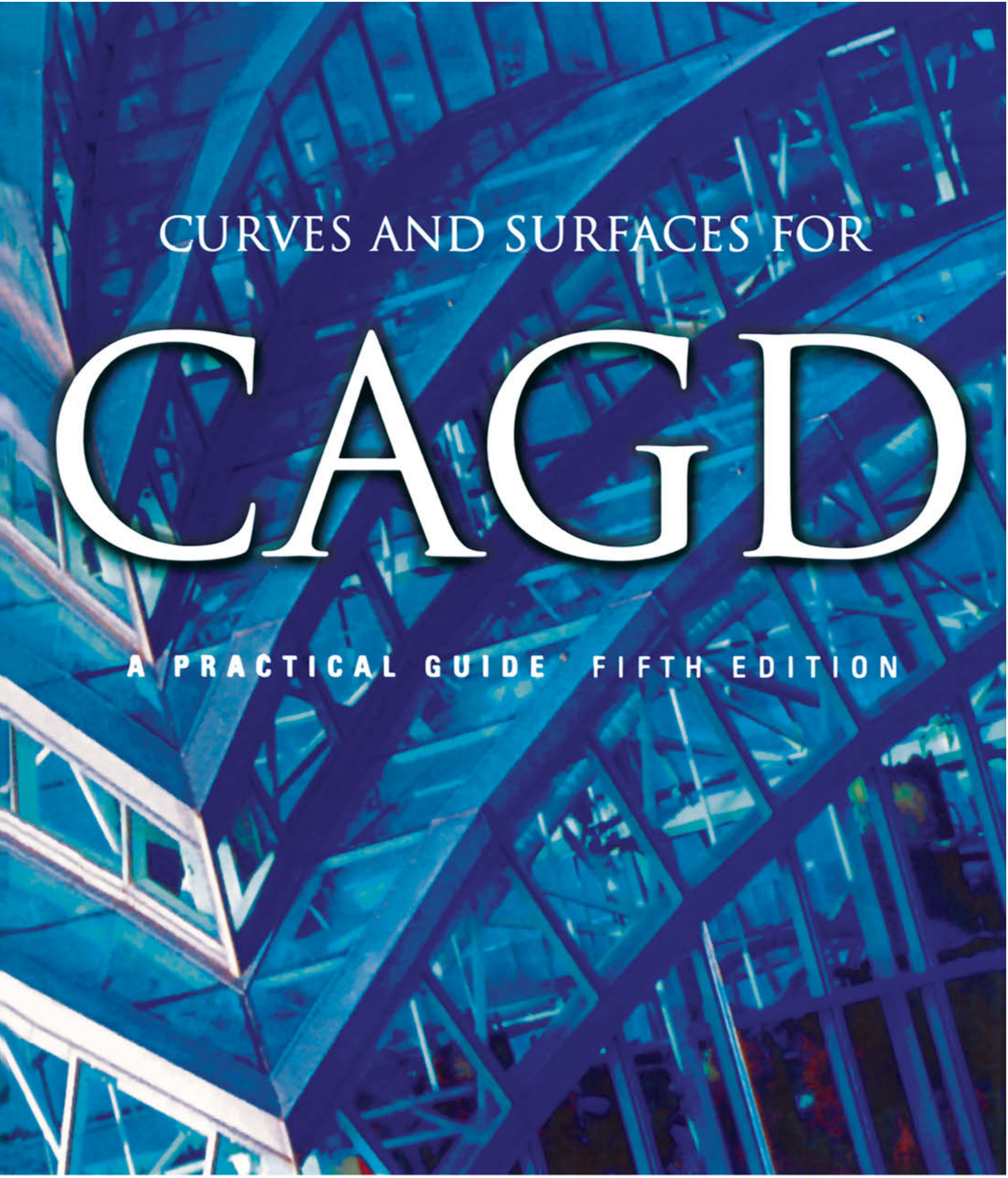


CURVES AND SURFACES FOR

CAGD

A PRACTICAL GUIDE FIFTH EDITION

GERALD FARIN

Curves and Surfaces for CAGD

A Practical Guide

Fifth Edition

The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling

Series Editor: Brian A. Barsky, University of California, Berkeley

Curves and Surfaces for CAGD: A Practical Guide, Fifth Edition
Gerald Farin

Subdivision Methods for Geometric Design: A Constructive Approach
Joe Warren and Henrik Weimer

The Computer Animator's Technical Handbook
Lynn Pocock and Judson Rosebush

Computer Animation: Algorithms and Techniques
Rick Parent

Advanced RenderMan: Creating CGI for Motion Pictures
Anthony A. Apodaca and Larry Gritz

Curves and Surfaces in Geometric Modeling: Theory and Algorithms
Jean Gallier

Andrew Glassner's Notebook: Recreational Computer Graphics
Andrew S. Glassner

Warping and Morphing of Graphical Objects
Jonas Gomes, Lucia Darsa, Bruno Costa, and Luis Velho

Jim Blinn's Corner: Dirty Pixels
Jim Blinn

Rendering with Radiance: The Art and Science of Lighting Visualization
Greg Ward Larson and Rob Shakespeare

Introduction to Implicit Surfaces
Edited by Jules Bloomenthal

Jim Blinn's Corner: A Trip Down the Graphics Pipeline
Jim Blinn

Interactive Curves and Surfaces: A Multimedia Tutorial on CAGD
Alyn Rockwood and Peter Chambers

Wavelets for Computer Graphics: Theory and Applications
Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin

Principles of Digital Image Synthesis
Andrew S. Glassner

Radiosity & Global Illumination
François X. Sillion and Claude Puech

Knotty: A B-Spline Visualization Program
Jonathan Yen

User Interface Management Systems: Models and Algorithms
Dan R. Olsen, Jr.

Making Them Move: Mechanics, Control, and Animation of Articulated Figures
Edited by Norman I. Badler, Brian A. Barsky, and David Zeltzer

Geometric and Solid Modeling: An Introduction
Christoph M. Hoffmann

An Introduction to Splines for Use in Computer Graphics and Geometric Modeling
Richard H. Bartels, John C. Beatty, and Brian A. Barsky

Curves and Surfaces for CAGD

A Practical Guide

Fifth Edition

Gerald Farin
Arizona State University

MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ACADEMIC PRESS
A Harcourt Science and Technology Company
SAN FRANCISCO SAN DIEGO NEW YORK BOSTON
LONDON SYDNEY TOKYO

Executive Director Diane D. Cerra
Publishing Services Manager Scott Norton
Senior Production Editor Cheri Palmer
Assistan Editor Belinda Breyer
Editorial Assistant Mona Buehler
Cover Design Yvo Riezbos Design
Cover Image © 1999 Jeff Greenberg / Rainbow / PictureQuest
Glass, curved rods of steel meet in the center to form a shape similar to the bow of a boat at the Tampa Aquarium, Tampa, Florida.
Text Design Rebecca Evans & Associates
Composition Windfall Software, using ZzTEX
Technical Illustration LM Graphics
Copyeditor Robert Fiske
Proofreader Erin Milnes
Printer Edwards Brothers

Morgan Kaufmann Publishers
340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205, USA
http://www.mkp.com

ACADEMIC PRESS
A Harcourt Science and Technology Company
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA
http://www.academicpress.com

Academic Press
Harcourt Place, 32 Jamestown Road, London, NW1 7BY, United Kingdom
http://www.academicpress.com

06 05 04 03 02 5 4 3 2 1

Library of Congress Control Number: 2001094373
ISBN: 1-55860-737-4

This book is printed on acid-free paper.

To My Parents

This Page Intentionally Left Blank

Contents

Preface **xv**

Chapter 1 **P. Bézier: How a Simple System Was Born** **1**

Chapter 2 **Introductory Material** **13**

2.1 Points and Vectors 13

2.2 Affine Maps 17

2.3 Constructing Affine Maps 20

2.4 Function Spaces 22

2.5 Problems 24

Chapter 3 **Linear Interpolation** **25**

3.1 Linear Interpolation 25

3.2 Piecewise Linear Interpolation 29

3.3 Menelaos' Theorem 30

3.4 Blossoms 31

3.5 Barycentric Coordinates in the Plane 34

3.6 Tessellations 37

3.7 Triangulations 39

3.8 Problems 41

Chapter 4 **The de Casteljau Algorithm** **43**

4.1 Parabolas 43

4.2 The de Casteljau Algorithm 45

4.3 Some Properties of Bézier Curves 47

4.4 The Blossom 50
4.5 Implementation 53
4.6 Problems 54

Chapter 5 **The Bernstein Form of a Bézier Curve** **57**
5.1 Bernstein Polynomials 57
5.2 Properties of Bézier Curves 60
5.3 The Derivatives of a Bézier Curve 62
5.4 Domain Changes and Subdivision 68
5.5 Composite Bézier Curves 71
5.6 Blossom and Polar 73
5.7 The Matrix Form of a Bézier Curve 74
5.8 Implementation 75
5.9 Problems 78

Chapter 6 **Bézier Curve Topics** **81**
6.1 Degree Elevation 81
6.2 Repeated Degree Elevation 83
6.3 The Variation Diminishing Property 84
6.4 Degree Reduction 85
6.5 Nonparametric Curves 86
6.6 Cross Plots 87
6.7 Integrals 88
6.8 The Bézier Form of a Bézier Curve 89
6.9 The Weierstrass Approximation Theorem 90
6.10 Formulas for Bernstein Polynomials 91
6.11 Implementation 92
6.12 Problems 93

Chapter 7 **Polynomial Curve Constructions** **95**
7.1 Aitken's Algorithm 95
7.2 Lagrange Polynomials 98
7.3 The Vandermonde Approach 99
7.4 Limits of Lagrange Interpolation 101
7.5 Cubic Hermite Interpolation 102
7.6 Quintic Hermite Interpolation 106
7.7 Point-Normal Interpolation 107

7.8 Least Squares Approximation 108
7.9 Smoothing Equations 110
7.10 Designing with Bézier Curves 112
7.11 The Newton Form and Forward Differencing 114
7.12 Implementation 116
7.13 Problems 117

Chapter 8 B-Spline Curves 119
8.1 Motivation 120
8.2 B-Spline Segments 122
8.3 B-Spline Curves 126
8.4 Knot Insertion 130
8.5 Degree Elevation 133
8.6 Greville Abscissae 134
8.7 Smoothness 135
8.8 B-Splines 140
8.9 B-Spline Basics 143
8.10 Implementation 144
8.11 Problems 146

Chapter 9 Constructing Spline Curves 147
9.1 Greville Interpolation 147
9.2 Least Squares Approximation 149
9.3 Modifying B-Spline Curves 153
9.4 C^2 Cubic Spline Interpolation 154
9.5 More End Conditions 157
9.6 Finding a Knot Sequence 161
9.7 The Minimum Property 167
9.8 C^1 Piecewise Cubic Interpolation 169
9.9 Implementation 174
9.10 Problems 176

Chapter 10 W. Boehm: Differential Geometry I 179
10.1 Parametric Curves and Arc Length 179
10.2 The Frenet Frame 181
10.3 Moving the Frame 182
10.4 The Osculating Circle 184

10.5 Nonparametric Curves 187
10.6 Composite Curves 188

Chapter 11 **Geometric Continuity** **191**
11.1 Motivation 191
11.2 The Direct Formulation 192
11.3 The γ, ν, and β Formulations 194
11.4 G^2 Cubic Splines 195
11.5 Interpolating G^2 Cubic Splines 199
11.6 Higher-Order Geometric Continuity 200
11.7 Implementation 203
11.8 Problems 203

Chapter 12 **Conic Sections** **205**
12.1 Projective Maps of the Real Line 205
12.2 Conics as Rational Quadratics 209
12.3 A de Casteljau Algorithm 214
12.4 Derivatives 215
12.5 The Implicit Form 216
12.6 Two Classic Problems 219
12.7 Classification 220
12.8 Control Vectors 223
12.9 Implementation 224
12.10 Problems 225

Chapter 13 **Rational Bézier and B-Spline Curves** **227**
13.1 Rational Bézier Curves 227
13.2 The de Casteljau Algorithm 230
13.3 Derivatives 233
13.4 Osculatory Interpolation 234
13.5 Reparametrization and Degree Elevation 235
13.6 Control Vectors 238
13.7 Rational Cubic B-Spline Curves 238
13.8 Interpolation with Rational Cubics 240
13.9 Rational B-Splines of Arbitrary Degree 241
13.10 Implementation 242
13.11 Problems 243

Chapter 14 **Tensor Product Patches** **245**

14.1 Bilinear Interpolation 245
14.2 The Direct de Casteljau Algorithm 247
14.3 The Tensor Product Approach 250
14.4 Properties 253
14.5 Degree Elevation 254
14.6 Derivatives 255
14.7 Blossoms 258
14.8 Curves on a Surface 260
14.9 Normal Vectors 261
14.10 Twists 264
14.11 The Matrix Form of a Bézier Patch 265
14.12 Nonparametric Patches 266
14.13 Problems 267

Chapter 15 **Constructing Polynomial Patches** **269**

15.1 Ruled Surfaces 269
15.2 Coons Patches 270
15.3 Translational Surfaces 272
15.4 Tensor Product Interpolation 274
15.5 Bicubic Hermite Patches 276
15.6 Least Squares 278
15.7 Finding Parameter Values 281
15.8 Shape Equations 282
15.9 A Problem with Unstructured Data 282
15.10 Implementation 283
15.11 Problems 284

Chapter 16 **Composite Surfaces** **285**

16.1 Smoothness and Subdivision 285
16.2 Tensor Product B-Spline Surfaces 288
16.3 Twist Estimation 290
16.4 Bicubic Spline Interpolation 293
16.5 Finding Knot Sequences 295
16.6 Rational Bézier and B-Spline Surfaces 297
16.7 Surfaces of Revolution 299

16.8 Volume Deformations 301
16.9 CONS and Trimmed Surfaces 304
16.10 Implementation 306
16.11 Problems 308

Chapter 17 **Bézier Triangles** **309**
17.1 The de Casteljau Algorithm 309
17.2 Triangular Blossoms 313
17.3 Bernstein Polynomials 315
17.4 Derivatives 316
17.5 Subdivision 320
17.6 Differentiability 323
17.7 Degree Elevation 326
17.8 Nonparametric Patches 326
17.9 The Multivariate Case 328
17.10 S-Patches 330
17.11 Implementation 331
17.12 Problems 332

Chapter 18 **Practical Aspects of Bézier Triangles** **335**
18.1 Rational Bézier Triangles 335
18.2 Quadrics 337
18.3 Interpolation 341
18.4 Cubic and Quintic Interpolants 341
18.5 The Clough–Tocher Interpolant 343
18.6 The Powell–Sabin Interpolant 345
18.7 Least Squares 346
18.8 Problems 347

Chapter 19 **W. Boehm: Differential Geometry II** **349**
19.1 Parametric Surfaces and Arc Element 349
19.2 The Local Frame 352
19.3 The Curvature of a Surface Curve 352
19.4 Meusnier's Theorem 354
19.5 Lines of Curvature 355
19.6 Gaussian and Mean Curvature 357
19.7 Euler's Theorem 358

19.8 Dupin's Indicatrix 359
19.9 Asymptotic Lines and Conjugate Directions 360
19.10 Ruled Surfaces and Developables 361
19.11 Nonparametric Surfaces 363
19.12 Composite Surfaces 364

Chapter 20 **Geometric Continuity for Surfaces** **367**
20.1 Introduction 367
20.2 Triangle-Triangle 368
20.3 Rectangle-Rectangle 372
20.4 Rectangle-Triangle 373
20.5 "Filling in" Rectangular Patches 374
20.6 "Filling in" Triangular Patches 375
20.7 Theoretical Aspects 375
20.8 Problems 376

Chapter 21 **Surfaces with Arbitrary Topology** **377**
21.1 Recursive Subdivision Curves 377
21.2 Doo–Sabin Surfaces 380
21.3 Catmull–Clark Subdivision 383
21.4 Midpoint Subdivision 386
21.5 Loop Subdivision 387
21.6 $\sqrt{3}$ Subdivision 389
21.7 Interpolating Subdivision Surfaces 389
21.8 Surface Splines 392
21.9 Triangular Meshes 394
21.10 Decimation 395
21.11 Problems 398

Chapter 22 **Coons Patches** **399**
22.1 Coons Patches: Bilinearly Blended 400
22.2 Coons Patches: Partially Bicubically Blended 402
22.3 Coons Patches: Bicubically Blended 404
22.4 Piecewise Coons Surfaces 406
22.5 Two Properties 407
22.6 Compatibility 408
22.7 Gordon Surfaces 410

22.8 Boolean Sums 412
22.9 Triangular Coons Patches 414
22.10 Problems 417

Chapter 23 **Shape** **419**
23.1 Use of Curvature Plots 419
23.2 Curve and Surface Smoothing 421
23.3 Surface Interrogation 425
23.4 Implementation 427
23.5 Problems 429

Chapter 24 **Evaluation of Some Methods** **431**
24.1 Bézier Curves or B-Spline Curves? 431
24.2 Spline Curves or B-Spline Curves? 431
24.3 The Monomial or the Bézier Form? 432
24.4 The B-Spline or the Hermite Form? 435
24.5 Triangular or Rectangular Patches? 435

Appendix A **Quick Reference of Curve and Surface Terms** **437**

Appendix B **List of Programs** **445**

Appendix C **Notation** **447**

References **449**

Index **491**

Preface

Computer aided geometric design (CAGD) is a discipline dealing with computational aspects of geometric objects. It is best explained by a brief historical sketch.[1]

Renaissance naval architects in Italy were the first to use drafting techniques that involved conic sections. Prior to that, ships were built "hands on" without any mathematics being involved. These design techniques were refined through the centuries, culminating in the use of *splines*—wooden beams that were bent into optimal shapes. In the beginning of the twentieth century, airplanes made their first appearance. Their design (or rather, the design of the outside fuselage) was streamlined by the use of conic sections, as pioneered by R. Liming [390]. He devised methods that went beyond traditional drafting with conics—for the first time, certain conic coefficients could be used to define a shape—thus numbers could be used to replace blueprints!

The automobile, one of the defining cultural icons of the twentieth century, also needed new design approaches as mass production started. In the late 1950s, hardware became available that allowed the machining of 3D shapes out of blocks of wood or steel.[2] 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 in

[1] For more details, see [204].

[2] A process that is now called CAM for *computer aided manufacturing*.

Color Plates I and III: Color Plate I shows the actual hood of a car; Color Plate III shows how it is represented internally as a collection of parametric surfaces.

The major breakthroughs in CAGD were the theory of Bézier curves and surfaces, later combined with B-spline methods. Bézier curves and surfaces were independently developed by P. de Casteljau at Citroën and by P. Bézier at Rénault. 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 [34]).

Several other disciplines have emerged and interacted with CAGD. *Computational geometry* is concerned with the analysis of geometric algorithms. An example would be finding a bound on the time it takes to triangulate a set of points. Knowledge of such bounds allows a comparison and evaluation of different algorithms. The literature includes Prepata and Shamos [497] and de Berg et al. [135]. Ironically, another book with the term computational geometry in it is the one by Faux and Pratt [228]. It was a very influential text, but today, it would be classified as a CAGD text.

Another related discipline is *solid modeling*. It 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. The literature includes Hoffmann [327] and Mantylä [416]. CAGD has also influenced fields such as medical imaging, geographic information systems, computer gaming, and scientific visualization. It should go without saying that computer graphics is one of the earliest and most important applications of CAGD; see [238] or [9].

For this fifth edition, the most notable addition is a chapter on subdivision surfaces that were of academic interest at best when the first edition appeared in 1988. A recent special issue of the journal *Computer Aided Geometric Design* highlights some of the new developments; see [393], [432], [470], [579], [598], and [631]. Other new topics include triangle meshes, more in-depth treatment of least squares techniques, and pervasive use of the blossoming principle.

Each chapter is concluded by a set of problems. They come in three categories: simpler exercises at the beginning of each Problem section, harder problems marked by asterisks, and programming problems marked by "P." Many of these programming problems use data on the Web site. 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.

The C programs on the Web site 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.

The Web page for the book is *www.mkp.com/cagd5e*. This page includes C programs, data sets, and errata.

As for all previous editions, sincere thanks go to Dianne Hansford for help and advice with all aspects of the book.

This Page Intentionally Left Blank

1

P. Bézier

How a Simple System Was Born

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

Around 1960, designers of stamped parts, that is, 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 resulted in discussions, arguments, haggling, retouches, expenses, and delay.

Obviously, no significant improvement could be expected so long as a method was not devised that could prove an accurate, complete, and undisputable definition of freeform shapes.

