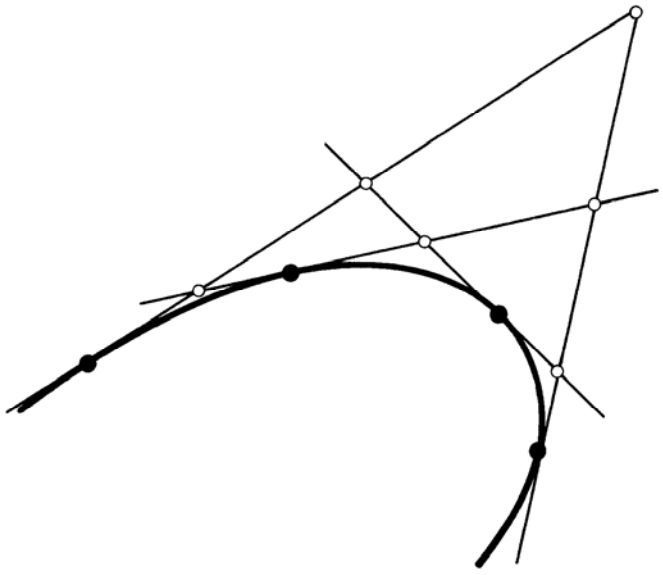

Figure 13.6: The four-tangent theorem: four points are marked on each of the four tangents to the shown conic. The four cross ratios generated by them are all equal.

13.3 A de Casteljau Algorithm

We may evaluate (13.5) by evaluating the numerator and the denominator separately and then dividing through. A more geometric algorithm is obtained by projecting each intermediate de Casteljau point $\begin{bmatrix} w_i^r \mathbf{b}_i^r & w_i^r \end{bmatrix}^{\mathrm{T}}$ into $\mathbb{E}^2$:

$$\mathbf{b}_i^r(t) = (1-t)\frac{w_i^{r-1}}{w_i^r}\mathbf{b}_i^{r-1} + t\frac{w_{i+1}^{r-1}}{w_i^r}\mathbf{b}_{i+1}^{r-1}, \tag{13.9}$$

where

$$w_i^r(t) = (1-t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t). \tag{13.10}$$

This algorithm has a strong connection to the four tangent theorem above: if we introduce *weight points*

$$\mathbf{q}_i^r(t) = \frac{w_i^r \mathbf{b}_i^r + w_{i+1}^r \mathbf{b}_{i+1}^r}{w_i^r + w_{i+1}^r}, \tag{13.11}$$

then

$$\mathrm{cr}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_i^{r+1}, \mathbf{b}_{i+1}^r) = \frac{1-t}{t} \tag{13.12}$$

assumes the same value for all r, i. While computationally more involved than the straightforward algebraic approach, this generalized de Casteljau algorithm has the advantage of being numerically stable: it uses only convex combinations, provided the weights are positive and $t \in [0, 1]$.

13.4 Derivatives

To find the derivative of a conic section, i.e., the vector $\dot{\mathbf{c}}(t) = \mathrm{d}\mathbf{c}/\mathrm{d}t$, we may employ the quotient rule. For a simpler derivation, let us rewrite (13.6) as

$$\mathbf{p}(t) = w(t)\mathbf{c}(t).$$

We apply the product rule:

$$\dot{\mathbf{p}}(t) = \dot{w}(t)\mathbf{c}(t) + w(t)\dot{\mathbf{c}}(t)$$

and solve for $\dot{\mathbf{c}}(t)$:

$$\dot{\mathbf{c}}(t) = \frac{1}{w(t)}[\dot{\mathbf{p}}(t) - \dot{w}(t)\mathbf{c}(t)]. \tag{13.13}$$

We may evaluate (13.13) at the endpoint $t = 0$:

$$\dot{\mathbf{c}}(0) = \frac{2}{w_0}[w_1\mathbf{b}_1 - w_0\mathbf{b}_0 - (w_1 - w_0)\mathbf{b}_0].$$

After some simplifications we obtain

$$\dot{\mathbf{c}}(0) = \frac{2w_1}{w_0}\Delta\mathbf{b}_0. \tag{13.14}$$

Similarly, we obtain

$$\dot{\mathbf{c}}(1) = \frac{2w_1}{w_2}\Delta\mathbf{b}_1. \tag{13.15}$$

Let us now consider two conics, one defined over the interval $[u_0, u_1]$ with control polygon $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ and weights w_0, w_1, w_2 and the other defined over the interval $[u_1, u_2]$ with control polygon $\mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ and weights w_2, w_3, w_4. Both segments form a C^1 curve if

$$\frac{w_1}{\Delta_0}\Delta\mathbf{b}_1 = \frac{w_3}{\Delta_1}\Delta\mathbf{b}_2, \tag{13.16}$$

where the appearance of the interval lengths Δ_i is due to the application of the chain rule, which is necessary since we now consider a composite curve with a global parameter u; see also Section 7.1.