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

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

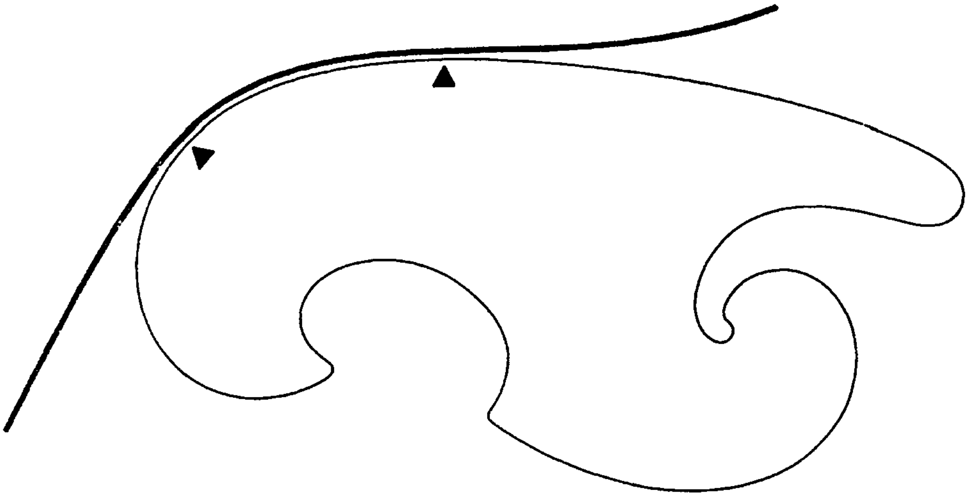


Figure 1.1 An arc of a hand-drawn curve is approximated by a part of a template.

In France, at that time, very little was known about the work performed in the American aircraft industry; the papers from James Ferguson were not much displayed before 1964; Citroen 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 before 1967; The works of W. Gordon and R. Riesenfeld were printed in 1974.

At the beginning, the idea 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 (A and B), together with their curvilinear abscissae, related with 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 neglecting the effect of friction on the tracing cloth. However, it is difficult to take into account the length between endpoints; moreover, the curves employed for software for NC machine tools, that is, 2D milling machines, were lines and circles, and sometimes, parabolas. Hence, a spline shape should be divided and subdivided into small arcs of circles put end to end.

To transform an arc of circle into a portion of an ellipse, imagine (Figure 1.2) a square frame containing two sets of strings, whose intersections would be located on an arc of a circle; the frame sides being hinged, the square is transformed into a diamond (Figure 1.3), and the circle becomes an arc of an ellipse, which would

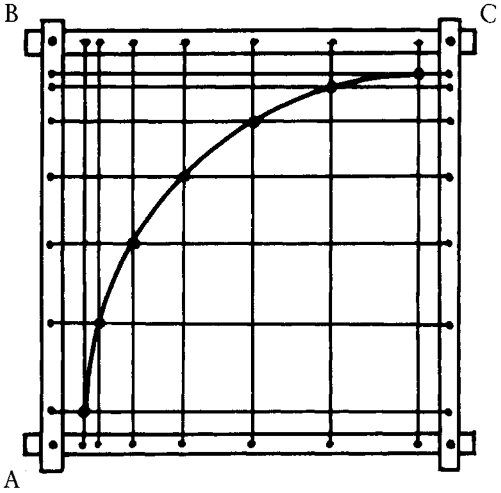


Figure 1.2 A circular arc is obtained by connecting the points in this rectangular grid.

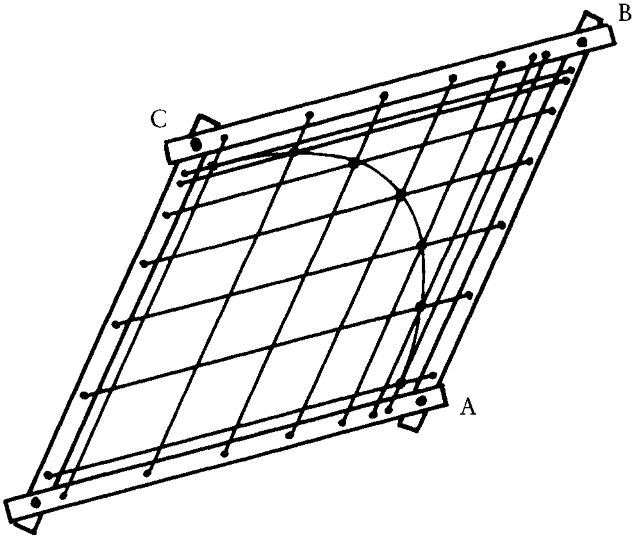


Figure 1.3 If the frame from the previous figure is sheared, an arc of an ellipse is obtained.

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 definition of the arc of ellipse still results from the coordinates of the three points A, B, and C (Figure 1.5).

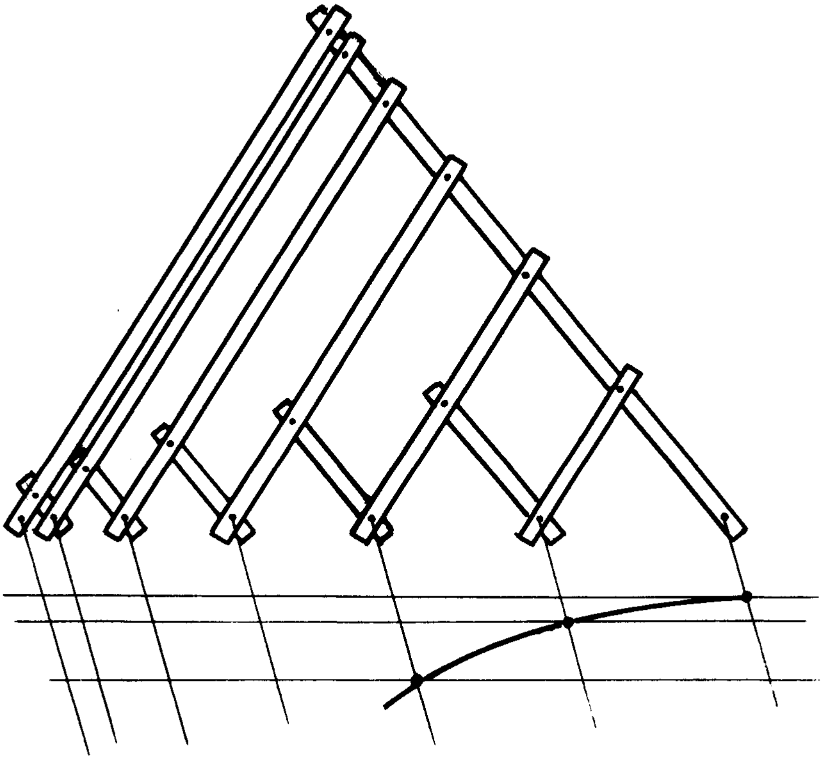


Figure 1.4 Pantograph construction of an arc of an ellipse.

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.

But 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 sign, 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 flashlight (Figure 1.6) would define the image of the curve projected on the wall of the drawing office. One could even imagine having a

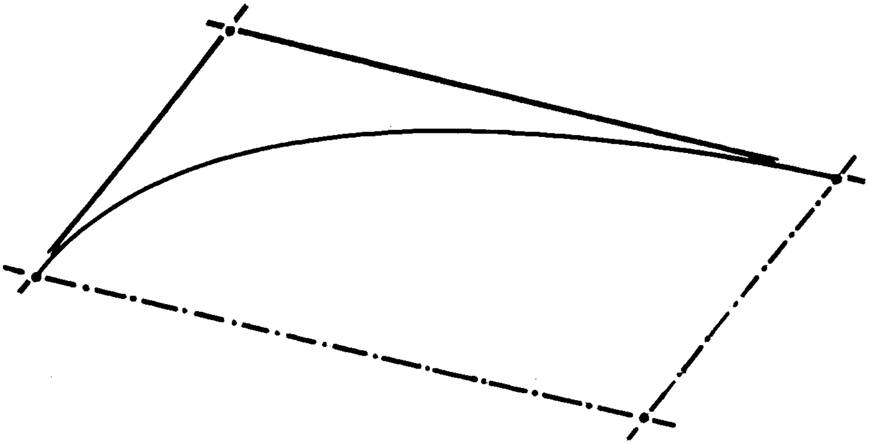


Figure 1.5 A "control polygon" for an arc of an ellipse.

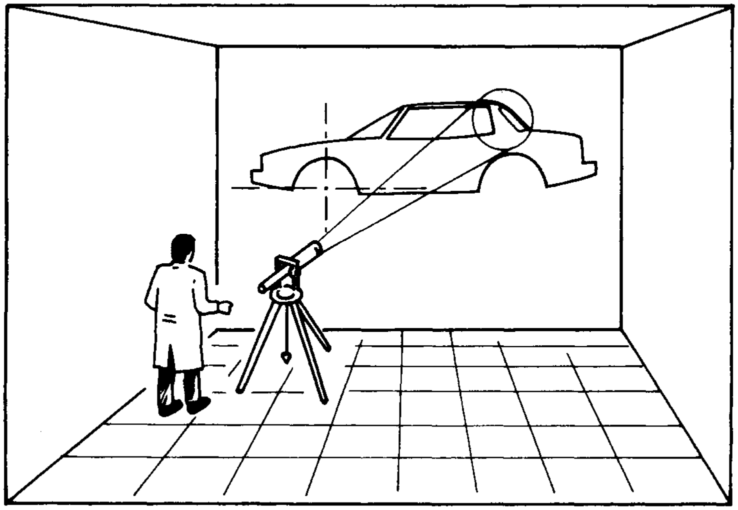


Figure 1.6 A projector producing a "template curve" on the drawing of an object.

variety of slides, each of which would bear a specific curve: circle, parabola, astroid, and so on.

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.

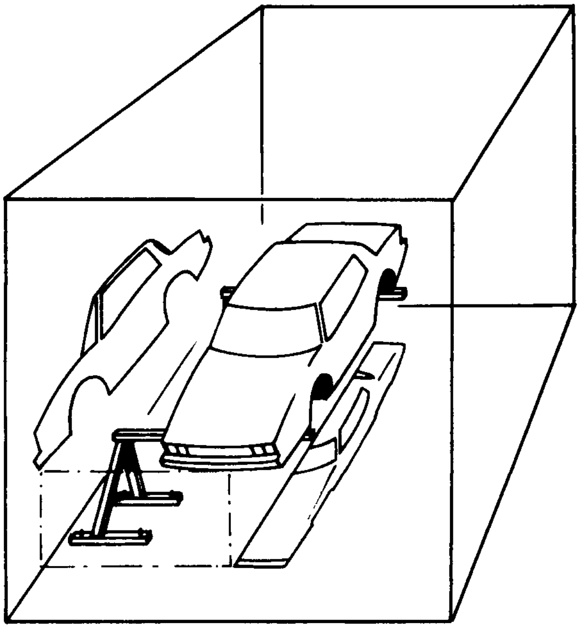


Figure 1.7 Two imaginary projections of a car.

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 with so-called character lines, which seldom are plane curves. Hence, a good system should be able to manipulate and define directly "space curves" or "freeform curves." 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 as 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 a curve inscribed in a square, it 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 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 defines directly the endpoint B and its

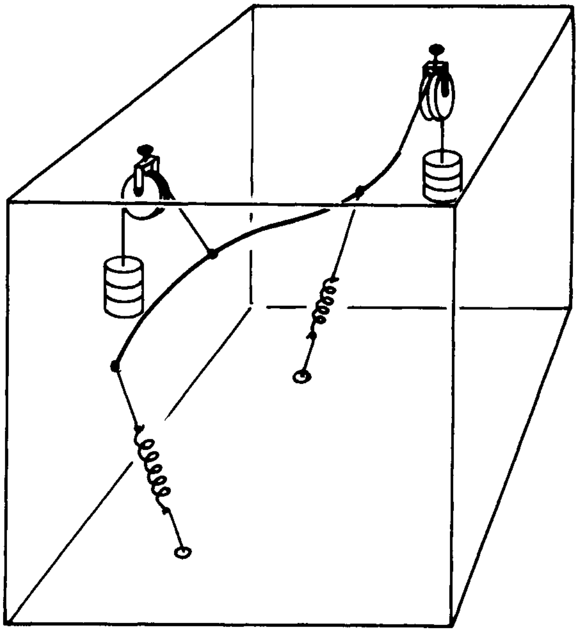


Figure 1.8 A curve held by springs.

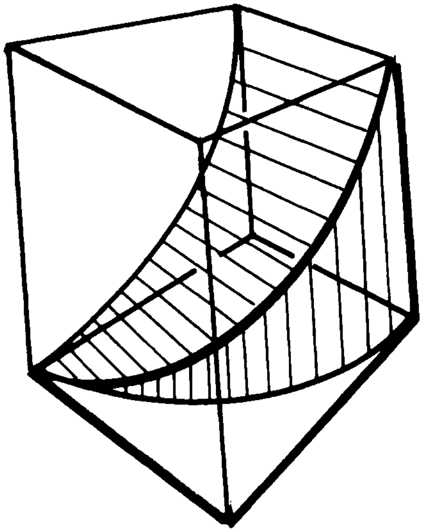


Figure 1.9 A curve defined inside a cube.

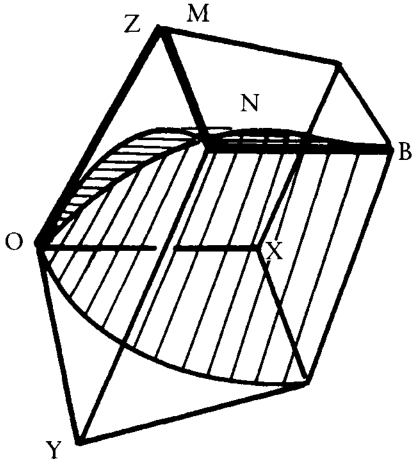


Figure 1.10 A curve defined inside a parallelepiped.

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 a variety of curves, but in order to increase it, we can imagine making use of cubes and hypercubes of any order (Figure 1.11) and the relevant polygons (Figure 1.13 and see Figure 4.4 in Section 4.3).

At that moment, it became necessary to do away with harmonic functions and revert to polynomials. This was even more desirable since digital computers were gradually replacing analog computers. The polynomial functions were chosen according to the properties that were considered best: tangency, curvature, and the like. Later, it was discovered that they could be regarded 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 should be translated in secret listings, and buried in the most secret part of the memory of the computer, and that nobody but they would keep the key of 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 with the shape of the corresponding curves.

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

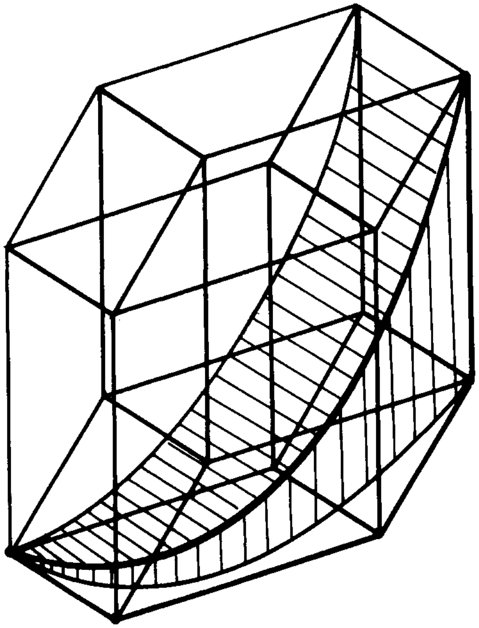


Figure 1.11 Higher-order curves can be defined inside higher-dimensional cubes.

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 since 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 at the same time moved and distorted. About 1970, a Dutch laboratory sculptured 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 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, 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

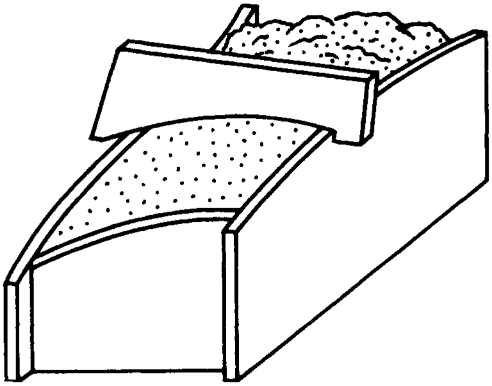


Figure 1.12 A surface is being obtained by scraping off excess material with wooden templates.

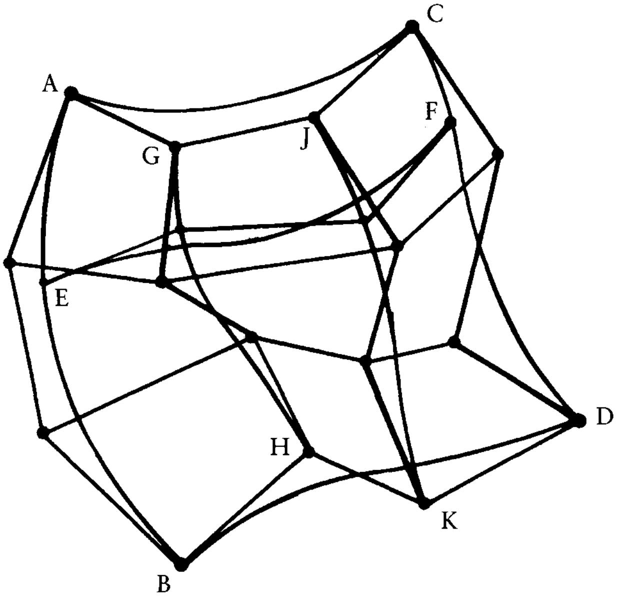


Figure 1.13 The characteristic net of a surface.

the patch, which plays, regarding the surface, the same part as a polygon of a curve. Hence, the cartesian coordinates of the points of the patch are computed according to the values of two parameters.

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

So, a system has been progressively created. If we consider the way an initial idea evolved, we observe that the first solution—parallelogram, pantograph—is the result of an education oriented toward kinematics, the conception of mechanisms. Next appeared geometry and optics, which very likely came from some training in the army, when geometry, cosmography, and topography played an important part. Then reflexion was oriented toward analysis, parametric spaces, and finally, data processing, because a theory, as convenient as it may look, must not impose too heavy a task to the computer and must be easily understood, at least in its principle, by the operators.

The various steps of this conception have a point in common: each idea must be related with the principle on a material system, however simple and primitive it may look, on which a variable solution could be based.

Engineers define what is to be done and how it could be done; they not only describe the goal, they lead the way toward it.

Before looking any deeper into this subject, it should be observed that elementary geometry played a major part, and it should not gradually disappear from the courses 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 equations (5.6) and (14.6) in the subsequent chapters. Evidently, these formulas, conveniently arranged, are best suited to express data given to a computer, but most people would better understand a simple figure than the equivalent algebraic expression.

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

Which is the part played by experience, by theory, and by imagination in the creation of a system? There is no definite answer to such a query. The importance of experience and of theoretical knowledge is not always clearly perceived. Imagination seems a gift, a godsend, or the result of a beneficial heredity; but is not, in fact, imagination the result of the maturation of the 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 based, partly, on the ability to connect notions that, at first sight, look quite unrelated, such as mechanics, electronics, optics, foundry, data processing, to catch barely seen analogies, as Alice in Wonderland, to go "through the mirror"?

Will, someday, psychologists be able to detect in man such a gift that would be applicable to science and technology? Has it a relation with the sense of humor that can detect unexpected relations between facts that look quite unconnected? Shall we learn how to develop it? Will it forever remain a gift, devoted by pure chance to some people, whereas for others carefulness and cold blood prevail?

It is important that, sometimes, "sensible" men 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."

Introductory Material 2

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]

We stress the concept of coordinate-free methods throughout this book. It motivates the strict distinction between points and vectors as discussed next. (For more details on this topic, see R. Goldman [262].)

[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.

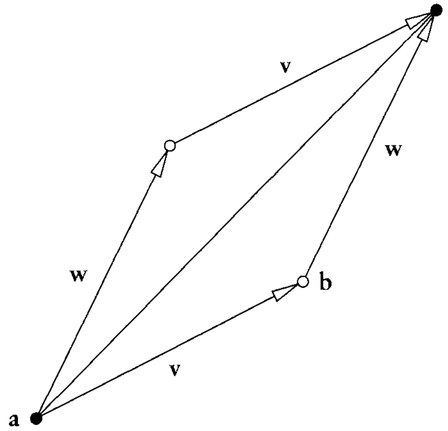


Figure 2.1 Points and vectors: vectors are not affected by translations.

We shall denote *points*, elements of three-dimensional (or 3D) euclidean (or point) space $\mathbb{E}^3$, by lowercase boldface letters such as **a**, **b**, and so on. (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.

The same notation (lowercase boldface) will be used for *vectors*, elements of 3D 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 **a** and **b**, there is a unique vector **v** that points from **a** to **b**. It is computed by componentwise subtraction:

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

On the other hand, given a vector **v**, there are infinitely many pairs of points **a**, **b** such that $\mathbf{v} = \mathbf{b} - \mathbf{a}$. For if **a**, **b** is one such pair and if **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 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.

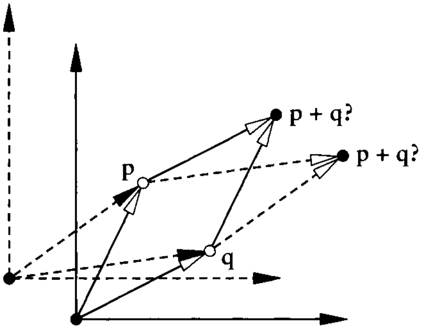


Figure 2.2 Addition of points: this is not a well-defined operation, since different coordinate systems would produce different "solutions."