13.5 The Implicit Form

Every conic $\mathbf{c}(t)$ has an *implicit representation* of the form

$$f(x, y) = 0,$$

where f is a quadratic polynomial in x and y. To find this representation, recall that $\mathbf{c}(t)$ may be written in terms of barycentric coordinates of the polygon vertices $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$:

$$\mathbf{c}(t) = \tau_0\mathbf{b}_0 + \tau_1\mathbf{b}_1 + \tau_2\mathbf{b}_2; \tag{13.17}$$

see Section 2.6. Since $\mathbf{c}(t)$ may also be written as a rational Bézier curve (13.5), and since both representations are unique, we may compare the coefficients of the $\mathbf{b}_i$:

$$\tau_0 = [w_0(1-t)^2]/D, \tag{13.18}$$

$$\tau_1 = [2w_1t(1-t)]/D, \tag{13.19}$$

$$\tau_2 = [w_2t^2]/D, \tag{13.20}$$

where $D = \sum w_iB_i^2$. We may solve (13.18) and (13.20) for $(1-t)$ and t, respectively. Inserting both expressions into (13.19) yields

$$\tau_1^2 = 4\frac{\tau_0\tau_2w_1^2}{w_0w_2}.$$

This may be written more symmetrically as

$$\frac{\tau_1^2}{\tau_0\tau_2} = \frac{4w_1^2}{w_0w_2}. \tag{13.21}$$

This is the desired implicit form, since the barycentric coordinates τ_0, τ_1, τ_2 of $\mathbf{c}(t)$ are given by

$$\tau_0 = \frac{\begin{vmatrix} c^x & b_1^x & b_2^x \\ c^y & b_1^y & b_2^y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} b_0^x & b_1^x & b_2^x \\ b_0^y & b_1^y & b_2^y \\ 1 & 1 & 1 \end{vmatrix}}, \quad \tau_1 = \frac{\begin{vmatrix} b_0^x & c^x & b_2^x \\ b_0^y & c^y & b_2^y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} b_0^x & b_1^x & b_2^x \\ b_0^y & b_1^y & b_2^y \\ 1 & 1 & 1 \end{vmatrix}}, \quad \tau_2 = \frac{\begin{vmatrix} b_0^x & b_1^x & c^x \\ b_0^y & b_1^y & c^y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} b_0^x & b_1^x & b_2^x \\ b_0^y & b_1^y & b_2^y \\ 1 & 1 & 1 \end{vmatrix}}.$$

The implicit form has an important application: suppose we are given a conic section $\mathbf{c}$ and an arbitrary point $\mathbf{x} \in \mathbb{E}^2$. Does $\mathbf{x}$ lie on $\mathbf{c}$? This question is hard to answer if $\mathbf{c}$ is given in the parametric form (13.5). Using the implicit form, this question is answered easily. First, compute the barycentric coordinates τ_0, τ_1, τ_2 of $\mathbf{x}$ with respect to $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$. Then insert τ_0, τ_1, τ_2 into (13.21). If (13.21) is satisfied, $\mathbf{x}$ lies on the conic (but see Exercises).

The implicit form is also important when dealing with the IGES data specification. In that data format, a conic is given by its implicit form $f(x, y) = 0$ and two points on it, implying a start and endpoint $\mathbf{b}_0$ and $\mathbf{b}_2$ of a conic arc. Many applications, however, need the rational quadratic form. To convert to this form, we have to determine $\mathbf{b}_1$ and its weight w_1, assuming standard form. First, we find tangents at $\mathbf{b}_0$ and $\mathbf{b}_2$: we know that the gradient of f is a vector that is perpendicular to the conic. The gradient at $\mathbf{b}_0$ is given by f's partials: $\nabla f(\mathbf{b}_0) = [f_x(\mathbf{b}_0), f_y(\mathbf{b}_0)]^{\mathrm{T}}$. The tangent is perpendicular to the gradient and thus has direction $\nabla^{\perp} f(\mathbf{b}_0) = [-f_y(\mathbf{b}_0), f_x(\mathbf{b}_0)]^{\mathrm{T}}$.

Plate I.
An automobile.
(Courtesy of Mercedes-Benz, FRG.)