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

$$\mathbf{b} = \sum_{j=0}^{n} \alpha_j \mathbf{b}_j; \quad \mathbf{b}_j \in \mathbb{E}^3, \alpha_0 + \cdots + \alpha_n = 1. \tag{2.1}$$

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 [130], p. 214.) Since a common factor in the m_j is immaterial for

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

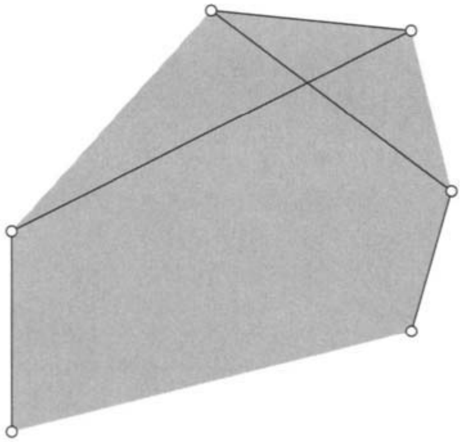


Figure 2.3 Convex hulls: a point set (a polygon) and its convex hull, shown shaded.

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 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 Problems). 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:

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

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 sum are vectors. Whereas 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 [160], [161]. R. Goldman's article [262] 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 12.) The term *affine map* is due to L. Euler; affine maps were first studied systematically by F. Moebius [429].

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

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

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 it has a simple interpretation. The expression $\mathbf{x} = \sum \alpha_j \mathbf{a}_j$ specifies how we have to weight the points $\mathbf{a}_j$ such 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, that is, that barycentric combinations are preserved by maps of that form. For the following, recall that $\sum \alpha_j = 1$:

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

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

Some examples of affine maps are as follows:

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

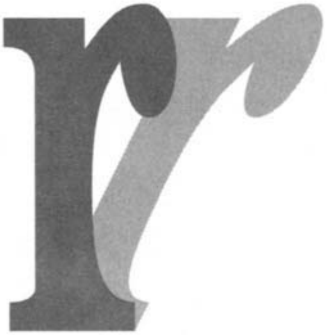


Figure 2.4 A shear: this affine map is used in font design in order to generate slanted fonts. Dark gray: original letter; light gray: slanted (sheared) letter.

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

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 while "tilting" the z-axis.

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.

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.[3]

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

[3] See also Figure 5.11.

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 3D objects to 3D objects. If the rank is less than three, Φ is a parallel projection onto a plane ($\mathrm{rank} = 2$) or even onto a straight line ($\mathrm{rank} = 1$).

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.

We may also define affine maps of vectors. If $\mathbf{w} = \mathbf{b} - \mathbf{a}$ is a vector, and $A\mathbf{x} + \mathbf{v}$ represents an affine map Φ, then

$$\Phi(\mathbf{w}) = A\mathbf{w}$$

is the image of $\mathbf{w}$ under Φ. As expected, the translational part $\mathbf{v}$ of the affine map is of no consequence when mapping vectors to vectors.

2.3 Constructing Affine Maps

Suppose we are given a 2D point set $\mathbf{p}_1, \ldots, \mathbf{p}_L$ whose centroid is located at the origin. Before discussing affine maps of these points, we first study a unique ellipse that is associated with this point set; it is called the *norm ellipse*, see [90], [155], [449], [448], [510].

Our derivation of this ellipse is as follows: an ellipse with center at the origin is given by a quadratic from

$$\mathbf{x}^{\mathrm{T}} A\mathbf{x} = 1 \tag{2.4}$$

where A is a symmetric matrix with two nonnegative eigenvalues.

Our goal is to find a symmetric matrix A that captures some of the characteristics of the given point set.

Each $\mathbf{p}_i$ is of the form

$$\mathbf{p}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

If it were on an ellipse defined by A, then all points would satisfy

$$\mathbf{p}_i^{\mathrm{T}} A\mathbf{p}_i = 1; \quad i = 1, \ldots, L. \tag{2.5}$$

We define

$$\mathbf{P} = [\,\mathbf{p}_1 \quad \dots \quad \mathbf{p}_L\,],$$

a matrix with two rows and L columns. Equation (2.5) can now be written in terms of one matrix equation:

$$\mathbf{P}^{\mathrm{T}} A \mathbf{P} = I.$$

We now multiply with $\mathbf{P}$ from the left and with $\mathbf{P}^{\mathrm{T}}$ from the right to obtain

$$\mathbf{P}\mathbf{P}^{\mathrm{T}} A \mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{P}\mathbf{P}^{\mathrm{T}}.$$

We define

$$B = \mathbf{P}\mathbf{P}^{\mathrm{T}},$$

a 2 × 2 matrix. Assuming it is invertible (which it is for a nondegenerate set of points $\mathbf{p}_i$), we find that

$$A = B^{-1}$$

is the desired matrix for our quadratic form (2.4). It is related to the points $\mathbf{p}_i$ in an affinely invariant way: subject the data to an affine map and recompute the norm ellipse. It is the same ellipse as is obtained by mapping the original norm ellipse by the affine map.

The matrix A is symmetric by construction; the fact that it has nonnegative eigenvalues (i.e., it represents an ellipse) follows from its definition (2.4).

The axes of the ellipse defined by A represent the distribution of the points $\mathbf{p}_i$; Figure 2.5 gives an example. The axes are given by the eigenvectors of A; their lengths are determined by the corresponding eigenvalues.

Returning to the topic of affine maps, suppose we are given a point set $\mathbf{p}_1, \dots, \mathbf{p}_L$ and a second set $\mathbf{q}_1, \dots, \mathbf{q}_L$, both with their centroids at the origin. If there is an affine map, represented by a matrix M with

$$\mathbf{q}_i = M\mathbf{p}_i,$$

how can we find it?

A simple and efficient way is to compute the two norm ellipses E_p and E_q for the two point sets. Since any two ellipses are related by an affine map Φ_{pq}, we simply compute it; then Φ_{pq} is the desired affine map.

In general, the two point sets are not related by an affine map; this procedure will still produce an affine map that approximately maps the two point sets.

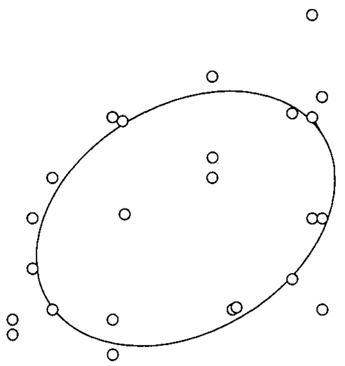


Figure 2.5 Norm ellipses: a point set and an associated ellipse.

This method works even if the number of $\mathbf{q}_i$ does not equal that of the $\mathbf{p}_i$. An obvious generalization works in 3D. Applications are in image registration, where two images (typically represented by point sets) have to be mapped to each other.

2.4 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 [133] and de Boor [138].

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 $c_1 = \ldots = c_n = 0$.

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 2.6 gives an example.

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

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

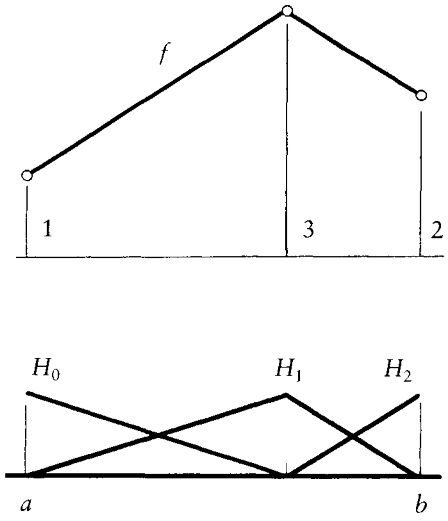


Figure 2.6 Hat functions: the piecewise linear function f can be written as $f = H_0 + 3H_1 + 2H_2$.

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

2.5 Problems

1 Of all affine maps, shears seem to be the least familiar to most people.[4] 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 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.

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

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

[4] Recall that Figure 2.4 illustrates a shear.

3 Linear Interpolation

Most of the computations that we use in CAGD may be broken down into seemingly trivial steps—sequences of linear interpolations. It is therefore important to understand the properties of these basic building blocks. This chapter explores those properties and introduces a related concept, called blossoms.

3.1 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

$$\mathbf{x} = \mathbf{x}(t) = (1-t)\mathbf{a} + t\mathbf{b}; \quad t \in \mathbb{R} \tag{3.1}$$

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

For $t = 0$, the straight line passes through $\mathbf{a}$ and for $t = 1$, it passes through $\mathbf{b}$. For $0 \le t \le 1$, the point $\mathbf{x}$ is between $\mathbf{a}$ and $\mathbf{b}$, whereas for all other values of t it is outside; see Figure 3.1.

Equation (3.1) represents $\mathbf{x}$ as a barycentric combination of two points in $\mathbb{E}^3$. The same barycentric combination holds for the three points $0, t, 1$ in $\mathbb{E}^1$: $t = (1-t) \cdot 0 + t \cdot 1$. So t is related to 0 and 1 by the same barycentric combination that relates $\mathbf{x}$ to $\mathbf{a}$ and $\mathbf{b}$. Hence, by the definition of affine maps, the three points $\mathbf{a}, \mathbf{x}, \mathbf{b}$ are an affine map of the three 1D points $0, t, 1$! *Thus linear interpolation is an affine map of the real line onto a straight line in* $\mathbb{E}^3$.[1]

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

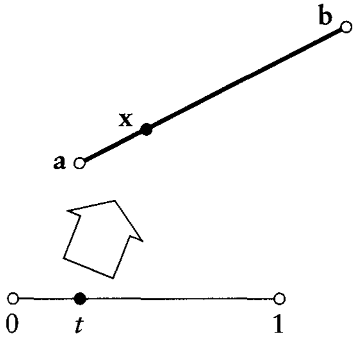


Figure 3.1 Linear interpolation: two points $\mathbf{a}, \mathbf{b}$ define a straight line through them. The point t in the domain is mapped to the point $\mathbf{x}$ in the range.

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 (3.1) holds, then also

$$\Phi\mathbf{x} = \Phi\big((1-t)\mathbf{a} + t\mathbf{b}\big) = (1-t)\Phi\mathbf{a} + t\Phi\mathbf{b}. \tag{3.2}$$

Since affine maps may be applied to vectors as well as to points, it makes sense to ask what linear interpolation will do to vector arguments. These vectors "live" in 1D domain space, and will be denoted by $\vec{v}$.

If c and d are two 1D points in the domain, they define a vector $\vec{v}$ by setting $\vec{v} = d - c$. The corresponding vector $\mathbf{l}(\vec{v})$ in the range is then defined as

$$\mathbf{l}(\vec{v}) = \mathbf{l}(d) - \mathbf{l}(c). \tag{3.3}$$

Figure 3.2 illustrates. For the special case of $\vec{v}$ being the 1D zero vector $\vec{v} = \vec{0}$, we have

$$\mathbf{l}(\vec{0}) = \mathbf{0}.^2 \tag{3.4}$$

Closely related to linear interpolation is the concept of *barycentric coordinates*, due to Moebius [429]. Let $\mathbf{a}, \mathbf{x}, \mathbf{b}$ be three collinear points in $\mathbb{E}^3$:

$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b}; \quad \alpha + \beta = 1. \tag{3.5}$$

[2] Here, 0 denotes the zero vector.

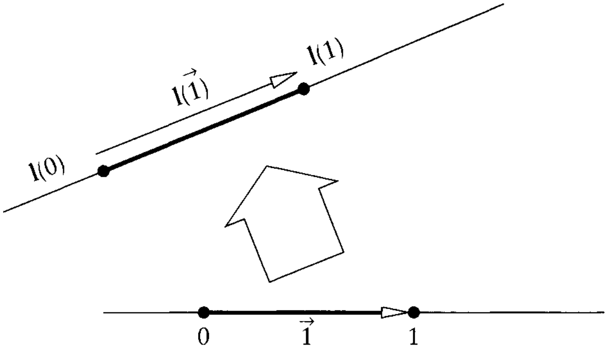


Figure 3.2 Linear interpolation: the vector $\vec{1}$ in the domain is mapped to the vector $\mathbf{l}(\vec{1})$ in the range.

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

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

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

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

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 3.5 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

$$\mathrm{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{\mathrm{vol}_1(\mathbf{a}, \mathbf{b})}{\mathrm{vol}_1(\mathbf{b}, \mathbf{c})}. \tag{3.6}$$

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

$$\mathrm{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{\beta}{\alpha}. \tag{3.7}$$

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 (3.7) holds, then also

$$\text{ratio}(\Phi\mathbf{a}, \Phi\mathbf{b}, \Phi\mathbf{c}) = \frac{\beta}{\alpha}, \tag{3.8}$$

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 **a**, **x**, and **b**, we would project them onto one of the coordinate axes and then use simple differences of their x- or y-coordinates.[3] This shortcut works since parallel projection is an affine map!

Equation (3.8) 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 $[a, b]$. 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

$$\mathbf{x}(u) = \frac{b - u}{b - a}\mathbf{a} + \frac{u - a}{b - a}\mathbf{b}. \tag{3.9}$$

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 more general way to express this is by saying that any barycentric combination of three domain points r, s, t (not necessarily involving any interval endpoints) carries over to the corresponding range points:

$$s = (1 - \alpha)r + \alpha t \Rightarrow \mathbf{x}(s) = (1 - \alpha)\mathbf{x}(r) + \alpha\mathbf{x}(t). \tag{3.10}$$

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

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.

3.2 Piecewise Linear Interpolation

Let $\mathbf{b}_0, \ldots, \mathbf{b}_n \in \mathbb{E}^3$ form a *polygon* **B**. This 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 **c**, then **B** is said to be a piecewise linear interpolant to **c**, and we write

$$\mathbf{B} = \mathcal{PL}\,\mathbf{c}. \tag{3.11}$$

One of the important properties of piecewise linear interpolation is *affine invariance*. If the curve **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:

$$\mathcal{PL}\,\Phi\mathbf{c} = \Phi\ \mathcal{PL}\,\mathbf{c}. \tag{3.12}$$

Another property is the *variation diminishing property*. Consider a continuous curve **c**, a piecewise linear interpolant $\mathcal{PL}\,\mathbf{c}$, and an arbitrary plane. Let cross **c** be the number of crossings that the curve **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 3.8.) Then we always have

$$\mathrm{cross}(\mathcal{PL}\,\mathbf{c}) \le \mathrm{cross}\,\mathbf{c}. \tag{3.13}$$

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, whereas the curve segment from **c** that connects them may cross the same plane in many arbitrary points. The variation diminishing property is illustrated in Figure 3.3.

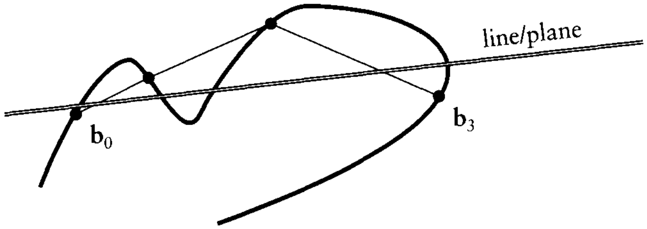


Figure 3.3 The variation diminishing property: a piecewise linear interpolant to a curve has no more intersections with any plane than the curve itself.

3.3 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 [146] and W. Boehm [67].

Referring to Figure 3.4, we define

$$\begin{aligned}\mathbf{b}[0,t] &= (1-t)\mathbf{b}_0 + t\mathbf{b}_1,\\ \mathbf{b}[s,0] &= (1-s)\mathbf{b}_0 + s\mathbf{b}_1,\\ \mathbf{b}[1,t] &= (1-t)\mathbf{b}_1 + t\mathbf{b}_2,\\ \mathbf{b}[s,1] &= (1-s)\mathbf{b}_1 + s\mathbf{b}_2.\end{aligned}$$

Let us further define two points

$$\begin{aligned}\mathbf{b}[s,t] &= (1-t)\mathbf{b}[s,0] + t\mathbf{b}[s,1] \quad \text{and}\\ \mathbf{b}[t,s] &= (1-s)\mathbf{b}[0,t] + s\mathbf{b}[t,1].\end{aligned} \tag{3.14}$$

Menelaos' theorem now states that these points are identical:

$$\mathbf{b}[s,t] = \mathbf{b}[t,s]. \tag{3.15}$$

For a proof, we simply verify that

$$\mathbf{b}[s,t] = \mathbf{b}[t,s] = (1-t)(1-s)\mathbf{b}_0 + [(1-t)s + t(1-s)]\mathbf{b}_1 + st\mathbf{b}_2. \tag{3.16}$$

Some interesting special cases are given by $\mathbf{b}[0,0] = \mathbf{b}_0$ or by $\mathbf{b}[0,1] = \mathbf{b}_1$.

Equation (3.15) is a "CAGD version" of the original Menelaos' theorem, which may be stated as (see Coxeter [130]):

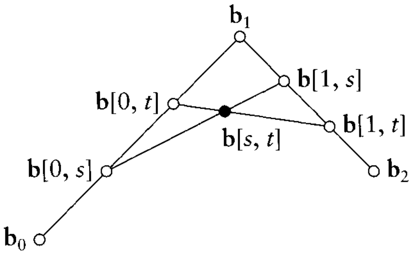


Figure 3.4 Menelaos' theorem: the point $\mathbf{b}[s, t]$ may be obtained from linear interpolation at t or at s.

$$\text{ratio}(\mathbf{b}[s,1], \mathbf{b}[1,t], \mathbf{b}_1) \cdot \text{ratio}(\mathbf{b}_1, \mathbf{b}[0,t], \mathbf{b}[s,0]) \cdot$$
$$\text{ratio}(\mathbf{b}[s,0], \mathbf{b}[s,t], \mathbf{b}[s,1]) = -1. \tag{3.17}$$

The proof of (3.17) is a direct consequence of (3.15). Note the ordering of points in the second ratio! Menelaos' theorem is closely related to Ceva's, which is given in Section 3.5.

3.4 Blossoms

The bivariate function $\mathbf{b}[t_1, t_2]$ from (3.16) will be very important for the remainder of this book. Functions of that type are called *blossoms*. Before we introduce the general concept, we will further explore properties of (3.16).

The first property is called *symmetry*. It states that the order of the blossom arguments does not matter—which is exactly Menelaos' theorem.

In Section 3.1, we saw that linear interpolation carries domain relationships over to corresponding range relationships; see (3.10). Since blossoms are evaluated using linear interpolations, we now have: if the first argument t_1 of a blossom is a barycentric combination of two (or more) 1D points r and s, we may compute the blossom values for each argument and then form their barycentric combination:

$$\mathbf{b}[\alpha r + \beta s, t_2] = \alpha \mathbf{b}[r, t_2] + \beta \mathbf{b}[s, t_2]; \quad \alpha + \beta = 1. \tag{3.18}$$

Equation (3.18) states that the blossom $\mathbf{b}$ is affine with respect to its first argument, but it is affine for the second one as well because of the symmetry property. This is the reason the blossom is called *multiaffine*—the second of its main properties.

For a third property, we study what happens if both blossom arguments are equal: $t_1 = t_2 = t$. Then the expression $\mathbf{b}[t, t]$ denotes a point that depends on one variable t—thus it traces out a polynomial curve.[4] This property is called the *diagonal property.*

Our special blossom $\mathbf{b}[t_1, t_2]$ has two arguments. Blossoms with an arbitrary number n of arguments are easily defined by the preceding three properties. A blossom is an n-variate function $\mathbf{b}[t_1, \ldots, t_n]$ from $\mathbb{R}^n$ into $\mathbb{E}^2$ or $\mathbb{E}^3$. It is defined by three properties:

Symmetry:

$$\mathbf{b}[t_1, \ldots, t_n] = \mathbf{b}[\pi(t_1, \ldots, t_n)] \tag{3.19}$$

where $\pi(t_1, \ldots, t_n)$ denotes a permutation of the arguments $t_1, \ldots, t_n$. Thus, for example $\mathbf{b}[t_1, t_2, t_3] = \mathbf{b}[t_2, t_3, t_1]$.

Multiaffinity:

$$\mathbf{b}[(\alpha r + \beta s), *] = \alpha\mathbf{b}[r, *] + \beta\mathbf{b}[s, *]; \;\; \alpha + \beta = 1. \tag{3.20}$$

Here, the symbol $*$ indicates that there are the same arguments on both sides of the equation, but their exact meaning is not of interest. Because of symmetry, this property holds for all arguments, not just the first one.

Diagonality:
If all arguments of the blossom are the same: $t = t_1, \ldots, t_n$, then we obtain a polynomial curve (to be discussed later). We will use the notation

$$\mathbf{b}[t, \ldots, t] = \mathbf{b}[t^{<n>}]$$

if the argument t is repeated n times.

We defined vector arguments for linear interpolation in Section 3.1. Blossoms may also have vector arguments, resulting in expressions such as $\mathbf{b}[\vec{h}, r, s]$. If we assume (without loss of generality) that the first argument of a blossom is a vector $\vec{h} = b - a$, then the multiaffine property becomes

$$\mathbf{b}[b - a, *] = \mathbf{b}[b, *] - \mathbf{b}[a, *]. \tag{3.21}$$

Thus if (at least) one of the blossom arguments is a vector, then the blossom value is a vector. For example, if we denote by $\vec{1}$ the 1D unit vector, then $\mathbf{b}[\vec{1}, r, s] = \mathbf{b}[1, r, s] - \mathbf{b}[0, r, s]$ or $\mathbf{b}[\vec{1}, r, s] = \mathbf{b}[3, r, s] - \mathbf{b}[2, r, s]$.

[4] This kind of curve will later be called a Bézier curve.