Plate II.
Color rendering of the hood. *(Courtesy of Mercedes-Benz, FRG.)*

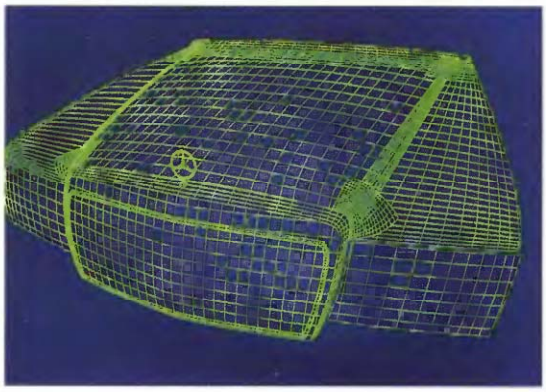

Plate III.
Wire frame rendering of the hood *(Courtesy of Mercedes-Benz, FRG.)*

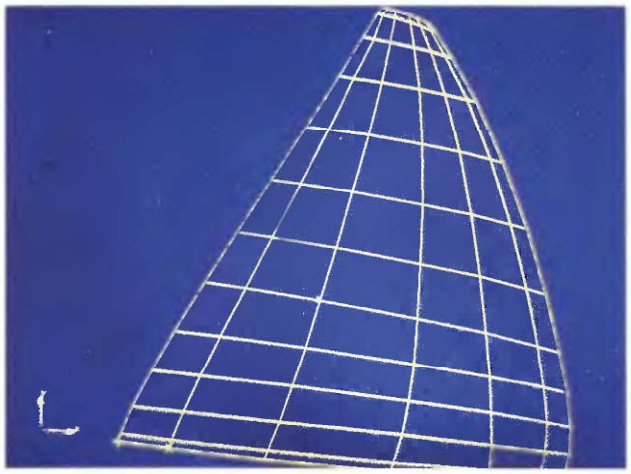

Plate IV. In a database, the hood is stored as an assembly of bicubic spline surfaces. The B-spline net of one of the surfaces is shown. *(Courtesy of Mercedes-Benz, FRG.)*

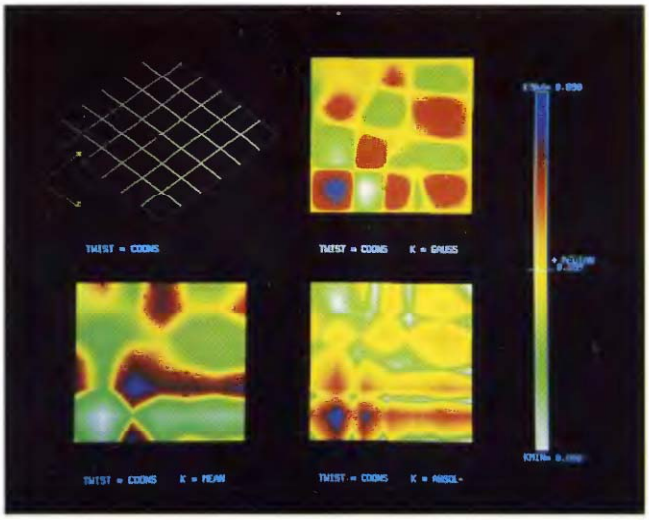

Plate V. A wire frame rendering of a surface (top left) and its Gaussian (top right), mean (bottom left), and absolute (bottom right) curvatures.

Thus our tangents are given by

$$\mathbf{t}_0(t) = \mathbf{b}_0 + t\nabla^\perp f(\mathbf{b}_0) \text{ and}$$
$$\mathbf{t}_2(s) = \mathbf{b}_2 + s\nabla^\perp f(\mathbf{b}_2).$$

Their intersection determines $\mathbf{b}_1$. Next, we compute the midpoint $\mathbf{m}$ of $\mathbf{b}_0$ and $\mathbf{b}_2$. Then the line $\overline{\mathbf{m}\mathbf{b}_1}$ will intersect our conic in the shoulder point $\mathbf{s}$. This requires the solution of a quadratic equation,[2] but then, using (13.8), we have found our desired weight w_1!

If the input is not well-defined—imagine $\mathbf{b}_0$ and $\mathbf{b}_2$ being on two different branches of a hyperbola!—then the preceding quadratic equation may have complex solutions. An error flag would be appropriate here. If the arc between $\mathbf{b}_0$ and $\mathbf{b}_2$ subtends an angle larger than, say, 120 degrees, it should be subdivided. For more details, see [502].

Any conic section is uniquely determined by five distinct points in the plane. If the points have coordinates $(x_1, y_1), \ldots, (x_5, y_5)$, the implicit form of the interpolating conic is given by

$$f(x,y) = \begin{vmatrix} x^2 & xy & y^2 & x & y & 1 \\ x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{vmatrix} = 0.$$

The fact that five points are sufficient to determine a conic is a consequence of the most fundamental theorem in the theory of conics, *Pascal's theorem*. Consider six points on a conic, arranged as in Figure 13.7. If we connect the points as shown,

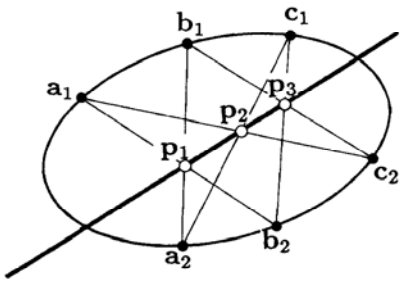

Figure 13.7: Pascal's theorem: the intersection points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ of the indicated pairs of straight lines are collinear.

[2]The quadratic equation will in general have two solutions. We take the one inside the triangle $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$.

we form six straight lines. Pascal's theorem states that the three intersection points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are always collinear.

It can be used to *construct* a conic through five points: referring to Figure 13.7 again, let $\mathbf{a}_1, \mathbf{b}_1, \mathbf{c}_1, \mathbf{a}_2, \mathbf{b}_2$ be given (no three of them collinear). Let $\mathbf{p}_1$ be the intersection of the two straight lines through $\mathbf{a}_1, \mathbf{b}_2$ and $\mathbf{a}_2, \mathbf{b}_1$. We may now fix a line $\mathbf{l}$ through $\mathbf{p}_1$, thus obtaining $\mathbf{p}_2$ and $\mathbf{p}_3$. The sixth point on the conic is then determined as the intersection of the two straight lines through $\mathbf{a}_1, \mathbf{p}_2$ and $\mathbf{b}_1, \mathbf{p}_3$. We may construct arbitrarily many points on the conic by letting the straight line $\mathbf{l}$ rotate around $\mathbf{p}_1$.