As an application of the blossom properties, let us derive a formula that will be used later. We consider the special case when a blossom argument is of the form $(\alpha r + \beta s)^{<n>}$. For this, we get

$$\mathbf{b}[(\alpha r + \beta s)^{<n>}] = \sum_{i=0}^{n} \binom{n}{i} \alpha^i \beta^{n-i} \mathbf{b}[r^{<i>}, s^{<n-i>}]. \tag{3.22}$$

We refer to this equation as the *Leibniz formula*.[5]

The proof is by induction. The case $n = 1$ is a trivial start. The inductive step proceeds as follows (keeping in mind that $\binom{n}{n+1} = \binom{n}{-1} = 0$):

$$\begin{aligned}\mathbf{b}[(\alpha r + \beta s)^{<n+1>}] &= \sum_{i=0}^{n} \binom{n}{i} \alpha^i \beta^{n-i} \mathbf{b}[(\alpha r + \beta s), r^{<i>}, s^{<n-i>}] \\ &= \sum_{i=0}^{n} \binom{n}{i} \alpha^{i+1} \beta^{n-i} \mathbf{b}[r^{<i+1>}, s^{<n-i>}] \\ &+ \sum_{i=0}^{n} \binom{n}{i} \alpha^i \beta^{n+1-i} \mathbf{b}[(r^{<i>}, s^{<n+1-i>}]\end{aligned}$$

Now we transform the index of the first sum and let the second sum run to $n + 1$:

$$\begin{aligned}&\sum_{i=1}^{n+1} \binom{n}{i-1} \alpha^i \beta^{n+1-i} \mathbf{b}[r^{<i>}, s^{<n+1-i>}] \\ &+ \sum_{i=0}^{n+1} \binom{n}{i} \alpha^i \beta^{n+1-i} \mathbf{b}[r^{<i>}, s^{<n+1-i>}]\end{aligned}$$

The first sum may start with $i = 0$. Keeping in mind the recursion

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

we can combine the last two sums and get

$$\mathbf{b}[(\alpha r + \beta s)^{<n+1>}] = \sum_{i=0}^{n+1} \binom{n+1}{i} \alpha^i \beta^{n+1-i} \mathbf{b}[r^{<i>}, s^{<n+1-i>}],$$

which concludes our proof. This result will be used several times later on.

[5] It has the structure of Leibniz's rule for higher-order derivatives of a product of functions.

A different form of (3.22) is sometimes useful:

$$\mathbf{b}[(\alpha r+\beta s)^{<n>}]=\sum_{\substack{i+j=n\\ i,j\geq 0}}\binom{n}{i,j}\alpha^i\beta^j\mathbf{b}[r^{<i>},s^{<j>}] \tag{3.23}$$

where

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

3.5 Barycentric Coordinates in the Plane

Barycentric coordinates were discussed in Section 3.1, 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 introduced by F. Moebius in 1827; see his collected works [429].

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**:

$$\mathbf{p}=u\mathbf{a}+v\mathbf{b}+w\mathbf{c}. \tag{3.24}$$

A reminder: if (3.24) is to be a barycentric combination (and hence geometrically meaningful), we require that

$$u+v+w=1. \tag{3.25}$$

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

If the four points **a**, **b**, **c**, and **p** are given, we can always determine **p**'s barycentric coordinates u, v, w: Equations (3.24) and (3.25) can be viewed as a linear system of three equations[6] in three unknowns u, v, w. The solution is obtained by an application of Cramer's rule:

$$u=\frac{\text{area}(\mathbf{p},\mathbf{b},\mathbf{c})}{\text{area}(\mathbf{a},\mathbf{b},\mathbf{c})},\quad v=\frac{\text{area}(\mathbf{a},\mathbf{p},\mathbf{c})}{\text{area}(\mathbf{a},\mathbf{b},\mathbf{c})},\quad w=\frac{\text{area}(\mathbf{a},\mathbf{b},\mathbf{p})}{\text{area}(\mathbf{a},\mathbf{b},\mathbf{c})}. \tag{3.26}$$

[6] Recall that (3.24) is shorthand for two scalar equations.

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

$$\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} \begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{vmatrix} . \tag{3.27}$$

We note that in order for (3.26) to be well defined, we require area(**a**, **b**, **c**) $\neq 0$, which means that **a**, **b**, **c** must not lie on a straight line.

Because of their connection with barycentric combinations, barycentric coordinates are *affinely invariant*: let **p** have barycentric coordinates u, v, w with respect to **a**, **b**, **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 3.5 illustrates more of the geometric properties of barycentric coordinates.

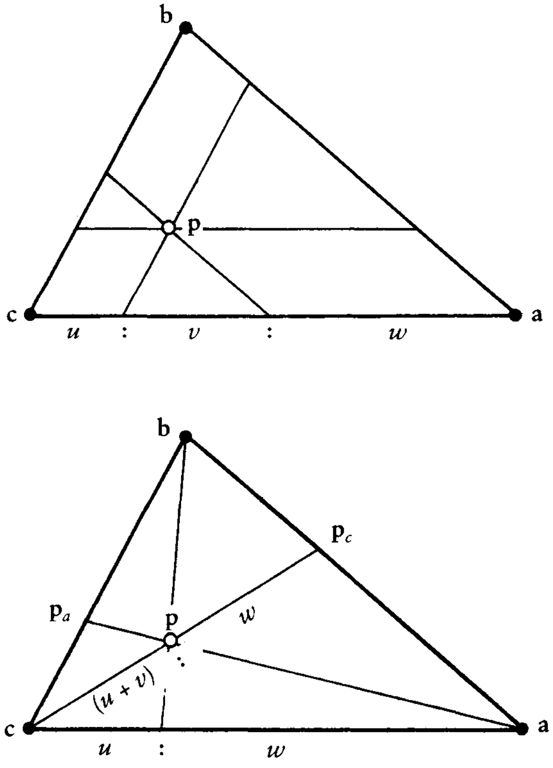


Figure 3.5 Barycentric coordinates: let $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$. The two figures show some of the ratios generated by certain straight lines through **p**.

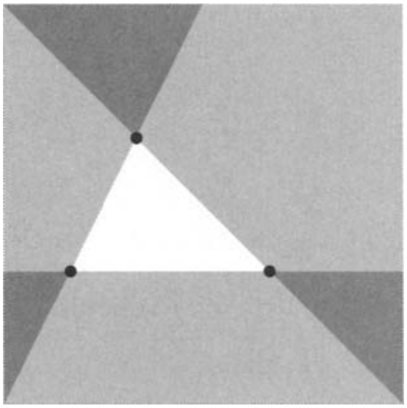


Figure 3.6 Barycentric coordinates: a triangle defines a coordinate system in the plane. Points with three positive barycentric coordinates: white. With one negative barycentric coordinate: light gray. With two negative barycentric coordinates: dark gray.

An immediate consequence of Figure 3.5 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 [253] or Berger [52], or Boehm and Prautzsch [85]).

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, whereas the remaining ones have (some) negative barycentric coordinates. Figure 3.6 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

$$\mathbf{p} = \mathbf{p}(\mathbf{u}) = \mathbf{p}(u, v, w) = u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3 \tag{3.28}$$

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 (3.28) 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 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 (3.28) 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 3D, 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 $\mathbf{p} = u_1\mathbf{p}_1 + u_2\mathbf{p}_2 + u_3\mathbf{p}_3 + u_4\mathbf{p}_4$.

3.6 Tessellations

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. Although straight line segments are 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* 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.[7] Two points are called *neighbors* if their tiles share a common edge. See Figure 3.7.

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 that is due to R. Sibson [576]: 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 occupied by $\mathbf{p}_L$; referring to Figure 3.8, let us assume it is T_k. We now draw all perpendicular bisectors between $\mathbf{p}_L$ and its neighbors, thus forming 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 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.

Although 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 T_L is formed by cutting

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

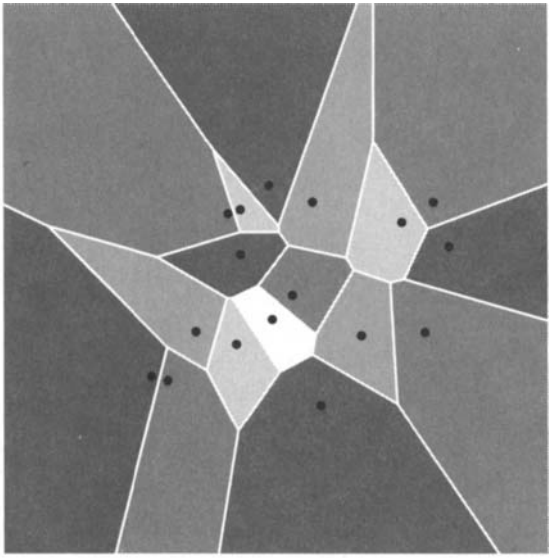


Figure 3.7 Dirichlet tessellations: a point set and its tile edges.

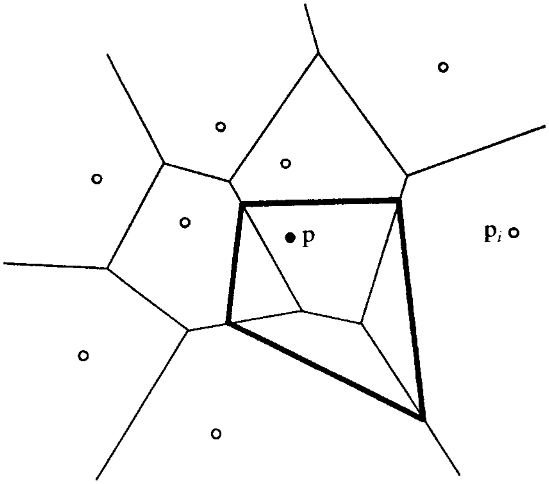


Figure 3.8 Dirichlet tessellations: a new point is inserted into an existing tessellation; its tile is outlined.

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 $\sum \mathcal{A}_i = \mathcal{A}$):

$$\mathbf{p}_L = \sum_i \frac{\mathcal{A}_i}{\mathcal{A}} \mathbf{p}_i. \tag{3.29}$$

This identity is also due to R. Sibson [576]; in case the summation is over only three neighbors, it reduces to the barycentric coordinates of Section 3.5.

3.7 Triangulations

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. Figure 3.9 was created in this way from the configuration of Figure 3.7. The points with infinite tiles are now connected; they are called *boundary points* of the triangulation.

We should mention one problem: although 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.

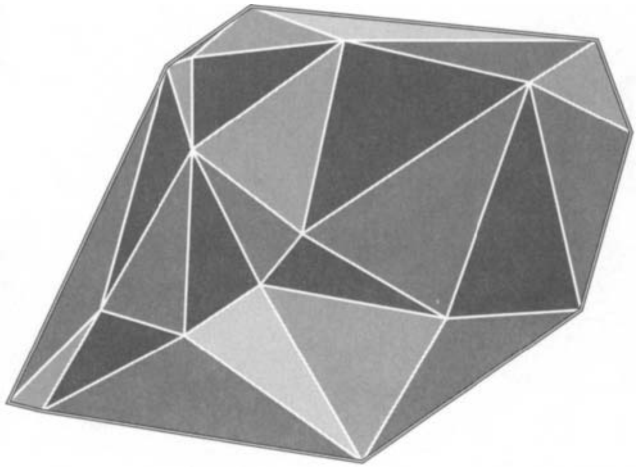


Figure 3.9 Delaunay triangulations: a point set and its Delaunay triangulation.

Clearly, there are many valid triangulations of a given point set. For example, every convex set of four points allows two different triangulations. It is now time to introduce the concept of a triangulation of a point set that is more general than the Delaunay triangulation. A triangulation $\mathcal{T}$ of a set of 2D points $\{\mathbf{p}_i\}$ is a collection of triangles such that

- The vertices of the triangles consist of the $\mathbf{p}_i$
- The interiors of any two triangles do not intersect
- If two triangles are not disjoint, then they share either a vertex or an edge

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[8]

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.

As it turns out, the Delaunay triangulation is one of the "nicer" triangulations. 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 [375]. The Delaunay triangulation is thus said to satisfy the maxmin criterion.

One might also consider the triangulation that satisfies the minmax 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 [312].

A 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 3.5—over each of the

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

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, Levin, and Rippa [180] or Brown [92]. Here, one does not just consider triangles in the plane, but rather the 3D triangles generated by the data points (x_k, y_k, z_k).

3.8 Problems

1 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.

***2** 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 both definitions are equivalent.

***3** 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.[9]

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

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

P1 Let three points be given by

$$\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 40 \\ 20 \end{bmatrix}, \begin{bmatrix} 50 \\ 10 \end{bmatrix}.$$

For $s = 0, 0.05, 0.1, \ldots, 1$ and $t = 0, 0.05, 0.1, \ldots, 1$, plot the points $\mathbf{b}[s, t]$ as defined by (3.16). Mark each point by a circle with radius 0.4.

P2 There is a 2D triangulation data set on this book's web site. Plot that triangulation using gray shades or colors such that no two neighboring triangles have the same color.

P3 Use the recursive algorithm from Section 3.6 to implement Dirichlet tessellations.

4

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 [145] and [146], both technical reports that are not easily accessible. De Casteljau's work went unnoticed until W. Boehm obtained copies of the reports in 1975. Since then, de Casteljau's work has gained more popularity.

4.1 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

$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1,$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1 + t\mathbf{b}_2,$$

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

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

$$\mathbf{b}_0^2(t) = (1-t)^2\mathbf{b}_0 + 2t(1-t)\mathbf{b}_1 + t^2\mathbf{b}_2. \tag{4.1}$$

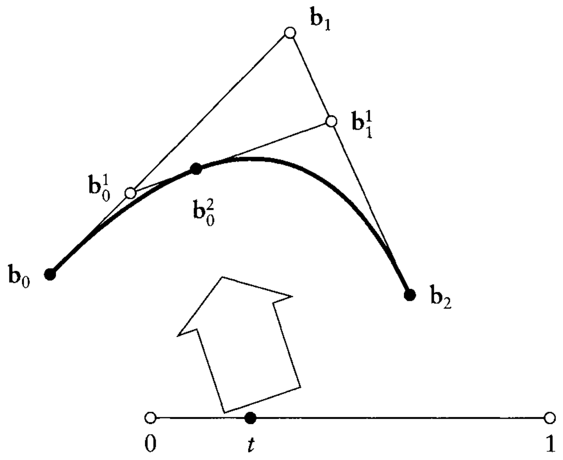


Figure 4.1 Parabolas: construction by repeated linear interpolation.

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 4.1. 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 4.1, 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 3.2.

We also note that a parabola is a plane curve, since $\mathbf{b}^2(t)$ is always a barycentric combination of three points, as is clear from inspecting (4.1). A parabola is a special case of *conic sections*, which will be discussed in Chapter 12.

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. Figure 4.1, although using a different notation, may serve as an illustration of the theorem.

4.2 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:

de Casteljau algorithm:

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

set

$$\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \quad \begin{cases} r = 1, \dots, n \\ i = 0, \dots, n-r \end{cases} \tag{4.2}$$

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* $\mathbf{b}^n$, hence $\mathbf{b}^n(t) = \mathbf{b}_0^n(t)$.

The polygon $\mathbf{P}$ formed by $\mathbf{b}_0, \dots, \mathbf{b}_n$ 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 4.2 illustrates the cubic case.

Sometimes we also write $\mathbf{b}^n(t) = \mathrm{B}[\mathbf{b}_0, \dots, \mathbf{b}_n; t] = \mathrm{B}[\mathbf{P}; t]$ or, shorter, $\mathbf{b}^n = \mathrm{B}[\mathbf{b}_0, \dots, \mathbf{b}_n] = \mathrm{B}\mathbf{P}$. This notation[3] defines B to be the (linear) operator that associates the Bézier curve with its control polygon. We say that the curve $\mathrm{B}[\mathbf{b}_0, \dots, \mathbf{b}_n]$ is the *Bernstein–Bézier approximation* to the control polygon, a terminology borrowed from approximation theory; see also Section 6.9.

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:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1 & & \\ \mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\ \mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3. \end{array} \tag{4.3}$$

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

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

[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 [65] in 1923; he called it "osculating tetrahedron."

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

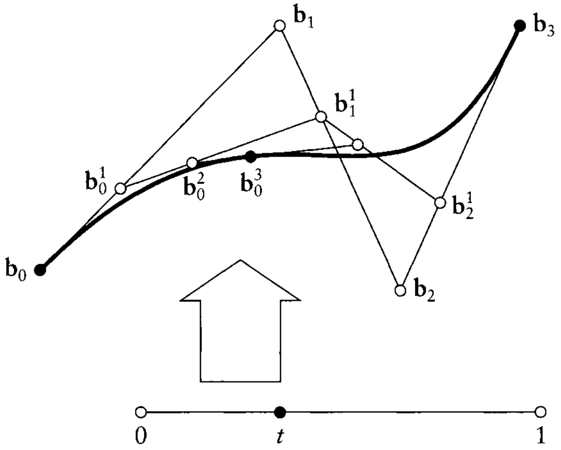


Figure 4.2 The de Casteljau algorithm: the point $\mathbf{b}_0^3(t)$ is obtained from repeated linear interpolation. The cubic case $n = 3$ is shown for $t = 1/3$.

Example 4.1 Computing a point on a Bézier curve with the de Casteljau algorithm.

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

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

storage, however: it is sufficient to use the left column only and to overwrite it appropriately.

For a numerical example, see Example 4.1. Figure 4.3 shows 60 evaluations of a Bézier curve. The intermediate points $\mathbf{b}_i^r$ are also plotted and connected.[4]

[4] Although the control polygon of the figure is symmetric, the plot is not. This is due to the organization of the plotting algorithm.

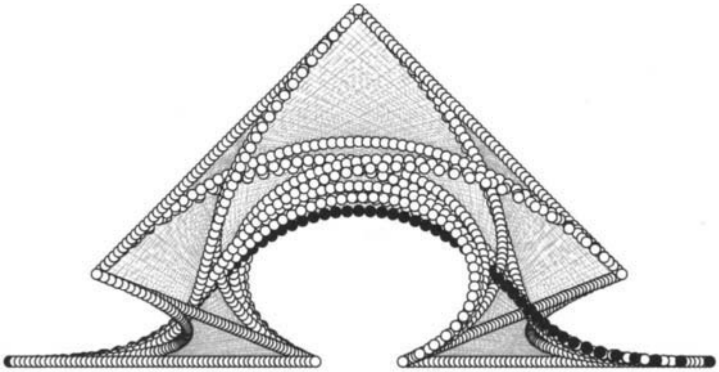


Figure 4.3 The de Casteljau algorithm: 60 points are computed on a degree six curve; all intermedediate points $\mathbf{b}_i^r$ are shown.

4.3 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 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, whereas 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 12.

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 $a \le u \le b$ 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 (3.9). The corresponding generalized de Casteljau algorithm is of the form:

$$\mathbf{b}_i^r(u) = \frac{b-u}{b-a}\mathbf{b}_i^{r-1}(u) + \frac{u-a}{b-a}\mathbf{b}_{i+1}^{r-1}(u). \tag{4.4}$$

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 since 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 "no interference" by the minmax box test.[5]

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 (4.3) 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 4.4 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 the reason 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.

[5] 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. [560].

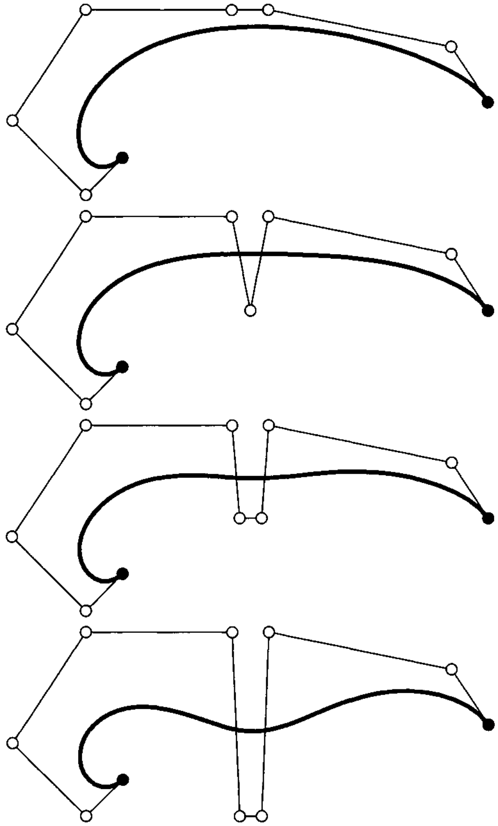


Figure 4.4 Bézier curves: some examples.

4.4 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 [147] and Ramshaw [498], [499]. Other literature includes Seidel [562], [565], [566]; DeRose and Goldman [165]; Boehm [75]; Lee [379]; and Gallier [252].