13.6 Two Classic Problems

A large number of methods exist to construct conic sections from given pieces of information, most based on Pascal's theorem. A nice collection is given in a book by R. Liming [335]. An in-depth discussion of those methods is beyond the scope of this book; we restrict ourselves to the solution of two problems.

1. Conic from two points and tangents plus another point. The given data amount to prescribing $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$. The missing weight w_1 must be determined from the point $\mathbf{p}$, which is assumed to be on the conic. We assume, without loss of generality, that the conic is in standard form ($w_0 = w_2 = 1$).

For the solution, we make use of the implicit form (13.21). We can easily determine the barycentric coordinates τ_0, τ_1, τ_2 of $\mathbf{p}$ with respect to the triangle formed by the three $\mathbf{b}_i$. We can then solve (13.21) for the unknown weight w_1:

$$w_1 = \frac{\tau_1}{2\sqrt{\tau_0 \tau_2}}. \tag{13.22}$$

If $\mathbf{p}$ is inside the triangle formed by $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$, then (13.22) always has a solution. Otherwise, problems might occur (see Exercises). If we do not insist on the conic in standard form, the given point may be given the parameter value $t = \frac{1}{2}$, in which case it is referred to as a *shoulder point*.

2. Conic from two points and tangents plus a third tangent. Again, we are given the Bézier polygon of the conic plus a tangent, which passes through two points that we call $\mathbf{b}_0^1$ and $\mathbf{b}_1^1$. We have to find the interior weight w_1, assuming the conic will be in standard form. The unknown weight w_1 determines the two weight points $\mathbf{q}_0$ and $\mathbf{q}_1$, with $\overline{\mathbf{q}_0\mathbf{q}_1}$ parallel to $\overline{\mathbf{b}_0\mathbf{b}_2}$; see Figure 13.8.

We compute the ratios $r_0 = \text{ratio}(\mathbf{b}_0, \mathbf{b}_0^1, \mathbf{b}_1)$ and $r_1 = \text{ratio}(\mathbf{b}_1, \mathbf{b}_1^1, \mathbf{b}_2)$. From the definition of the $\mathbf{q}_i$ in (13.11), it follows that $\text{ratio}(\mathbf{b}_0, \mathbf{q}_0, \mathbf{b}_1) = w_1$ and $\text{ratio}(\mathbf{b}_1, \mathbf{q}_1, \mathbf{b}_2) = 1/w_1$. The cross ratio property (13.12) now yields

$$\frac{r_0}{w_1} = r_1 w_1, \tag{13.23}$$

from which we easily determine $w_1 = \sqrt{r_0/r_1}$. The number under the square root must be nonnegative for this to be meaningful (see Exercises). Again, if we do not

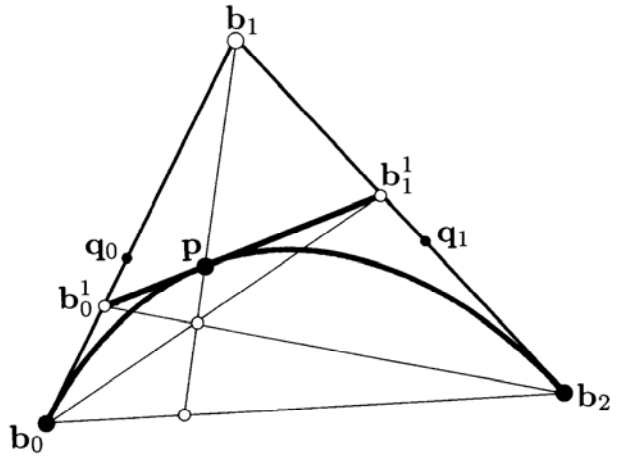

Figure 13.8: Conic constructions: $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$, and the tangent through $\mathbf{b}_0^1$ and $\mathbf{b}_1^1$ are given.

insist on standard form, we may associate the parameter value $t = \frac{1}{2}$ with the given tangent—it is then called a *shoulder tangent*.

Figure 13.8 also gives a strictly geometric construction: intersect lines $\overline{\mathbf{b}_0\mathbf{b}_1^1}$ and $\overline{\mathbf{b}_2\mathbf{b}_0^1}$. Connect the intersection with $\mathbf{b}_1$ and intersect with the given tangent: the intersection is the desired point $\mathbf{p}$.

13.7 Classification

In a projective environment, all conics are equivalent: projective maps map conics to conics. In affine geometry, conics fall into three classes: hyperbolas, parabolas, and ellipses. Thus ellipses are mapped to ellipses under affine maps, parabolas to parabolas, and hyperbolas to hyperbolas. How can we determine what type a given conic is?