Blossoms were introduced in Section 3.4. They are closely related to the de Casteljau algorithm: 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:

$$\begin{array}{llll}
\mathbf{b}_0 & & & \\
\mathbf{b}_1 & \mathbf{b}_0^1[t_1] & & \\
\mathbf{b}_2 & \mathbf{b}_1^1[t_1] & \mathbf{b}_0^2[t_1,t_2] & \\
\mathbf{b}_3 & \mathbf{b}_2^1[t_1] & \mathbf{b}_1^2[t_1,t_2] & \mathbf{b}_0^3[t_1,t_2,t_3].
\end{array} \tag{4.5}$$

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 from Section 3.4. 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 (4.5) reduces to:

$$\begin{array}{llll}
\mathbf{b}_0 & & & \\
\mathbf{b}_1 & \mathbf{b}_0 & & \\
\mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\
\mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_1 = \mathbf{b}[0,0,1].
\end{array} \tag{4.6}$$

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 (4.3) 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

$$\begin{array}{llll}
\mathbf{b}_0 & & & \\
\mathbf{b}_1 & \mathbf{b}_0 & & \\
\mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\
\mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0^1 = \mathbf{b}[0,0,t].
\end{array} \tag{4.7}$$

Continuing in the same manner, we may write the complete scheme (4.3) as:

$$\begin{array}{llll}
\mathbf{b}_0 = \mathbf{b}[0,0,0] & & & \\
\mathbf{b}_1 = \mathbf{b}[0,0,1] & \mathbf{b}[0,0,t] & & \\
\mathbf{b}_2 = \mathbf{b}[0,1,1] & \mathbf{b}[0,t,1] & \mathbf{b}[0,t,t] & \\
\mathbf{b}_3 = \mathbf{b}[1,1,1] & \mathbf{b}[t,1,1] & \mathbf{b}[t,t,1] & \mathbf{b}[t,t,t].
\end{array} \tag{4.8}$$

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

$$\mathbf{b}_i = \mathbf{b}[0^{<n-i>}, 1^{<i>}], \tag{4.9}$$

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

The de Casteljau recursion (4.2) can now be expressed in terms of the blossom $\mathbf{b}$:

$$\mathbf{b}[0^{<n-r-i>}, t^{<r>}, 1^{<i>}] = (1-t)\mathbf{b}[0^{<n-r-i+1>}, t^{<r-1>}, 1^{<i>}] \\ + t\mathbf{b}[0^{<n-r-i>}, t^{<r-1>}, 1^{<i+1>}]. \tag{4.10}$$

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

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 using (4.4)—we find that the Bézier points $\mathbf{b}_i$ are found as the blossom values

$$\mathbf{b}_i = \mathbf{b}[a^{<n-i>}, b^{<i>}]. \tag{4.11}$$

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.

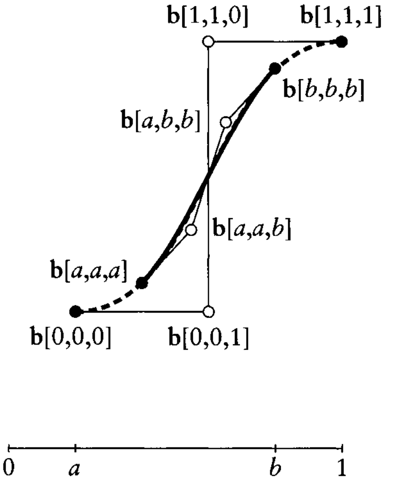


Figure 4.5 Subdivision: the relevant blossom values.

We may also find explicit formulas for blossoms; here is the case of a cubic:

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

For each step, we have exploited the fact that blossoms are multiaffine, following the inductive proof of the Leibniz equation (3.22).

We should add that every multivariate polynomial function may be interpreted as the blossom of a Bézier curve—as long as it is both symmetric and multiaffine.

4.5 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, that is, 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.

4.6 Problems

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

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

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

* **4** Show that every nonplanar cubic in $\mathbb{E}^3$ can be obtained as an affine map of the *standard cubic* (see Boehm [70])

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

P1 Write an experimental program that replaces $(1 - t)$ and t in the recursion (4.2) 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 a high heel shoe sole.

This Page Intentionally Left Blank

The Bernstein Form of a Bézier Curve

5

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.

5.1 Bernstein Polynomials

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

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \tag{5.1}$$

where the binomial coefficients are given by

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

There is a fair amount of literature on these polynomials. We cite just a few: Bernstein [53], Lorentz [399], Davis [133], and Korovkin [364]. An extensive bibliography is given in Gonska and Meier [269].

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:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \tag{5.2}$$

with

$$B_0^0(t) \equiv 1 \tag{5.3}$$

and

$$B_j^n(t) \equiv 0 \quad \text{for} \quad j \notin \{0, \ldots, n\}. \tag{5.4}$$

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*:

$$\sum_{j=0}^{n} B_j^n(t) \equiv 1. \tag{5.5}$$

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 5.1 shows the family of the four cubic 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. Recall that a Bézier curve may be written as $\mathbf{b}[t^{<n>}]$ in blossom form. Since $t = (1-t) \cdot 0 + t \cdot 1$, the blossom may be expressed as $\mathbf{b}[(1-t) \cdot 0 + t \cdot 1)^{<n>}]$, and now the Leibniz formula (3.22) directly yields

$$\mathbf{b}(t) = \mathbf{b}[t^{<n>}] = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) \tag{5.6}$$

since $\mathbf{b}_i = \mathbf{b}[0^{<n-i>}, 1^{<i>}]$ according to (4.9).

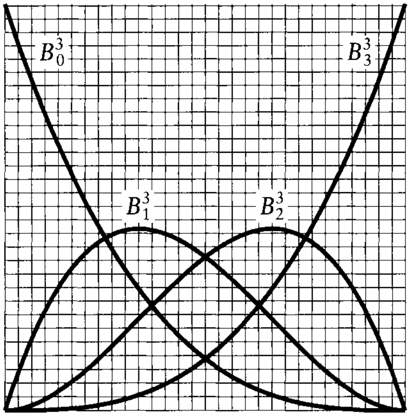


Figure 5.1 Bernstein polynomials: the cubic case.

Similarly, the intermediate de Casteljau points $\mathbf{b}_i^r$ can be expressed in terms of Bernstein polynomials of degree r:

$$\mathbf{b}_i^r(t) = \sum_{j=0}^{r} \mathbf{b}_{i+j} B_j^r(t). \tag{5.7}$$

This follows directly from

$$\mathbf{b}_i^r(t) = \mathbf{b}[0^{<n-r-i>}, t^{<r>}, 1^{<i>}]$$

and the Leibniz formula.

Equation (5.7) shows exactly how the intermediate point $\mathbf{b}_i^r$ depends on the given Bézier points $\mathbf{b}_i$. Figure 5.2 shows how these intermediate points form Bézier curves themselves.

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

$$\mathbf{b}^n(t) = \sum_{i=0}^{n-r} \mathbf{b}_i^r(t) B_i^{n-r}(t). \tag{5.8}$$

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.

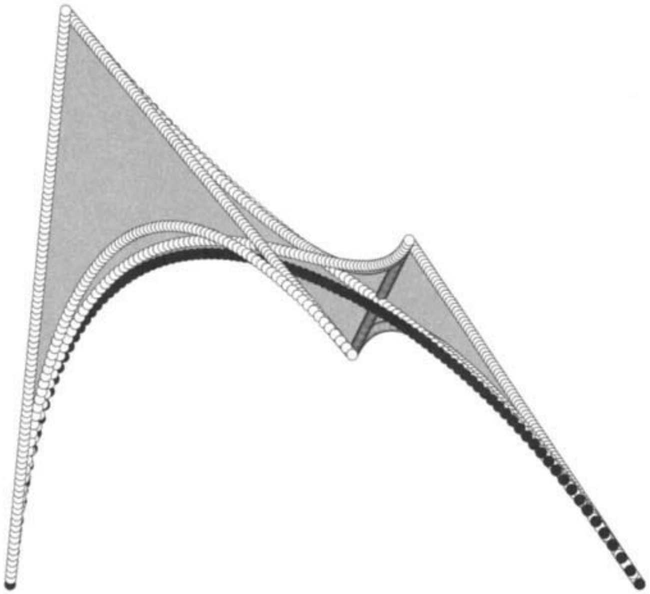


Figure 5.2 The de Casteljau algorithm: 50 points are computed on a quartic curve, and the intermediate points $\mathbf{b}_i^r$ are connected.

5.2 Properties of Bézier Curves

Many of the properties in this section have already appeared in Chapter 4. They were derived using geometric arguments. We shall now rederive several of 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, (5.5) 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

$$\sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n\left(\frac{u-a}{b-a}\right). \tag{5.9}$$

Convex hull property. This follows, since for $t \in [0,1]$, the Bernstein polynomials are nonnegative. They sum to one as shown in (5.5). For values of t outside $[0,1]$, the convex hull property does not hold; Figure 5.3 illustrates.

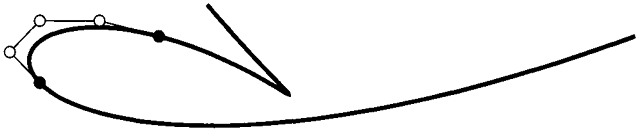


Figure 5.3 Convex hull property: a quartic Bézier curve is plotted for parameter values $t \in [-1, 2]$.

Endpoint interpolation. This is a consequence of the identities

$$\begin{aligned} B_i^n(0) &= \delta_{i,0} \\ B_i^n(1) &= \delta_{i,n} \end{aligned} \tag{5.10}$$

and (5.5). 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 4.4, 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:

$$\sum_{j=0}^{n} \mathbf{b}_j B_j^n(t) = \sum_{j=0}^{n} \mathbf{b}_{n-j} B_j^n(1-t). \tag{5.11}$$

This follows from the identity

$$B_j^n(t) = B_{n-j}^n(1-t), \tag{5.12}$$

which follows from inspection of (5.1). 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

$$\sum_{j=0}^{n} (\alpha \mathbf{b}_j + \beta \mathbf{c}_j) B_j^n(t) = \alpha \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t) + \beta \sum_{j=0}^{n} \mathbf{c}_j B_j^n(t). \tag{5.13}$$

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 14. It is illustrated in Figure 5.4.

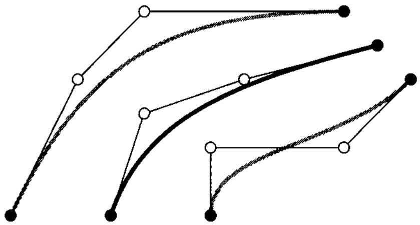


Figure 5.4 Barycentric combinations: the middle curve (black) is the average of the two outer curves (gray).

Linear precision. The following is a useful identity:

$$\sum_{j=0}^{n} \frac{j}{n} B_j^n(t) = t, \tag{5.14}$$

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 $\mathbf{q}$, that is, the initial straight line is reproduced. This property is called *linear precision.*[1]

Pseudolocal 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 1/3; thus a change of $\mathbf{b}_i$ by three units will change the curve by one unit.

5.3 The Derivatives of a Bézier Curve

We start with an identity, closely resembling Leibniz's formula for derivatives. Let t be a point on the real line, and let $\vec{v}$ be a vector in the associated 1D linear

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

space. Then

$$\mathbf{b}[(t+\vec{v})^{<n>}] = \sum_{i=0}^{n} \binom{n}{i} \mathbf{b}[t^{<n-i>}, \vec{v}^{<i>}]. \tag{5.15}$$

This is an immediate consequence of the Leibniz formula (3.22).

The derivative of a curve $\mathbf{x}(t)$ is typically defined as

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \lim_{h\to 0} \frac{1}{h}[\mathbf{x}(t+h) - \mathbf{x}(t)].$$

We will be a little more precise and observe that t is a 1D point, whereas h is a 1D vector. We thus denote it by $\vec{h}$ and obtain

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \lim_{\vec{h}\to \vec{0}} \frac{1}{|\vec{h}|}[\mathbf{x}(t+\vec{h}) - \mathbf{x}(t)].$$

Invoking (5.15), we have

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \lim_{\vec{h}\to \vec{0}} \frac{1}{|\vec{h}|}\left[\sum_{i=0}^{n} \binom{n}{i} \mathbf{b}[t^{<n-i>}, \vec{h}^{<i>}] - \mathbf{b}[t^{<n>}]\right]. \tag{5.16}$$

For $i=0$, two terms $\mathbf{b}[t^{<n>}]$ cancel. We expand the rest and factor in the term $|\vec{h}|$:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \lim_{\vec{h}\to \vec{0}} \left(n\mathbf{b}\left[t^{<n-1>}, \frac{\vec{h}}{|\vec{h}|}\right] + \binom{n}{2}\mathbf{b}\left[t^{<n-2>}, \frac{\vec{h}}{|\vec{h}|}, \vec{h}\right] + \ldots \right)$$

We observe that $\frac{\vec{h}}{|\vec{h}|} = \vec{1}$. Taking the limit annihilates all other terms containing $\vec{h}$, and we thus have

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = n\mathbf{b}[t^{<n-1>}, \vec{1}]. \tag{5.17}$$

Figure 5.5 illustrates the cubic case.

From now on, we use the expression $\dot{\mathbf{x}}(t)$ for the first derivative.

This has two possible interpretations. For the first one, we perform a de Casteljau step with respect to $\vec{1}$, and then $n-1$ steps with respect to t; as an equation:

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

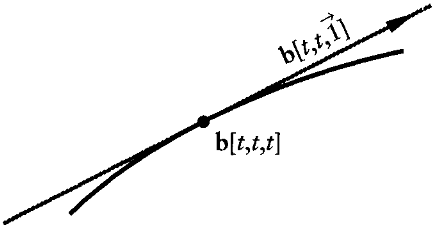


Figure 5.5 Blossoms and derivatives: the underlying geometry.

This can be simplified somewhat by the introduction of the *forward difference operator* Δ:

$$\Delta \mathbf{b}_j = \mathbf{b}_{j+1} - \mathbf{b}_j. \tag{5.18}$$

We now have for the derivative of a Bézier curve:

$$\dot{\mathbf{x}}(t) = n \sum_{j=0}^{n-1} \Delta \mathbf{b}_j B_j^{n-1}(t); \quad \Delta \mathbf{b}_j \in \mathbb{R}^3. \tag{5.19}$$

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, that is, *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 5.6 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 [244], Bézier [59], or Sederberg and Wang [559].

For a second interpretation of (5.17), we first perform $n - 1$ steps of the de Casteljau algorithm, resulting in the two points $\mathbf{b}_1^{n-1}(t)$ and $\mathbf{b}_0^{n-1}(t)$. Now performing one step with respect to $\vec{1}$ yields (after multiplication by n):

$$\dot{\mathbf{x}}(t) = n\big(\mathbf{b}_1^{n-1}(t) - \mathbf{b}_0^{n-1}(t)\big). \tag{5.20}$$

Thus the first derivative vector is a "byproduct" of the de Casteljau algorithm; see Figure 4.2. The de Casteljau algorithm is not the fastest way to evaluate a Bézier curve, but this property makes it a desirable tool: very often, we not only need a point on a curve, but the derivative vector as well. Using (5.20), we get both in parallel. The two ways of computing the derivative are shown in Example 5.1.

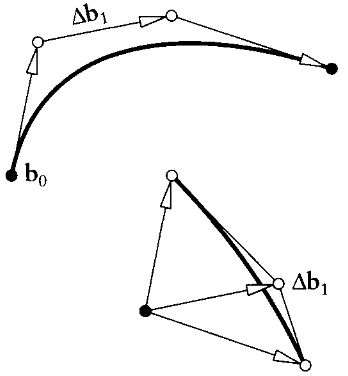


Figure 5.6 Derivatives: a Bézier curve and its first derivative curve (scaled down by a factor of three). Note that this derivative curve does not change if a translation is applied to the original curve.

Example 5.1 Two ways to compute derivatives.

To compute the derivative of the Bézier curve from Example 4.1, we could form the first differences of the control points and evaluate the corresponding quadratic curve at $t = 1/2$:

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

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.

Higher derivatives follow the same pattern:

$$\frac{\mathrm{d}^r \mathbf{x}(t)}{\mathrm{d}t^r} = \frac{n}{(n-r)!}\mathbf{b}[t^{\langle n-r\rangle}, \vec{1}^{\langle r\rangle}]. \tag{5.21}$$

To compute these derivatives from the Bézier points, we first generalize the forward difference operator (5.18): the *iterated forward difference operator* Δ^r is defined by

$$\Delta^r \mathbf{b}_j = \Delta^{r-1}\mathbf{b}_{j+1} - \Delta^{r-1}\mathbf{b}_j. \tag{5.22}$$

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:

$$\Delta^r \mathbf{b}_i = \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j} \mathbf{b}_{i+j}. \tag{5.23}$$

The r^{th} derivative of a Bézier curve is now given by

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(t) = \frac{n!}{(n-r)!} \sum_{j=0}^{n-r} \Delta^r \mathbf{b}_j B_j^{n-r}(t). \tag{5.24}$$

Two important special cases of (5.24) are given by $t = 0$ and $t = 1$. Because of (5.10), we obtain

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(0) = \frac{n!}{(n-r)!}\Delta^r \mathbf{b}_0 \tag{5.25}$$

and

$$\frac{\mathrm{d}^r}{\mathrm{d}t^r}\mathbf{b}^n(1) = \frac{n!}{(n-r)!}\Delta^r \mathbf{b}_{n-r}. \tag{5.26}$$

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

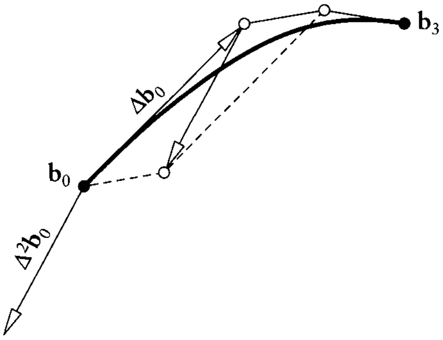


Figure 5.7 Endpoint derivatives: the first and second derivative vectors at $t = 0$ are multiples of the first and second difference vectors at $\mathbf{b}_0$.

$\mathbf{b}_1$ define the tangent at $t = 0$, provided they are distinct.[2] 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 5.7.

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

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

Using (5.25), we have

$$\mathbf{b}^n(t) = \sum_{j=0}^{n} \binom{n}{j} \Delta^j \mathbf{b}_0\, t^j. \tag{5.27}$$

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

If $\mathbf{x}(t)$ is defined over an interval $[a, b]$, (5.17) becomes

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \frac{n}{b-a}\mathbf{b}[t^{<n-1>}, \vec{1}]. \tag{5.28}$$

[2] 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.

5.4 Domain Changes and 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 5.8. 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 4.4. There, (4.11) 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:

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

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

$$\mathbf{c}_j = \mathbf{b}_0^j(c). \tag{5.29}$$

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 (5.11), 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 4.1 and 4.2, we see the two subpolygons defining the arcs from $\mathbf{b}^n(0)$ to $\mathbf{b}^n(t)$ and from $\mathbf{b}^n(t)$ to $\mathbf{b}^n(1)$.

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

$$\mathbf{d}_j = \mathbf{b}[1^{<n-j>}, d^{<j>}] = \mathbf{b}_{n-j}^j(d).$$

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 [146], was rigorously proved by E. Staerk [578]. Our blossom development is due to Ramshaw [498] and de Casteljau [147].

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 [369].

Convergence of this repeated subdivision process is very fast (see Cohen and Schumaker [123] and Dahmen [131]), and thus it has many practical applica-

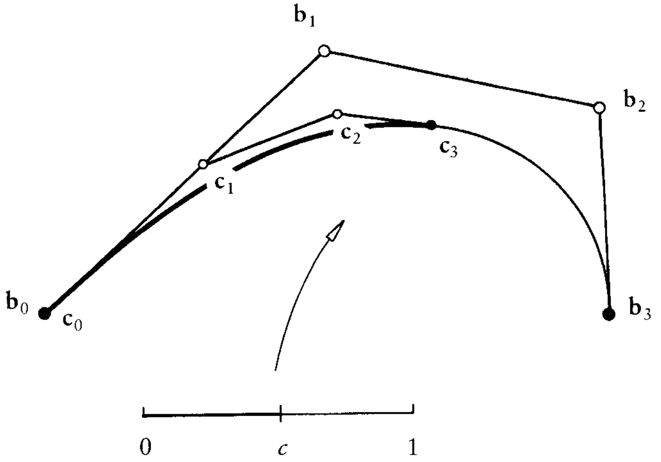


Figure 5.8 Subdivision: two Bézier polygons describing the same curve: one (the $\mathbf{b}_i$) is associated with the parameter interval $[0, 1]$, the other (the $\mathbf{c}_i$) with $[0, c]$.

tions. 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 L, if they exist.

If the curve and 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 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 5.9 gives an example.

Having found the minmax box, it is trivial to determine if it interferes with 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 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.[3] 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.

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

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

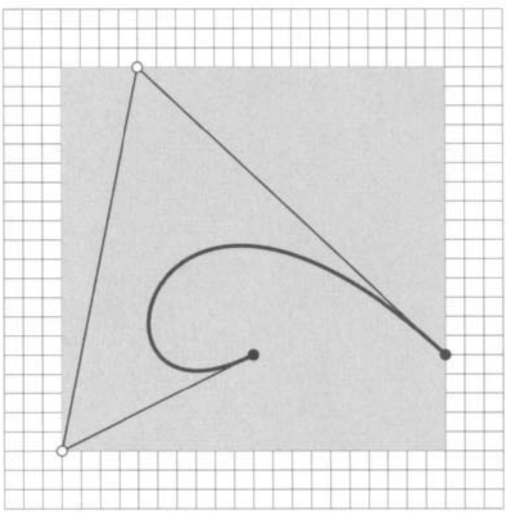


Figure 5.9 The minmax box of a Bézier curve: the smallest rectangle that contains the curve's control polygon.

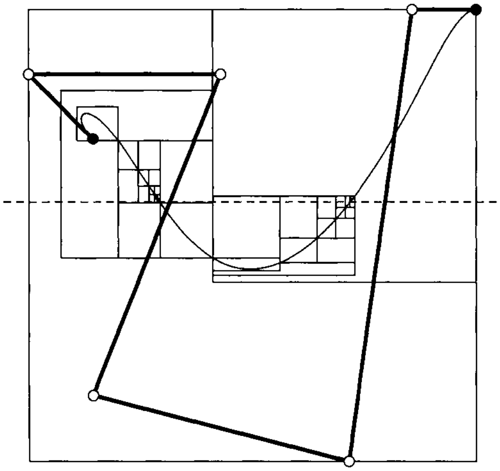


Figure 5.10 Subdivision: finding the intersections of a curve with a line (dashed). Note the clustering of minmax boxes near the intersection points.

polygon are almost collinear, we may replace them with 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 5.10 gives an example. Note how the subdivision process finds *all* intersection points. These points will not, however, be recorded by increasing values of t.

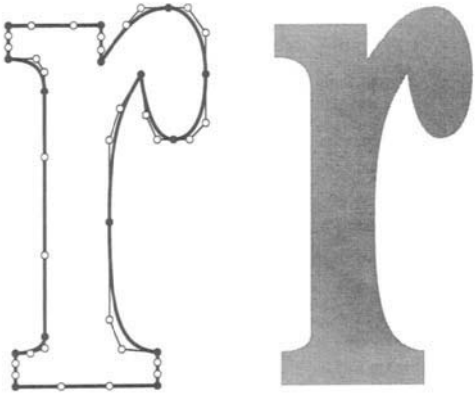


Figure 5.11 Font design: the characters in this book are stored as a sequence of cubic Bézier curves.

5.5 Composite Bézier Curves

Curves may be composed of several Bézier curves in order to generate shapes that are too complex for a single Bézier curve to handle. For example, Figure 5.11 shows how composite Bézier curves may be used in *font design*.[4]

In piecing Bézier curves together, we need to control the smoothness of the resulting curve. Let $\mathbf{b}_0, \ldots, \mathbf{b}_3$ and $\mathbf{b}_3, \ldots, \mathbf{b}_6$ be the Bézier points of two cubic curve segments $\mathbf{x}_-$ and $\mathbf{x}_+$. Since they both share the point $\mathbf{b}_3$, they clearly form a continuous, or C^0, curve. With this minimal continuity requirement, the two curves may form a corner; for several examples, see Figure 5.11.

But if we want to ensure that the two pieces meet smoothly, more care is called for. Based on our knowledge of endpoint derivatives from Section 5.3, the three points $\mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ must be collinear. That condition ensures that the tangent[5] at $\mathbf{b}_3$ is the same for both curves. Again, consult Figure 5.11 for examples. Curves with a continuously changing tangent are called G^1, or first-order geometrically continuous; see Chapter 11.

A stronger condition is to require that the two curve segments form a C^1, or continuously differentiable curve. Since the derivative of a curve (more precisely, the length of the derivative vector) depends on the domain of the curve, we need to introduce domains for our two curve segments. We adopt the convention that $\mathbf{x}_-$ is defined over an interval $[a, b]$ and that $\mathbf{x}_+$ is defined over $[b, c]$. The derivatives

[4] This book was printed using the PostScript language. It represents all characters as piecewise cubic Bézier curves in order to have a *scalable* font set. As an estimate, the text in this book is made up using about 10 million cubic Bézier curves.

[5] By "tangent," we refer to the tangent line, not to the derivative vector!

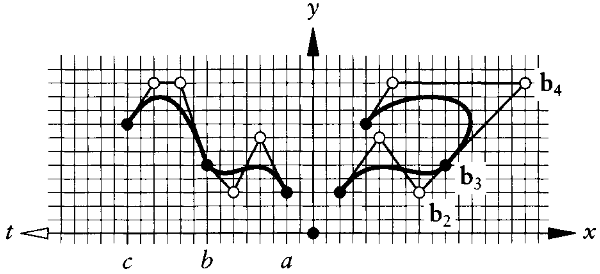


Figure 5.12 Composite curves: a C^0 example.

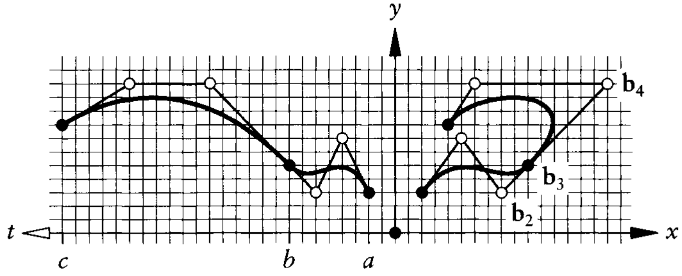


Figure 5.13 Composite curves: a C^1 example.

of both segments at parameter value b are now obtained using (5.28):

$$\frac{3}{b-a}[\mathbf{b}_3 - \mathbf{b}_2] = \frac{3}{c-b}[\mathbf{b}_4 - \mathbf{b}_3]. \tag{5.30}$$

A geometric interpretation is that the ratio of the three points $\mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ is the same as the ratio of the three parameter values a, b, c. This is a much stronger condition than that for G^1 continuity above!

Figures 5.12 and 5.13 illustrate this difference. The composite parametric curves—in the x, y-coordinate systems—are identical. The difference is their domains: in Figure 5.12, we chose $a, b, c = 0, 1, 2$. Thus ratio$(a, b, c) = 1$ while the figure suggests that ratio$(\mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4) = 1/3$. Hence the composite curve is not C^1, despite the collinearity of the points $\mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$. This is demonstrated by the cross plot (y-part only): each component must form a C^1 function for a curve to be C^1. Clearly, the y-component is not C^1.

If we adjust the domain, however, such that the range geometry is reflected by the domain geometry, we can achieve C^1. This is shown in Figure 5.13, where now ratio$(a, b, c) = 1/3$. This results in C^1 components, and hence also in a C^1 composite curve.

Higher-order smoothness of composite curves is best dealt with in the context of B-spline curves and blossoms; see Section 8.7.

5.6 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 4.4, we can write:

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

Invoking our knowledge about derivatives, we have:

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

Therefore,

$$\mathbf{p}_1(t) = \mathbf{b}(t) + \frac{t_1 - t}{n}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}(t). \tag{5.31}$$

The polynomial $\mathbf{p}_1$ is called *first polar* of $\mathbf{b}(t)$ with respect to t_1. Figure 5.14 illustrates the geometric significance of (5.31): the tangent at any point $\mathbf{b}(t)$ intersects the polar 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 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 $\mathbf{b}[0, t^{<n-1>}]$: this is the polynomial defined by $\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}$. Similarly, $\mathbf{b}[1, t^{<n-1>}]$ is defined by $\mathbf{b}_1, \ldots, \mathbf{b}_n$. This observation may be used for a proof of (4.9).

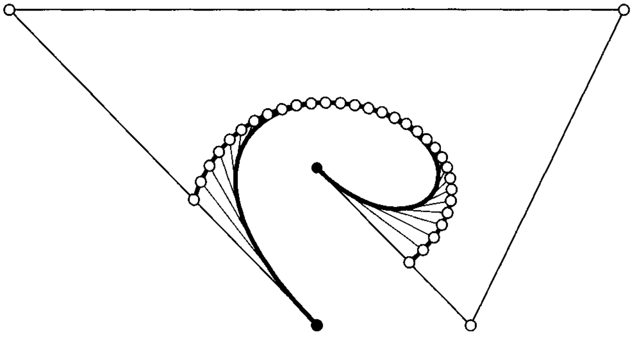


Figure 5.14 Polars: the polar $\mathbf{p}_1(t)$ with respect to $t_1 = 0.4$ is intersected by the tangents of the given curve $\mathbf{b}(t)$.

Returning to the general case, we may repeat the process of forming polars, thus obtaining a second polar $\mathbf{p}_{1,2}(t) = \mathbf{b}[t_1, t_2, t^{<n-2>}]$, and so on. 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 [499]. The preceding geometric arguments are due to S. Jolles, who developed a geometric theory of blossoming as early as 1886 in [346].[6]

5.7 The Matrix Form of a Bézier Curve

Some authors (Faux and Pratt [228], Mortenson [433], Chang [106]) 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:

$$\mathbf{x}(t) = [\,\mathbf{c}_0 \quad \ldots \quad \mathbf{c}_n\,] \begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix}.$$

One can take this a step further and write

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

$$\begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} = \begin{bmatrix} m_{00} & \dots & m_{0n} \\ \vdots & & \vdots \\ m_{n0} & \dots & m_{nn} \end{bmatrix} \begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix}. \tag{5.32}$$

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

$$m_{ij} = (-1)^{j-i}\binom{n}{j}\binom{j}{i}, \tag{5.33}$$

a simple consequence of (5.27).

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 (5.32) 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.

5.8 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 superefficient 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 we present the routine to intersect a Bézier curve with a straight line (the straight line is assumed to be the x-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.
*/
```

5.9 Problems

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: [35], [411].

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: [336], [341].

This Page Intentionally Left Blank

Bézier Curve Topics 6

6.1 Degree Elevation

Suppose we were designing with Bézier curves as described in Section 4.3, trying to use a Bézier curve of degree n. After modifying 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, we 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 (6.24) to (6.26)—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

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

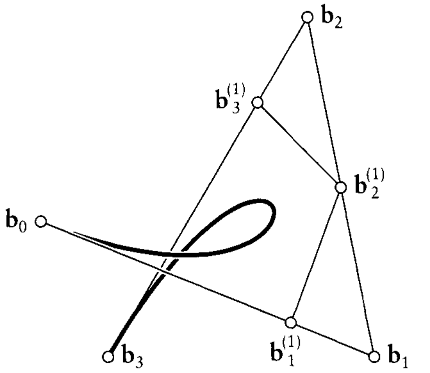


Figure 6.1 Degree elevation: both polygons define the same (degree three) curve.

Combining both sums and comparing coefficients yields the desired result:

$$\mathbf{b}_i^{(1)} = \frac{i}{n+1}\mathbf{b}_{i-1} + \left(1 - \frac{i}{n+1}\right)\mathbf{b}_i; \quad i = 0, \ldots, n+1. \tag{6.1}$$

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 $\mathcal{E}\mathbf{P}$ lies in the convex hull of the old one. Figure 6.1 gives an example. Note how $\mathcal{E}\mathbf{P}$ is "closer" to the curve $\mathcal{B}\mathbf{P}$ than the original polygon $\mathbf{P}$.

Although 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 (4.9). After some experimentation (try the case $n = 2$!), it is easy to see that the blossom is given by

$$\mathbf{b}^{(1)}[t_1, \ldots, t_{n+1}] = \frac{1}{n+1}\sum_{j=0}^{n+1} \mathbf{b}[t_1, \ldots, t_{n+1}|t_j]. \tag{6.2}$$

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 (4.9):

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

Inspection of all terms that now arise in (6.2) reveals that the point $\mathbf{b}_{i-1}$ appears i times and that the point $\mathbf{b}_i$ appears $n+1-i$ times, thus reproving 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 knows only about cubics. All you have to do is degree elevate your parabola.

6.2 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 $\mathbf{P}, \mathcal{E}\mathbf{P}, \mathcal{E}^2\mathbf{P}$, and so on. 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

$$\mathbf{b}_i^{(r)} = \sum_{j=0}^{n} \mathbf{b}_j \binom{n}{j} \frac{\binom{r}{i-j}}{\binom{n+r}{i}}. \tag{6.3}$$

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:

$$\lim_{r\to\infty} \mathcal{E}^r\mathbf{P} = \mathcal{B}\mathbf{P}. \tag{6.4}$$

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. We can now show (using Stirling's formula) that

$$\lim_{i/(n+r)\to t} \frac{\binom{r}{i-j}}{\binom{r+n}{i}} = t^j(1-t)^{n-j}, \tag{6.5}$$

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

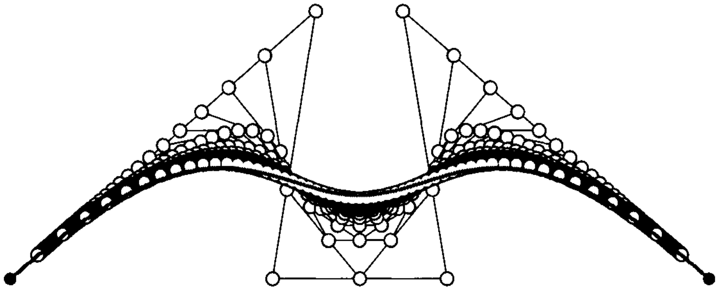


Figure 6.2 Degree elevation: a sequence of polygons approaching the curve that is defined by each of them.

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

Figure 6.2 shows an example of the limit behavior of the polygons $\mathcal{E}^r\mathbf{P}$.

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 A. R. Forrest [244] and proved in Farin [187]. The preceding proof follows an approach taken by J. Zhou [630]. Degree elevation may be generalized to "corner-cutting"; for a brief description, see Section 8.4.

6.3 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 than does the polygon **P**. Degree elevation is an instance of piecewise linear interpolation, and we know that operation is variation diminishing (see Section 3.2). 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

[2] The variation diminishing property was first investigated by I. Schoenberg [543] in the context of B-spline approximation.

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!

Though 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 jotting down a few examples. Yet it is false, as shown by Prautzsch [494].

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

The equations for degree elevation may be combined into one matrix equation:

$$\begin{bmatrix} 1 & & & & \\ \star & \star & & & \\ & \star & \star & & \\ & & \ddots & & \\ & & & \star & \star \\ & & & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0^{(1)} \\ \vdots \\ \mathbf{b}_{n+1}^{(1)} \end{bmatrix}. \tag{6.6}$$

Abbreviated:

$$M\mathbf{B} = \mathbf{B}^{(1)}, \tag{6.7}$$

M being a matrix with $n+2$ rows and $n+1$ columns.

In *degree reduction*, we seek to approximate a degee $n+1$ curve by one of degree n. In terms of (6.7), this means that we would be given $\mathbf{B}^{(1)}$ and wish to find $\mathbf{B}$. Clearly this is not possible in terms of solving a linear system, since M is not a square matrix.

A trick will help: simply multiply both sides of (6.7) by M^{T}, thus getting

$$M^{\mathrm{T}}M\mathbf{B} = M^{\mathrm{T}}\mathbf{B}^{(1)}. \tag{6.8}$$

Now we have a linear system for the unknown $\mathbf{B}$ with a square coefficient matrix $M^{\mathrm{T}}M$—and any linear system solver will do the job![3]

The linear system (6.8) is called the system of *normal equations*. It guarantees that $\mathbf{B}$ is optimal in a least squares sense; for more discussion on this technique, see Section 7.8. It is also optimal in the sense that the original and the degree reduced curves are as close as possible in the least squares sense; see [406].

In many cases, it might be desired that $\mathbf{b}_0 = \mathbf{b}_0^{(1)}$ and $\mathbf{b}_n = \mathbf{b}_{n+1}^{(1)}$. Our least squares solution will not meet these conditions in most cases—the simplest solution is to enforce them after having found $\mathbf{B}$.

Degree reduction has received a fair amount of attention in the literature; we cite [97], [182], [406], [472], [612].

6.5 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 (5.14). We can now write our

[3] This linear system is *bidiagonal*, and may be solved much faster using simple backward substitution.

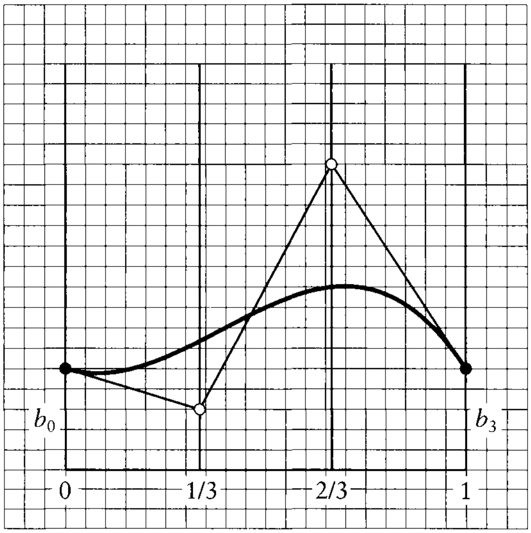


Figure 6.3 Functional curves: the control polygon of a cubic polynomial has abscissa values of 0, 1/3, 2/3, 1.

functional curve as

$$\mathbf{b}(t) = \sum_{j=0}^{n} \begin{bmatrix} j/n \\ b_j \end{bmatrix} B_j^n(t). \tag{6.9}$$

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 6.3 illustrates the cubic case. We also emphasize that the b_i are real numbers, not points; we call the b_i *Bézier ordinates*.

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 $a + i(b - a)/n; \quad i = 0, \ldots, n$.

6.6 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 6.4 shows the decomposition of a Bézier curve into its two coordinate functions.

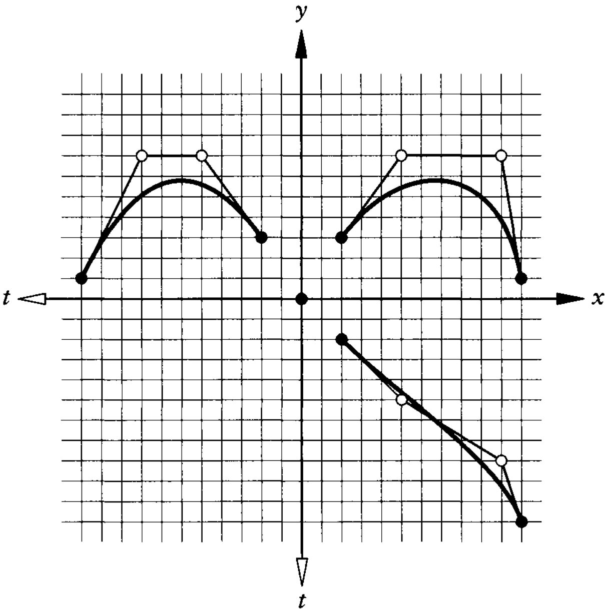


Figure 6.4 Cross plots: a two-dimensional Bézier curve together with its two coordinate functions.

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.

6.7 Integrals

The derivative of a polynomial function $b(t)$ in Bernstein form is given by

$$\dot{b}(t) = n \sum_{i=0}^{n-1} \Delta b_i B_i^{n-1}(t).$$

In order to find the indefinite integral, or antiderivative, we must invert this process. So if a polynomial in Bernstein form is given by its control ordinates

$$b_0, \ldots, b_n, \tag{6.10}$$

then the set of control ordinates

$$\frac{1}{n+1}[c, c+b_0, c+b_0+b_1, \ldots, c+\sum_{i=0}^{n} b_i]$$

defines a polynomial $B(t)$ whose derivative is exactly the one defined by the control ordinates (6.10). The scalar c is the usual constant encountered in indefinite integration.

Since

$$\int_0^1 b(t)\mathrm{d}t = B(1) - B(0), \tag{6.11}$$

we have immediately

$$\int_0^1 b(t)\mathrm{d}t = \frac{1}{n+1}\sum_{i=0}^{n} b_i. \tag{6.12}$$

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

$$\int_0^1 B_j^n(x)\mathrm{d}x = \frac{1}{n+1}; \tag{6.13}$$

that is, all basis functions B_i^n (for a fixed n) have the same definite integral.

6.8 The Bézier Form of a Bézier Curve

In his work ([56], [57], [58], [59], [60], [61], [63], see also Vernet [599]), 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:

$$\mathbf{b}^n(t) = \sum_{j=0}^{n} \mathbf{c}_j F_j^n(t), \tag{6.14}$$

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

$$F_i^n(t) = (1-t)F_i^{n-1}(t) + tF_{i-1}^{n-1}(t) \tag{6.15}$$

with

$$F_0^0(t) = 1, \quad F_{r+1}^r(t) = 0, \quad F_{-1}^r(t) = 1. \tag{6.16}$$

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

$$F_i^n = \sum_{j=i}^{n} B_j^n. \tag{6.17}$$

A consequence of (6.17) 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 (6.14) is not a barycentric combination of the $\mathbf{c}_j$. In fact, $\mathbf{c}_0$ is a point whereas the other $\mathbf{c}_j$ are vectors. The following relations hold:

$$\mathbf{c}_0 = \mathbf{b}_0, \tag{6.18}$$

$$\mathbf{c}_j = \Delta \mathbf{b}_{j-1}; \quad j > 0. \tag{6.19}$$

This undesirable distinction between points and vectors was abandoned soon after Forrest's discovery that the Bézier form (6.14) of a Bézier curve could be written in terms of Bernstein polynomials (see the appendix in [59]).

Comparing both forms, we notice that the Bernstein form is symmetric with respect to t and $1 - t$, whereas the Bézier form is not. Let us assume the defining coefficients $\mathbf{b}_i$ or $\mathbf{c}_i$ are affected by some numerical error and then let us check the effect on the point $\mathbf{x}(1)$. In the Bernstein form, $\mathbf{x}(1)$ changes its value only if $\mathbf{b}_n$ is in error. In the Bézier form, the value of $\mathbf{x}(1)$ is the sum of all errors in the $\mathbf{c}_i$. If those cancel out, no harm is done—but if they do not, we may see serious error accumulation!

6.9 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 (see Davis [133] or Korovkin [364]).

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, that is, 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, we would have to choose values of n in the thousands or even millions in order to obtain a reasonable closeness of fit (see Korovkin [364] 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.

6.10 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. Farouki and V. Rajan [225].

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

$$t^i = \sum_{j=i}^{n} \frac{\binom{j}{i}}{\binom{n}{i}} B_j^n(t) \tag{6.20}$$

and

$$B_i^n(t) = \sum_{j=i}^{n} (-1)^{j-i} \binom{n}{j} \binom{j}{i} t^j. \tag{6.21}$$

Recursion:

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

Subdivision:

$$B_i^n(ct) = \sum_{j=0}^{n} B_i^j(c) B_j^n(t). \tag{6.22}$$

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:

$$\int_0^t B_i^n(x)\mathrm{d}x = \frac{1}{n+1} \sum_{j=i+1}^{n+1} B_j^{n+1}(t), \tag{6.23}$$

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

Three degree elevation formulas:

$$(1-t)B_i^n(t) = \frac{n+1-i}{n+1} B_i^{n+1}(t), \tag{6.24}$$

$$tB_i^n(t) = \frac{i+1}{n+1} B_{i+1}^{n+1}(t), \tag{6.25}$$

$$B_i^n(t) = \frac{n+1-i}{n+1} B_i^{n+1}(t) + \frac{i+1}{n+1} B_{i+1}^{n+1}(t). \tag{6.26}$$

Product:

$$B_i^m(u) B_j^n(u) = \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}} B_{i+j}^{m+n}(u). \tag{6.27}$$

6.11 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.
*/
```

6.12 Problems

***1** Prove (6.17).

***2** Prove the relationship between the "Bézier" and the Bernstein form for a Bézier curve (6.14).

***3** Prove that

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

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

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

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 6.4. Work with the Bézier polygon supplied in the file `degred.dat`.