Before we answer that question (following Lee [326]), let us consider the *complementary segment* of a conic. If the conic is in standard form, it is obtained by reversing the sign of w_1. Note that the implicit form (13.21) is not affected by this; hence we still have the same conic, but with a different representation. If $\mathbf{c}(t)$ is a point on the original conic and $\hat{\mathbf{c}}(t)$ is a point on the complementary segment, one easily verifies that $\mathbf{b}_1$, $\mathbf{c}(t)$, and $\hat{\mathbf{c}}(t)$ are collinear, as shown in Figure 13.9. If we assume that $w_1 > 0$, then the behavior of $\hat{\mathbf{c}}(t)$ determines what type the conic is: if $\hat{\mathbf{c}}(t)$ has no singularities in $[0, 1]$, it is an ellipse; if it has one singularity, it is a parabola; and if it has two singularities, it is a hyperbola.

The singularities, corresponding to points at infinity of $\hat{\mathbf{c}}(t)$, are determined by the real roots of the denominator $\hat{w}(t)$ of $\hat{\mathbf{c}}(t)$. There are at most two real roots, and

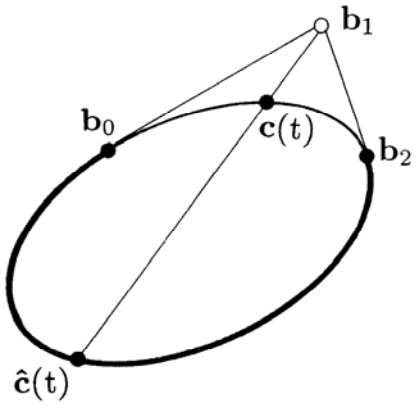

Figure 13.9: The complementary segment: the original conic segment and the complementary segment, both evaluated for all parameter values $t \in [0, 1]$, comprise the whole conic section.

they are given by

$$t_{1,2} = \frac{1 + w_1 \pm \sqrt{w_1^2 - 1}}{2 + 2w_1}.$$

Thus, a conic is an ellipse if $w_1 < 1$, a parabola if $w_1 = 1$, and a hyperbola if $w_1 > 1$. The three types of conics are shown in Figure 13.10 (see also Figure 13.5).

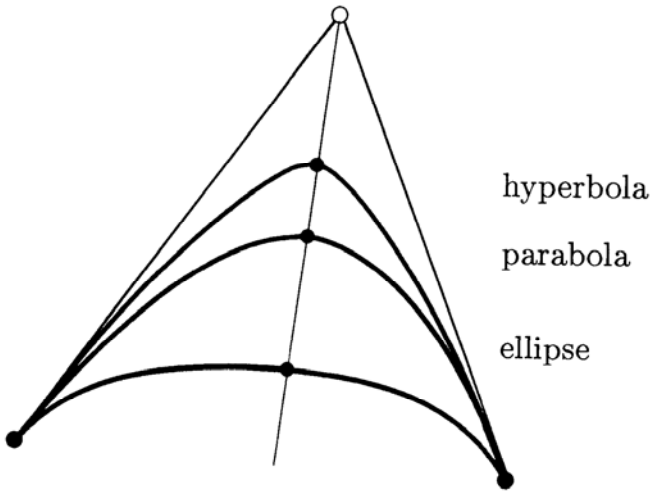

Figure 13.10: Conic classification: the three types of conics are obtained by varying the center weight w_1, assuming $w_0 = w_2 = 1$.

The circle is one of the more important conic sections; let us now pay some special attention to it. Let our rational quadratic (with $w_1 < 1$) describe an arc of a circle. Because of the symmetry properties of the circle, the control polygon must form an isosceles triangle. If we know the angle $\alpha = \angle(\mathbf{b}_2, \mathbf{b}_0, \mathbf{b}_1)$, we should be able to determine the weight w_1.[3] We may utilize the solution to the second problem in Section 13.6 together with some elementary trigonometry and obtain

$$w_1 = \cos\alpha.$$

A whole circle can be represented by piecing several such arcs together. For example, we might choose to represent a circle by three equal arcs, resulting in a configuration like that shown in Figure 13.11. The angles α equal 60 degrees, and so the weights of the inner Bézier points are $\frac{1}{2}$, whereas the junction Bézier points have weights of unity, since each arc is in standard form.

Our representation of the circle is C^1, assuming uniform parameter intervals; see (13.16). It is not C^2, however! Still we have an *exact* representation of the circle, not an approximation. Thus this particular representation of the circle is an example of a G^2 curve.

We should mention that the parametrization of our circle is not the arc length parametrization as explained in Chapter 11. If uniform traversal of the circle is necessary for some application, one has no choice but to resort to the classical sine and cosine representation. It can be shown (Farouki and Sakkalis [198]) that no

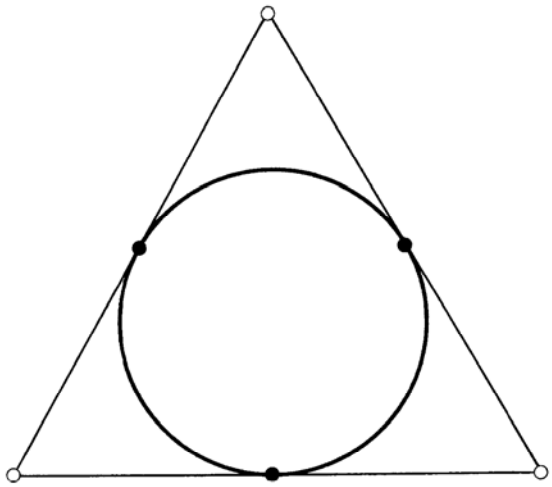

Figure 13.11: Circles: a whole circle may be written as three rational Bézier quadratics.