This Page Intentionally Left Blank

Polynomial Curve Constructions

7

Polynomial interpolation is a fundamental concept for all of CAGD. Although its uses are limited to low degrees, the basic concept still needs to be understood in order to develop new algorithms. If the amount of data is too large for interpolation to be successful, one uses approximation methods instead.

7.1 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 3.1. The Aitken recursion computes a point on the interpolating polynomial through a sequence

[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 9.6.

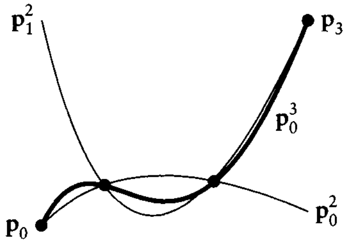


Figure 7.1 Polynomial interpolation: a cubic interpolating polynomial may be obtained as a "blend" of two quadratic interpolants.

of *repeated linear interpolations*, starting with

$$p_i^1(t) = \frac{t_{i+1}-t}{t_{i+1}-t_i}p_i + \frac{t-t_i}{t_{i+1}-t_i}p_{i+1}; \quad i=0,\ldots,n-1.$$

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 p_0^{n-1} that interpolates to the n first data points $p_0,\ldots,p_{n-1}$, and also a polynomial p_1^{n-1} that interpolates to the n last data points $p_1,\ldots,p_n$. Under these assumptions, it is easy to write down the form of the final interpolant, now called p_0^n:

$$p_0^n(t) = \frac{t_n-t}{t_n-t_0}p_0^{n-1}(t) + \frac{t-t_0}{t_n-t_0}p_1^{n-1}(t). \tag{7.1}$$

Figure 7.1 illustrates this form for the cubic case.

Let us verify that (7.1) does in fact interpolate to all given data points p_i: for $t=t_0$,

$$p_0^n(t_0) = 1 * p_0^{n-1}(t_0) + 0 * p_1^{n-1}(t_0) = p_0.$$

A similar result is derived for $t=t_n$. Under our assumption, we have $p_0^{n-1}(t_i) = p_1^{n-1}(t_i) = p_i$ for all other values of i.

Since the weights in (7.1) sum to one identically, we get the desired $p_0^n(t_i) = p_i$.

We can now generalize (7.1) to solve the polynomial interpolation problem: starting with the given parameter values t_i and the data points $p_i = p_i^0$, we set

$$p_i^r(t) = \frac{t_{i+r}-t}{t_{i+r}-t_i}p_i^{r-1}(t) + \frac{t-t_i}{t_{i+r}-t_i}p_{i+1}^{r-1}(t); \begin{cases} r=1,\ldots,n; \\ i=0,\ldots,n-r \end{cases}. \tag{7.2}$$

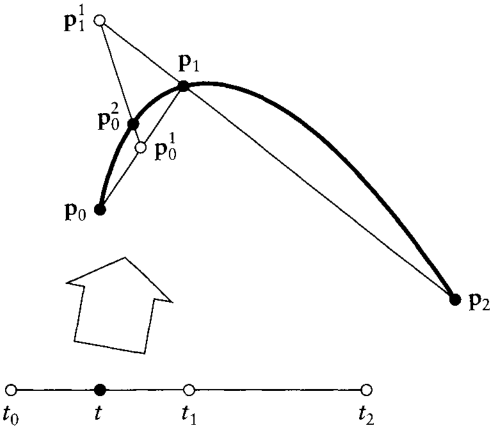


Figure 7.2 Aitken's algorithm: a point on an interpolating polynomial may be found from repeated linear interpolation.

It is clear from the preceding consideration that $\mathbf{p}_0^n(t)$ is indeed a point on the interpolating polynomial. The recursive evaluation (7.2) 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 7.2 for the quadratic case.

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

$$\begin{array}{llll} \mathbf{p}_0 & & & \\ \mathbf{p}_1 & \mathbf{p}_0^1 & & \\ \mathbf{p}_2 & \mathbf{p}_1^1 & \mathbf{p}_0^2 & \\ \mathbf{p}_3 & \mathbf{p}_2^1 & \mathbf{p}_1^2 & \mathbf{p}_0^3. \end{array} \tag{7.3}$$

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

- *Affine invariance:* This follows since Aitken's 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 (7.2) 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 augment variation to an extent that renders them useless for practical problems.

7.2 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

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{p}_i L_i^n(t), \tag{7.4}$$

where the L_i^n are *Lagrange polynomials*

$$L_i^n(t) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{n} (t - t_j)}{\prod_{\substack{j=0 \\ j\neq i}}^{n} (t_i - t_j)}. \tag{7.5}$$

Before we proceed further, we should note that the L_i^n must sum to one in order for (7.4) to be a barycentric combination and thus be geometrically meaningful; we will return to this topic later.

[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.

We verify (7.4) by observing that the Lagrange polynomials are *cardinal*: they satisfy

$$L_i^n(t_j) = \delta_{i,j}, \tag{7.6}$$

with $\delta_{i,j}$ being the Kronecker delta. In other words, the ith Lagrange polynomial vanishes at all knots except at the ith one, where it assumes the value 1. Because of this property of Lagrange polynomials, (7.4) is called the *cardinal* form of the interpolating polynomial $\mathbf{p}$. The polynomial $\mathbf{p}$ has many other representations, of course (we can rewrite it in monomial form, for example), but (7.4) 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 (7.6), the L_i^n form a basis of all polynomials of degree n. Thus, (7.4) 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 (7.4). Based on that knowlege, we can conclude a property of Lagrange polynomials that was already mentioned right after (7.5), namely, that they sum to 1:

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

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

7.3 The Vandermonde Approach

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

$$\mathbf{p}^n(t) = \sum_{j=0}^{n} \mathbf{a}_j t^j. \tag{7.7}$$

[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.

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

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

In matrix form:

$$\begin{bmatrix}\mathbf{p}_0\\ \mathbf{p}_1\\ \vdots\\ \mathbf{p}_n\end{bmatrix} = \begin{bmatrix}1 & t_0 & t_0^2 & \ldots & t_0^n\\ 1 & t_1 & t_1^2 & \ldots & t_1^n\\ \vdots & \vdots & \vdots & & \vdots\\ 1 & t_n & t_n^2 & \ldots & t_n^n\end{bmatrix}\begin{bmatrix}\mathbf{a}_0\\ \mathbf{a}_1\\ \vdots\\ \mathbf{a}_n\end{bmatrix}. \tag{7.8}$$

We can shorten this to

$$\mathbf{p} = T\mathbf{a}. \tag{7.9}$$

We already know that a solution **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, that is, the vector **a** containing the coefficients $\mathbf{a}_i$, can be found from

$$\mathbf{a} = T^{-1}\mathbf{p}. \tag{7.10}$$

This should be taken only as a shorthand notation for the solution—not as an algorithm! Note that the linear system (7.9) 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 **a**. This will be even more important when we discuss tensor product surface interpolation in Section 15.4.

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

$$\mathbf{p}^n(t) = \sum_{j=0}^{n} \mathbf{c}_j F_j^n(t). \tag{7.11}$$

This reasoning again leads to a linear system (three linear systems to be more precise) for the coefficients c_j, this time with the *generalized Vandermonde* F:

$$F = \begin{bmatrix} F_0^n(t_0) & F_1^n(t_0) & \dots & F_n^n(t_0) \\ F_0^n(t_1) & F_1^n(t_1) & \dots & F_n^n(t_1) \\ \vdots & \vdots & & \vdots \\ F_0^n(t_n) & F_1^n(t_n) & \dots & F_n^n(t_n) \end{bmatrix}. \tag{7.12}$$

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.

7.4 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 7.3: polynomial interpolants may *oscillate.*

The top curve in that figure is the Lagrange interpolant to 21 points read off from a quarter of an ellipse. The data points were computed to a precision of six digits. Slightly changing the input data points, namely, by reducing their accuracy to four digits, produces the bottom interpolant. This is a disturbing phenomenon: miniscule changes in the input data may result in serious changes of the result. Processes with that behavior are called *ill conditioned.* From a more geometric viewpoint, we may state that polynomial interpolation is not *shape preserving.*

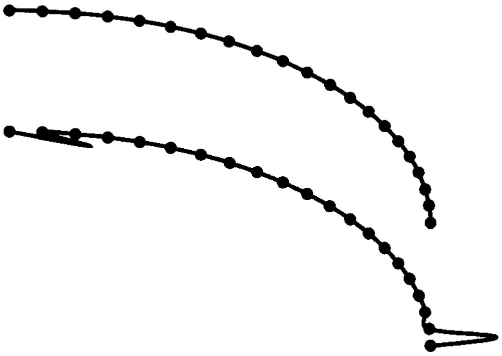


Figure 7.3 Lagrange interpolation: The top and bottom input data differ only by the amount of accuracy: six digits after the decimal point, top; four digits, bottom.

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 **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, we would expect the corresponding interpolants to converge to the sampled curve **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 [513]. 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, that is, the number of operations necessary to construct and then evaluate the interpolant. Solving the Vandermonde system (7.8) 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!

7.5 Cubic Hermite Interpolation

Polynomial interpolation is not restricted to interpolation to point data; we 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 a set of points $\mathbf{p}_i$, associated parameter values t_i, and associated tangent vectors (i.e., derivatives) $\mathbf{m}_i$. We just consider the case of two points $\mathbf{p}_0, \mathbf{p}_1$ and two tangent vectors $\mathbf{m}_0, \mathbf{m}_1$, setting $t_0 = 0$ and $t_1 = 1$. The objective is to find a cubic polynomial curve **p** that interpolates to these data:

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

where the dot denotes differentiation.

We will write **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:

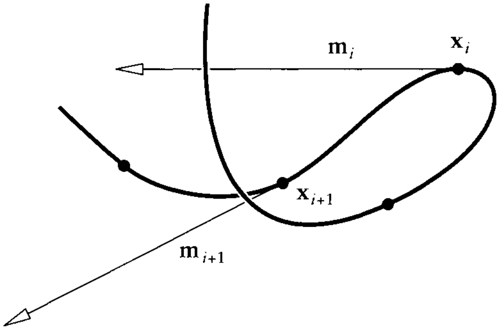


Figure 7.4 Cubic Hermite interpolation: the given data—points and tangent vectors—together with an interpolating cubic.

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

For the remaining two, we recall (from Section 5.3) 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—for the case of a general set of points and tangent vectors—is shown in Figure 7.4.

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:

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

To obtain the cardinal form, we simply rearrange:

$$\mathbf{p}(t) = \mathbf{p}_0 H_0^3(t) + \mathbf{m}_0 H_1^3(t) + \mathbf{m}_1 H_2^3(t) + \mathbf{p}_1 H_3^3(t), \tag{7.13}$$

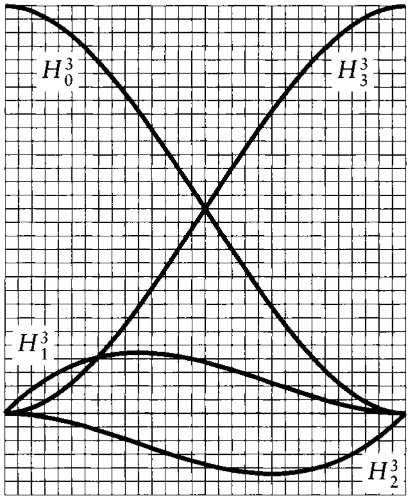


Figure 7.5 Cubic Hermite polynomials: the four H_i^3 are shown over the interval [0, 1].

where we have set[5]

$$\begin{aligned}
H_0^3(t) &= B_0^3(t) + B_1^3(t),\\
H_1^3(t) &= \frac{1}{3}B_1^3(t),\\
H_2^3(t) &= -\frac{1}{3}B_2^3(t),\\
H_3^3(t) &= B_2^3(t) + B_3^3(t).
\end{aligned} \tag{7.14}$$

The H_i^3 are called cubic Hermite polynomials and are shown in Figure 7.5.

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 $t = 1$, that is, each of the H_i^3 equals 1 for one of these four operations and is 0 for the remaining three:

[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 (7.13) was chosen since it groups coefficients according to their geometry.

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

Another important property of the H_i^3 follows from the geometry of the interpolation problem; (7.13) contains combinations of points and vectors. We know that the point coefficients must sum to 1 if (7.13) is to be geometrically meaningful:

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

This is, of course, also verified by inspection of (7.14).

Cubic Hermite interpolation has one annoying peculiarity: it is not invariant under affine domain transformations. Let a cubic Hermite interpolant be given as in (7.13), that is, 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 (7.13) becomes

$$\hat{\mathbf{p}}(\hat{t}) = \mathbf{p}_0\hat{H}_0^3(\hat{t}) + \mathbf{m}_0\hat{H}_1^3(\hat{t}) + \mathbf{m}_1\hat{H}_2^3(\hat{t}) + \mathbf{p}_1\hat{H}_3^3(\hat{t}), \tag{7.15}$$

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

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

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

$$\begin{aligned}\hat{H}_0^3(\hat{t}) &= H_0^3(t),\\ \hat{H}_1^3(\hat{t}) &= (b-a)H_1^3(t),\\ \hat{H}_2^3(\hat{t}) &= (b-a)H_2^3(t),\\ \hat{H}_3^3(\hat{t}) &= H_3^3(t),\end{aligned} \tag{7.16}$$

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

Evaluation of (7.15) 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 smaller speed of traversal. Since the magnitude of the derivative equals that speed, it must also shrink by a factor of 10.

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 above by applying 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.

7.6 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

$$\mathbf{p}(t) = \mathbf{p}_0 H_0^5(t) + \mathbf{m}_0 H_1^5(t) + \mathbf{s}_0 H_2^5(t) + \mathbf{s}_1 H_3^5(t) + \mathbf{m}_1 H_4^5(t) + \mathbf{p}_1 H_5^5(t), \quad (7.17)$$

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 presented in Chapter 9 are recommended.

7.7 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 7.6. Thus we know only 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 [483] and Nielson [447]. Both approaches, although formulated differently, yield the same result.

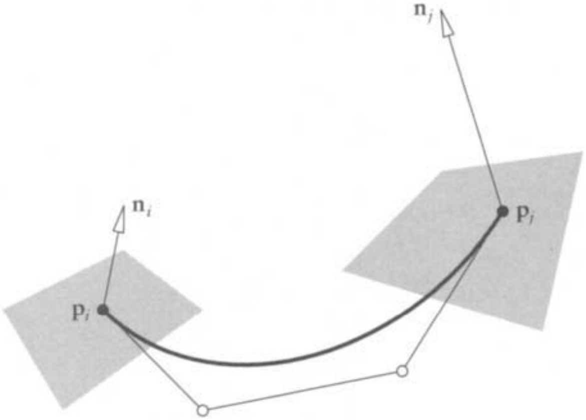


Figure 7.6 Finding cubic boundaries: although the endpoints of a boundary curve are fixed, its end tangents only have to lie in specified planes.

In order to find $\mathbf{b}_1$, project $\mathbf{b}_3$ into the plane defined by $\mathbf{b}_0 = \mathbf{p}_i$ and $\mathbf{n}_i$. This defines a tangent at $\mathbf{b}_0$. As a rule of thumb, the distance $\|\mathbf{b}_1 - \mathbf{b}_0\|$ should be roughly $0.4\|\mathbf{b}_3 - \mathbf{b}_0\|$; if our current $\mathbf{b}_1$ violates this rule, it may have to be adjusted accordingly. The remaining point $\mathbf{b}_2$ is then obtained analogously.

7.8 Least Squares Approximation

In many applications, we are given more data points than can be interpolated by a polynomial curve. In such cases, an *approximating* curve will be needed. Such a curve does not pass through the data points exactly; rather, it passes near them, still capturing the shape inherent to the given points. The technique best known for finding such curves is known as *least squares approximation.* An example is given in Figure 7.7.

To make matters more precise, assume we are given $P + 1$ data points $\mathbf{p}_0, \ldots, \mathbf{p}_P$, each $\mathbf{p}_i$ being associated with a parameter value t_i. We wish to find a polynomial curve $\mathbf{x}(t)$ of a given degree n such that the distances $\|\mathbf{p}_i - \mathbf{x}(t_i)\|$ are small. Ideally, we would have $\mathbf{p}_i = \mathbf{x}(t_i)$; $i = 0, \ldots, P$. If our polynomial curve $\mathbf{x}(t)$ is of the form

$$\mathbf{x}(t) = \mathbf{c}_0 C_0^n(t) + \ldots + \mathbf{c}_n C_n^n(t)$$

for some set of basis functions $C_i^n(t)$, then we would have

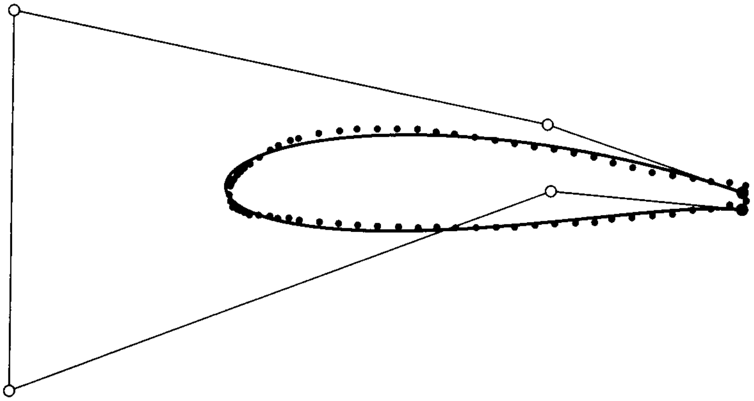


Figure 7.7 Least squares approximation: data points are sampled from the cross section of an airplane wing and a quinitic Bézier curve is fitted to them.

$$\begin{aligned} {}_0C_0^n(t_0) + \ldots + \mathbf{c}_nC_n^n(t_0) &= \mathbf{p}_0 \\ &\vdots \\ \mathbf{c}_0C_0^n(t_P) + \ldots + \mathbf{c}_nC_n^n(t_P) &= \mathbf{p}_P. \end{aligned}$$

This may be condensed into matrix form:

$$\begin{bmatrix} C_0^n(t_0) & \ldots & C_n^n(t_0) \\ & \vdots & \\ & \vdots & \\ C_0^n(t_P) & \ldots & C_n^n(t_P) \end{bmatrix} \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \vdots \\ \mathbf{p}_P \end{bmatrix}.$$

Or, even shorter:

$$M\mathbf{C} = \mathbf{P}. \tag{7.18}$$

Since we assume the number P of data points is larger than the degree n of the curve, this linear system is clearly overdetermined. We attack it by simply multiplying both sides by M^{T}:

$$M^{\mathrm{T}}M\mathbf{C} = M^{\mathrm{T}}\mathbf{P}. \tag{7.19}$$

This is a linear system with $n+1$ equations in $n+1$ unknowns, with a square and symmetric coefficient matrix $M^{\mathrm{T}}M$. Its solution is straightforward, provided that $M^{\mathrm{T}}M$ is always invertible. This is in fact the case: since the C_i^n are assumed to be linearly independent, the $n+1$ columns of M are linearly independent, thus ensuring full rank of $M^{\mathrm{T}}M$.

But is the solution meaningful? After all, we employed a rather crude trick in going from (7.18) to (7.19). It turns out that our solution is not only meaningful—in fact it is *optimal*.

In order to justify this claim (and also making it more precise), we give a second derivation of (7.19).

Consider the following expression:

$$f(\mathbf{c}_0, \ldots, \mathbf{c}_n) = \sum_{i=0}^{P} \|\mathbf{p}_i - \mathbf{x}(t_i)\|^2. \tag{7.20}$$

If the curve $\mathbf{x}(t)$ does not deviate from the data points $\mathbf{p}_i$ by much, then f will attain a small value. Ideally, if $\mathbf{x}$ were to pass through all $\mathbf{p}_i$ exactly, we would have $f = 0$.