[3]The actual size of the control polygon does not matter, of course: it can be changed by a scaling to any size we want, and scalings do not affect the weights!

rational curve other than the straight line is parametrized with respect to arc length when evaluated at equal increments of its parameter t, and the curve will not be traced out at uniform speed.

13.8 Control Vectors

In principle, any arc of a conic may be written as a rational quadratic curve segment (possibly with negative weights). But what happens for the case where the tangents at $\mathbf{b}_0$ and $\mathbf{b}_2$ become parallel? Intuitively, this would send $\mathbf{b}_1$ to infinity. A little bit of analysis will overcome this problem, as we shall see from the following example.

Let a conic be given by $\mathbf{b}_0 = [-1, 0]^{\mathrm{T}}$, $\mathbf{b}_2 = [1, 0]^{\mathrm{T}}$, and $\mathbf{b}_1 = [0, \tan\alpha]^{\mathrm{T}}$ and a weight $w_1 = c\cos\alpha$ (we assume standard form). The angle α is formed by $\mathbf{b}_0\mathbf{b}_1$ and $\mathbf{b}_0\mathbf{b}_2$ at $\mathbf{b}_0$. Note that for $c = 1$, we obtain a circular arc, as illustrated in Figure 13.12.

The equation of our conic is given by

$$\mathbf{c}(t) = \frac{(1-t)^2 \begin{bmatrix} -1 \\ 0 \end{bmatrix} + \cos\alpha \cdot 2ct(1-t) \begin{bmatrix} 0 \\ \tan\alpha \end{bmatrix} + t^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{(1-t)^2 + 2ct(1-t)\cos\alpha + t^2}.$$

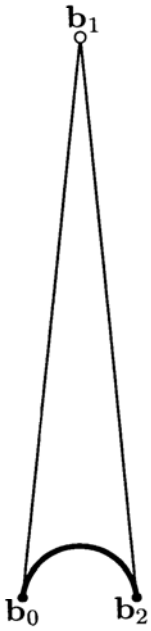

Figure 13.12: Conic arcs: a 168 degree arc of a circle is shown. Note that α is close to 90 degrees.

What happens as α tends to $\frac{\pi}{2}$? For the limiting conic, we obtain the equation

$$\mathbf{c}(t) = \frac{(1-t)^2 \begin{bmatrix} -1 \\ 0 \end{bmatrix} + 2t(1-t) \begin{bmatrix} 0 \\ c \end{bmatrix} + t^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{(1-t)^2 + t^2}. \tag{13.24}$$

The problem of a weight tending to zero and a control point tending to infinity has thus been resolved. For $c = 1$, we obtain a semicircle; other values of c give rise to different conics. For $c = -1$, we obtain the "lower" half of the unit circle.

We have been able to overcome possible problems with parallel end tangents. But there is a price to be paid: if we look at (13.24) closely, we see that it does not constitute a barycentric combination any more! The factors of $\mathbf{b}_0$ and $\mathbf{b}_2$ sum to one identically, hence $[0, c]^{\mathrm{T}}$ must be interpreted as a *vector*. Thus (13.24) contains both control points and control vectors.[4] An important property of Bézier curves is thus lost; namely, the convex hull property: it is only defined for point sets, not for a potpourri of points and vectors.

The use of control vectors allows a very compact form of writing a semi-circle. But two disadvantages argue against its use: first, the loss of the convex hull property. Second: to write the control vector form in the context of "normal" rational quadratics, one will have to resort to a special case treatment. We shall see later (Section 14.6) how to avoid the use of the control vector form.

13.9 Implementation

The following routine solves the first problem in Section 13.6:

```
 float conic_weight(b0,b1,b2,p)
/*
  Input:b0,b1,b2:  conic control polygon vertices
        p:         point on conic
  Output:          weight of b1 (assuming standard form).

  Note:            will crash in "forbidden" situations.
*/
```

13.10 Exercises

1. Equation (13.22) does not always have a solution. Identify the "forbidden" regions for the third point $\mathbf{p}$ on the conic.
2. In the same manner, investigate (13.23).
3. Prove that the four-tangent theorem holds for parabolas.

[4] In projective geometry, vectors are sometimes called "points at infinity." This has given rise to the name "infinite control points" by Vesprille [492]; see also L. Piegl [397]. We prefer the term "control vector" since this allows us to distinguish between $[0, c]^{\mathrm{T}}$ and $[0, -c]^{\mathrm{T}}$.

*4. Establish the connection between (13.12) and the four-tangent theorem.

*5. Our discussion of the implicit form (13.21) was somewhat academic: in a "real-life" situation, (13.21) will never be satisfied *exactly*. Discuss the tolerance problem that arises here, i.e., how closely does (13.21) have to be satisfied for a point to be within a given tolerance to the conic?

P1. Write a routine to iteratively subdivide a conic, putting each piece into standard form. The middle weights will converge to unity. How do the convergence rates depend on the type of the intial conic? (See also [342].)

P2. Write a routine to approximate a given Bézier curve by a sequence of elliptic arcs within a given tolerance.

Chapter 14

Rational Bézier and B-spline Curves

Rational B-spline curves[1] have become the standard curve and surface description in the field of CAD and graphics. The use of rational curves in CAGD may be traced back to Coons [113], [115], and Forrest [211]. By now, there are books on NURBS: Fiorot and Jeannin [204], Farin [183], Piegl and Tiller [401].

14.1 Rational Bézier Curves

In the previous chapter, we obtained a conic section in $\mathbb{E}^2$ as the projection of a parabola (a quadratic) in $\mathbb{E}^3$. Conic sections may be expressed as rational quadratic (Bézier) curves, and their generalization to higher degree rational curves is quite straightforward: a rational Bézier curve of degree n in $\mathbb{E}^3$ is the projection of an n^{th}-degree Bézier curve in $\mathbb{E}^4$ into the hyperplane $w = 1$. We may view this 4D hyperplane as a copy of $\mathbb{E}^3$; we assume that a point in $\mathbb{E}^4$ is given by its coordinates $\begin{bmatrix} x & y & z & w \end{bmatrix}^{\mathsf{T}}$. Proceeding in exactly the same way as we did for conics, we can show that an n^{th}-degree rational Bézier curve is given by

$$\mathbf{x}(t) = \frac{w_0\mathbf{b}_0 B_0^n(t) + \cdots + w_n\mathbf{b}_n B_n^n(t)}{w_0 B_0^n(t) + \cdots + w_n B_n^n(t)}; \quad \mathbf{x}(t), \mathbf{b}_i \in \mathbb{E}^3. \tag{14.1}$$

The w_i are again called *weights*; the $\mathbf{b}_i$ form the control polygon. It is the projection of the 4D control polygon $\begin{bmatrix} w_i\mathbf{b}_i & w_i \end{bmatrix}^{\mathsf{T}}$ of the nonrational 4D preimage of $\mathbf{x}(t)$.

[1] Often called NURBS for *nonuniform rational B-splines*.

If all weights equal one, we obtain the standard nonrational Bézier curve, since the denominator is identically equal to one.[2] If some w_i are negative, singularities may occur; we will therefore deal only with nonnegative w_i. Rational Bézier curves enjoy all the properties that their nonrational counterparts possess; for example, they are affinely invariant. We can see this by rewriting (14.1) as

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

We see that the basis functions

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

sum to one identically, thus asserting affine invariance. If all w_i are nonnegative, we have the convex hull property. We also have symmetry, invariance under affine parameter transformations, endpoint interpolation, and the variation diminishing property. Obviously, the conic sections from the preceding chapter are included in the set of all rational Bézier curves, further justifying their increasing popularity.

The w_i are typically used as *shape parameters*. If we increase one w_i, the curve is pulled toward the corresponding $\mathbf{b}_i$, as illustrated in Figure 14.1. Note that the effect of changing a weight is different from that of moving a control vertex, illustrated in Figure 14.1. If we let all weights tend to infinity at the same rate, we do *not* approach the control polygon since a common (if large) factor in the weights does not matter—the rational Bézier curve shape parameters behave differently from γ- or ν-spline shape parameters.

Two properties differ from the nonrational case. First, we have *projective* invariance. That is, if a rational Bézier curve is transformed by a projective transformation, we could just as well apply that transformation to the control polygon (using its weights to write it in homogeneous form) and would end up with the same curve. Note that nonrational curves only have this property for a subset of all projective maps, i.e., the affine maps.

The second difference is the *linear precision* property. Rational curves may have all Bézier points $\mathbf{b}_i$ distributed on a straight line in a totally arbitrary fashion:

$$\mathbf{b}_i = (1 - \alpha_i)\mathbf{b}_0 + \alpha_i \mathbf{b}_n; \quad i = 1, \ldots, n-1$$

with arbitrary real numbers α_i. We can still find weights w_i such that the resulting curve traces out the straight line $\overline{\mathbf{b}_0\mathbf{b}_n}$ in a *linear* fashion. They are given by $w_0 = 1$ and

$$w_i = \frac{i}{n+1-i} \frac{1-\alpha_{i-1}}{\alpha_i} w_{i-1}; \quad i = 1, \ldots, n.$$

For proofs, see [187] and [205].

[2] This is also true if the weights are not unity, but are equal to each other—a common factor does not matter.

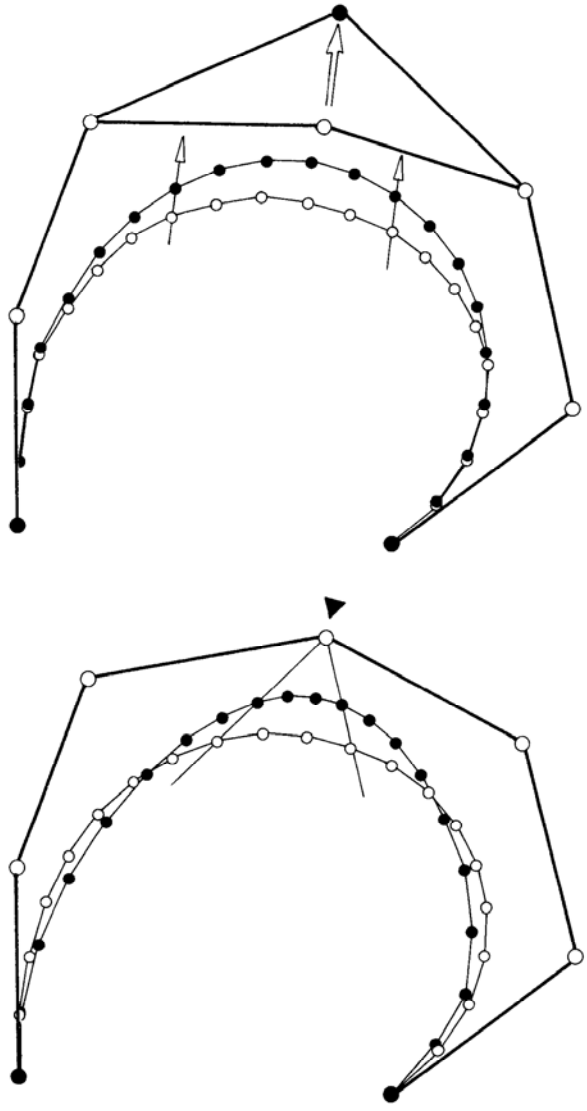

Figure 14.1: Influence of the weights: top, changing one control point; bottom, changing one weight.

14.2 The de Casteljau Algorithm

A rational Bézier curve may be evaluated by applying the de Casteljau algorithm to both numerator and denominator and finally dividing through. A warning is appropriate: while simple and usually effective, this method is not numerically stable for weights that vary significantly in magnitude. If some of the w_i are large, the 3D intermediate points $[w_i\mathbf{b}_i]^r$ (interpreted as points in a given coordinate system) are no longer in the convex hull of the original control polygon $\{\mathbf{b}_i\}$; this may result in a loss of accuracy.[3]

An expensive yet more geometric technique is to project every intermediate de Casteljau point $\begin{bmatrix} w_i\mathbf{b}_i & w_i \end{bmatrix}^{\mathsf{T}}$; $\mathbf{b}_i \in \mathbb{E}^3$ into the hyperplane $w = 1$. This yields the rational de Casteljau algorithm (see Farin [174]):

$$\mathbf{b}_i^r(t) = (1-t)\frac{w_i^{r-1}}{w_i^r}\mathbf{b}_i^{r-1} + t\frac{w_{i+1}^{r-1}}{w_i^r}\mathbf{b}_{i+1}^{r-1}, \tag{14.2}$$

with

$$w_i^r(t) = (1-t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t). \tag{14.3}$$

An explicit form for the intermediate points $\mathbf{b}_i^r$ is given by

$$\mathbf{b}_i^r(t) = \frac{\sum_{j=0}^r w_{i+j}\mathbf{b}_{i+j}B_j^r(t)}{\sum_{j=0}^r w_{i+j}B_j^r(t)}.$$

Note that for positive weights, the $\mathbf{b}_i^r$ are all in the convex hull of the original $\mathbf{b}_i$, thus assuring numerical stability.

The rational de Casteljau algorithm allows a nice geometric interpretation. While the standard de Casteljau algorithm makes use of ratios of three points, this one makes use of the *cross ratio of four points*. Let us define points $\mathbf{q}_i^r(t)$, which are located on the straight lines joining $\mathbf{b}_i^r$ and $\mathbf{b}_{i+1}^r$, subdividing them in the ratios

$$\text{ratio}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_{i+1}^r) = \frac{w_{i+1}^r}{w_i^r}.$$

We shall call these points *weight points*, because they indicate the relative magnitude of the weights in a geometric way. Then all of the following cross ratios are equal:

$$\text{cr}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_i^{r+1}, \mathbf{b}_{i+1}^r) = \frac{1-t}{t} \quad \text{for all } r, i.$$

For $r = 0$, the weight points

$$\mathbf{q}_i = \mathbf{q}_i^0 = \frac{w_i\mathbf{b}_i + w_{i+1}\mathbf{b}_{i+1}}{w_i + w_{i+1}}$$

[3] These points are obtained by applying the de Casteljau algorithm to the control points $w_i\mathbf{b}_i$ of the numerator of (14.1). They have no true geometric interpretation, because their location is not invariant under translations of the original control polygon.

Figure 14.2: Convex hulls: if the weight points are used, tighter bounds on the curve are possible.

[4]To be precise, we can only find them modulo an—immaterial—common factor.

[5]This situation is similar to the way curves are generated using the direct G^2 spline algorithm from Chapter 12 compared to the generation of γ-splines.