If we substitute the full definition of $\mathbf{x}(t)$ into (7.20), we obtain

$$f(\mathbf{c}_0, \ldots, \mathbf{c}_n) = \sum_{i=0}^{P} \left\| \mathbf{p}_i - \sum_{j=0}^{n} \mathbf{c}_j C_j^n(t_i) \right\|^2. \tag{7.21}$$

We wish to find a set of $\mathbf{c}_i$ such that the value of f becomes minimal. Since f is a multivariate function of all components of the $\mathbf{c}_i$, that minimum is achieved if f's partials with respect to all these components vanish:

$$\frac{\partial f}{\partial \mathbf{c}_k^d} = 0; \quad k = 0, \ldots, n; \quad d = 1, 2 \text{ or } d = 1, 2, 3,$$

where the superscript d labels the individual components of the $\mathbf{c}_k$. Computing the required derivatives yields

$$\sum_{i=0}^{P} \left[\mathbf{p}_i^d - \sum_{j=0}^{n} \mathbf{c}_j^d C_j^n(t_i) \right] C_k^n(t_i) = 0; \quad k = 0, \ldots, n; \quad d = 1, 2 \text{ or } d = 1, 2, 3.$$

Upon rearranging, we see that this is identical to (7.19)!

7.9 Smoothing Equations

The solution to the least squares problem aims only at minimizing the error function f in (7.20). It does not "care" about the shape of the resulting curve. It may wiggle more than we would like, yet it is as close to the data points as possible. Sometimes wiggles are undesired, even if closeness of approximation is

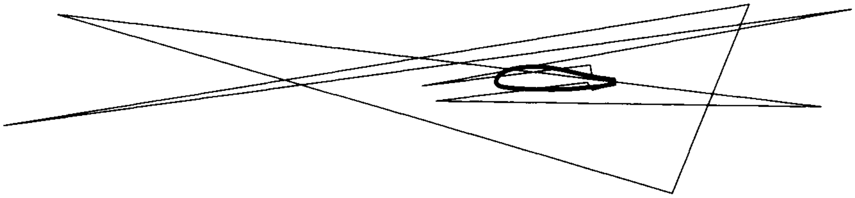


Figure 7.8 Least squares approximation: a degree 13 Bézier curve fitted to the airplane wing data set.

lost. Typically, this is the case for noisy data: if the data points exhibit extraneous wiggles, there is no point in trying to reproduce these. Figure 7.8 shows a "wiggly" control polygon. The next paragraph shows a way to improve the shape of the approximating curve, albeit at the cost of deviating more from the given data points.

We now assume that the basis functions $C_i^n(t)$ are in fact the Bernstein polynomials $B_i^n(t)$ and the unknown coefficients are contained in a vector $\mathbf{B}$. Instead of addressing the shape of the *curve*, we will simply look at the shape of the *control polygon*. Clearly, if the polygon behaves nicely, then so does the curve.

How do we measure the shape of a polygon? Many such measures are conceivable; here, we pick one of the simplest ones. If all second differences $\Delta^2\mathbf{b}_i$ are small, then the polygon does not wiggle much. In addition to (7.18), we might thus aim at also achieving all the following:

$$\begin{aligned} \mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2 \quad &= 0 \\ &\vdots \\ \mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n \quad &= 0. \end{aligned}$$

This we abbreviate as

$$S\mathbf{B} = 0. \tag{7.22}$$

If we simply append these equations to the overdetermined system (7.18), we obtain one that is even more overdetermined:

$$\begin{bmatrix} M \\ S \end{bmatrix} \mathbf{B} = \begin{bmatrix} \mathbf{P} \\ 0 \end{bmatrix}. \tag{7.23}$$

It is solved in the same way as (7.18), that is, by forming the symmetric linear system of normal equations. The system is solvable because the coefficient matrix of (7.23) still has $n + 1$ linearly independent columns, as inherited from the initial matrix M.

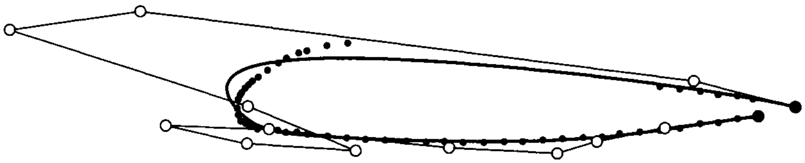


Figure 7.9 Least squares approximation: a degree 13 Bézier curve fitted to an incomplete airplane wing data set. For this example, $\alpha = 0.1$.

In (7.23), we have little control over the effect of the added equations dealing with the shape of the curve. We gain such control by weighting the two different components:

$$\begin{bmatrix} (1-\alpha)M \\ \alpha S \end{bmatrix} \mathrm{B} = \begin{bmatrix} (1-\alpha)\mathrm{P} \\ 0 \end{bmatrix}. \tag{7.24}$$

For $\alpha \in [0, 1)$, this allows a weighting between the initial data fitting equations (7.23) and the shape optimizing equations (7.22). For $\alpha = 0$, we retrieve (7.23); for values of α near 1, we give more weight to the fitting equations. As a rule of thumb, the noisier the data, the larger α. If no information about noise is available, $\alpha = 0.1$ works well.

The effect of these shape equations is shown in Figure 7.9: the airplane wing data set was artificially decimated, resulting in a least squares solution that looked worse than Figure 7.8. Employing shape equations with $\alpha = 0.1$ results in the shown solution.

7.10 Designing with Bézier Curves

According to Bézier, designers at Rénault were quickly getting used to manipulating control points of a curve in order to create a particular shape. Other designers may not like the concept of control points[6] and would prefer to manipulate points on the curve directly.

In that context, a designer would "grab" a point on a curve and designate a new location for it, intending that the new curve should pass through the changed location. We now describe one way of doing this, following ideas from Bartels and Forsey [48].

Suppose we are given a Bézier curve

[6] Such as the designers at Mercedes-Benz, where I worked in the 1980s.

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t).$$

We would like to change a point $\mathbf{x}(\hat{t})$ to a new location $\mathbf{y}$. How do we have to change the $\mathbf{b}_i$ such that the curve passes through $\mathbf{y}$?

We need to find new control points $\hat{\mathbf{b}}_i$ such that

$$\mathbf{y} = \sum_{i=0}^{n} \hat{\mathbf{b}}_i B_i^n(\hat{t}),$$

which can be viewed as one equation in $n-1$ unknowns, assuming that $\mathbf{b}_0$ and $\mathbf{b}_n$ should stay the same. This is an underdetermined linear system (for $n > 2$) with infinitely many solutions. We would like the one that is the closest to the original control points.

We write our linear system as

$$[B_1^n(\hat{t}) \ldots B_{n-1}^n(\hat{t})] \begin{bmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{n-1} \end{bmatrix} = \mathbf{y} - \mathbf{b}_0 B_0^n(\hat{t}) - \mathbf{b}_n B_n^n(\hat{t}) = \mathbf{z} \tag{7.25}$$

and shorten this to

$$A\hat{\mathbf{B}} = \mathbf{z} \tag{7.26}$$

with $A = [B_1^n(\hat{t}) \ldots B_{n-1}^n(\hat{t})]$. We now use a little trick and write our unknowns as

$$\hat{\mathbf{B}} = \mathbf{B} + A^{\mathrm{T}}\mathbf{c} \tag{7.27}$$

with an unknown vector $\mathbf{c}$. The column vector $\mathbf{B}$ contains the current control points $\mathbf{b}_1$ through $\mathbf{b}_{n-1}$. This may be viewed as an overdetermined system for $\mathbf{c}$, which may be solved using a least squares approach:

$$A\hat{\mathbf{B}} = A\mathbf{B} + AA^{\mathrm{T}}\mathbf{c}.$$

Since $A\hat{\mathbf{B}} = \mathbf{z}$, this simplifies to

$$\mathbf{z} - A\mathbf{B} = AA^{\mathrm{T}}\mathbf{c}.$$

We observe that AA^{T} is a scalar and denote it by a. Thus[7]

[7] Keep in mind that $A\mathbf{B} = \mathbf{x} - \mathbf{b}_0 B_0^n(\hat{t}) - \mathbf{b}_n B_n^n(\hat{t})$.

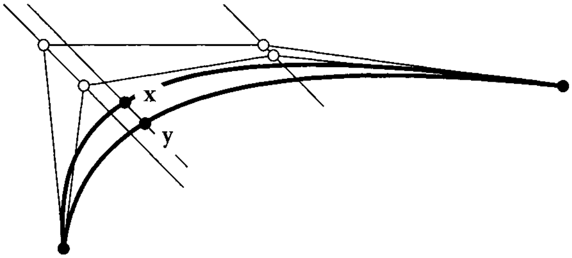


Figure 7.10 Modifying Bézier curves: the location of a point on the curve is changed.

$$\mathbf{c} = \frac{1}{a}(\mathbf{y} - \mathbf{x})$$

and finally

$$\hat{\mathbf{B}} = \mathbf{B} + \frac{1}{a}A^{\mathrm{T}}(\mathbf{y} - \mathbf{x}).$$

All Bézier points $\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$ move in the same direction $\mathbf{y} - \mathbf{x}$, but with different displacement magnitudes. Figure 7.10 gives an example.

7.11 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 deal only 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 $t_{i+1} - t_i = h$, that is, at equally spaced parameter intervals. Then it can be shown that this polynomial may be written as

$$\mathbf{p}(t) = \mathbf{p}_0 + \frac{1}{h}(t - t_0)\Delta\mathbf{p}_0 + \frac{1}{2!h^2}(t - t_0)(t - t_1)\Delta^2\mathbf{p}_0$$
$$+ \frac{1}{3!h^3}(t - t_0)(t - t_1)(t - t_2)\Delta^3\mathbf{p}_0. \qquad (7.28)$$

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

The differences $\Delta^i \mathbf{p}_j$ are defined as

$$\Delta^i \mathbf{p}_j = \Delta^{i-1}\mathbf{p}_{j+1} - \Delta^{i-1}\mathbf{p}_j \tag{7.29}$$

and $\Delta^0 \mathbf{p}_j = \mathbf{p}_j$.

The coefficients in (7.28) 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 (7.28) there. Since our evaluation points t_j are equally spaced, a much faster way exists. Suppose we had computed $\mathbf{p}_j = \mathbf{p}(t_j)$, and so on, from (7.28). Then we could compute all entries in the following table:

$$\begin{array}{llll}
\mathbf{p}_0 & & & \\
\mathbf{p}_1 & g\Delta\mathbf{p}_0 & & \\
\mathbf{p}_2 & g\Delta\mathbf{p}_1 & g^2\Delta^2\mathbf{p}_0 & \\
\mathbf{p}_3 & g\Delta\mathbf{p}_2 & g^2\Delta^2\mathbf{p}_1 & g^3\Delta^3\mathbf{p}_0\,. \\
\mathbf{p}_4 & g\Delta\mathbf{p}_3 & g^2\Delta^2\mathbf{p}_2 & g^3\Delta^3\mathbf{p}_1 \\
\mathbf{p}_5 & g\Delta\mathbf{p}_4 & g^2\Delta^2\mathbf{p}_3 & g^3\Delta^3\mathbf{p}_2 \\
\vdots & \vdots & \vdots & \vdots
\end{array} \tag{7.30}$$

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 (7.28) 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 (7.30) from *right to left:* instead of computing the entry $\mathbf{p}_4$ from (7.28), first compute $g^2\Delta^2\mathbf{p}_2$ from (7.29):

$$g^2\Delta^2\mathbf{p}_2 = g^3\Delta^3\mathbf{p}_1 + g^2\Delta^2\mathbf{p}_1,$$

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 $\mathbf{q}_j^i = g^i\Delta^i\mathbf{p}_j$:

$$\mathbf{q}_j^i = \mathbf{q}_{j-1}^{i+1} + \mathbf{q}_{j-1}^i; \quad i = 2, 1, 0. \tag{7.31}$$

It yields the points $\mathbf{p}_j = \mathbf{q}_j^0$.[8]

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 [389] and by Chang, Shantz, and Rochetti [109].

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, Beatty, and Barsky [47], or Shantz and Chang [571].

7.12 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.
```

[8] It holds for any degree n if we replace $i = 2, 1, 0$ by $i = n-1, n-2, \ldots, 0$.

```
          coeff:  array with coordinates to be interpolated.
          t:      parameter value.
Output:   coordinate value.

        Note: we  assume a uniform knot sequence!
*/
```

7.13 Problems

1 Show that the cubic and quintic Hermite polynomials are linearly independent.

2 Generalize Hermite interpolation to degrees 7, 9, and so on.

*3 The de Casteljau algorithm for Bézier curves has as its "counterpart" the recursion formula (5.2) for Bernstein polynomials. Deduce a recursion formula for Lagrange polynomials from Aitken's algorithm.

*4 The de Casteljau algorithm may be generalized to yield the concept of blossoms. This is not possible for Aitken's algorithm. Why?

*5 The Hermite form is not invariant under affine domain transformations, whereas 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?

*6 When $P = n$ in least squares approximation, we should be back in an interpolation context. Show that this is indeed the case.

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 [205]). 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.

This Page Intentionally Left Blank

B-Spline Curves

8

B-splines were investigated as early as the nineteenth century by N. Lobachevsky (see Renyi [506], p. 165); they were constructed as convolutions of certain probability distributions.[1] In 1946, I. J. Schoenberg [542] 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 [137], M. Cox [129], and L. Mansfield was one of the most important developments in this theory. The recurrence relations were first used by Gordon and Riesenfeld [284] 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 and numerically unstable; see de Boor [138] or Schumaker [546]. A different approach to B-splines was taken by de Boor and Höllig [143]; 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 *blossoming* method proposed by L. Ramshaw [498] and, in a different form, by P. de Casteljau [147]. More literature on blossoms: Gallier [252], Boehm and Prautzsch [87].

[1] However, those were only defined over a very special knot sequence.

8.1 Motivation

B-spline curves consist of many polynomial pieces, offering much more versatility than do Bézier curves. Many B-spline curve properties can be understood by considering just one polynomial piece—that is how we start this chapter.

The Bézier points of a quadratic Bézier curve may be written as blossom values

$$\mathbf{b}[0,0], \mathbf{b}[0,1], \mathbf{b}[1,1].$$

Based on this, we could get the de Casteljau algorithm by repeated use of the identity $t = (1-t)\cdot 0 + t\cdot 1$. The pairs $[0,0], [0,1], [1,1]$ may be viewed as being obtained from the sequence $0, 0, 1, 1$ by taking successive pairs.

Let us now generalize the sequence $0, 0, 1, 1$ to a sequence u_0, u_1, u_2, u_3. The quadratic blossom $\mathbf{b}[u,u]$ may be written as

$$\begin{aligned}
\mathbf{b}[u,u] &= \frac{u_2-u}{u_2-u_1}\mathbf{b}[u_1,u] + \frac{u-u_1}{u_2-u_1}\mathbf{b}[u,u_2] \\
&= \frac{u_2-u}{u_2-u_1}\left(\frac{u_2-u}{u_2-u_0}\mathbf{b}[u_0,u_1] + \frac{u-u_0}{u_2-u_0}\mathbf{b}[u_1,u_2]\right) \\
&+ \frac{u-u_1}{u_2-u_1}\left(\frac{u_3-u}{u_3-u_1}\mathbf{b}[u_1,u_2] + \frac{u-u_1}{u_3-u_1}\mathbf{b}[u_2,u_3]\right).
\end{aligned}$$

This uses the identity

$$u = \frac{u_2-u}{u_2-u_1}u_1 + \frac{u-u_1}{u_2-u_1}u_2$$

for the first step and the two identities

$$u = \frac{u_2-u}{u_2-u_0}u_0 + \frac{u-u_0}{u_2-u_0}u_2$$

and

$$u = \frac{u_3-u}{u_3-u_1}u_1 + \frac{u-u_1}{u_3-u_1}u_3$$

for the second step. Note that we successively express u in terms of intervals of growing size.

Starting with the $\mathbf{b}[u_i, u_{i+1}]$ as input control points, we may rewrite this as:

$$\begin{array}{lll}
\mathbf{b}[u_0, u_1] & & \\
\mathbf{b}[u_1, u_2] & \mathbf{b}[u_1, u] & \\
\mathbf{b}[u_2, u_3] & \mathbf{b}[u, u_2] & \mathbf{b}[u, u].
\end{array}$$

This is a first instance of the de Boor generalization of the de Casteljau algorithm. See Example 8.1 for a detailed computation.

Figure 8.1 illustrates the algorithm, but using the knot sequence $u_0, u_1, u_2, u_3 = 0, 1, 3, 4$ and $u = 2.0$.

Example 8.1 **The de Boor algorithm for $n = 2$.**

Let $u_0, u_1, u_2, u_3 = 0, 2, 4, 6$. Let the control points be given by

$$\mathbf{b}[u_0, u_1] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{b}[u_1, u_2] = \begin{bmatrix} 8 \\ 8 \end{bmatrix}, \quad \mathbf{b}[u_2, u_3] = \begin{bmatrix} 8 \\ 0 \end{bmatrix}.$$

Setting $u = 3$, we now compute the point $\mathbf{b}[3, 3]$. At the first level, we compute two points

$$\mathbf{b}[2, 3] = \frac{4-3}{4-0}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{3-0}{4-0}\begin{bmatrix} 8 \\ 8 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

and

$$\mathbf{b}[3, 4] = \frac{6-3}{6-2}\begin{bmatrix} 8 \\ 8 \end{bmatrix} + \frac{3-2}{6-2}\begin{bmatrix} 8 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}.$$

Finally,

$$\mathbf{b}[3, 3] = \frac{4-3}{4-2}\begin{bmatrix} 6 \\ 6 \end{bmatrix} + \frac{3-2}{4-2}\begin{bmatrix} 8 \\ 6 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}.$$

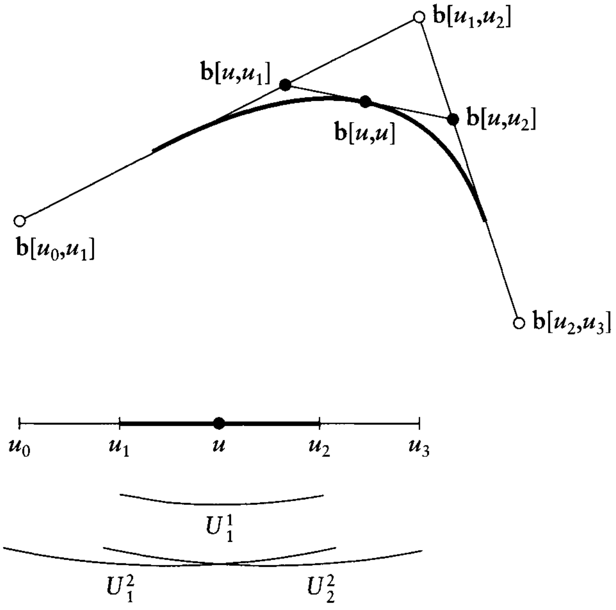


Figure 8.1 The de Boor algorithm: the quadratic case.

8.2 B-Spline Segments

B-spline curves consist of a sequence of polynomial curve segments. In this section, we focus on just one of them.

Let U be an interval $[u_I, u_{I+1}]$ in a sequence $\{u_i\}$ of knots. We define ordered sets U_i^r of successive knots, each containing u_I or u_{I+1}. The set U_i^r is defined such that:

U_i^r consists of $r + 1$ successive knots.

u_I is the $(r - i)$th element of U_i^r, with $i = 0$ denoting the first of U_i^r's elements.

We also observe

$$U_i^r = U_i^{r+1} \cap U_{i+1}^{r+1}.$$

When the context is unambiguous, we also refer to the U_i^r as *intervals*, having the first and last elements of U_i^r as endpoints. In that context, $U_1^1 = U$. We also define $U = [U_0^0, U_1^0]$ and use the term *interval* only if $U_0^0 \neq U_1^0$.

A degree n curve segment corresponding to the interval U is given by $n + 1$ control points $\mathbf{d}_i$ which are defined by

$$\mathbf{d}_i = \mathbf{b}[U_i^{n-1}]; \quad i = 0, \ldots, n. \tag{8.1}$$

The point $\mathbf{x}(u) = \mathbf{b}[u^{<n>}]$ on the curve is recursively computed as

$$\mathbf{d}_i^r(u) = \mathbf{b}[u^{<r>}, U_i^{n-1-r}]; \quad r = 1, \ldots, n; i = 0, \ldots, n-r \tag{8.2}$$

with $\mathbf{x}(u) = \mathbf{d}_0^n(u)$.[2] This is known as the *de Boor algorithm* after Carl de Boor see [137]. See Example 8.2 for the case $n = 3$ and Figure 8.2 for an illustration.

Equation (8.2) may alternatively be written as

$$\mathbf{d}_i^r(u) = (1 - t_{i+1}^{n-r+1})\mathbf{d}_i^{r-1} + t_{i+1}^{n-r+1}\mathbf{d}_{i+1}^{r-1}; \quad r = 1, \ldots, n; i = 0, \ldots, n-r, \tag{8.3}$$

where t_{i+1}^{n-r+1} is the local parameter in the interval U_{i+1}^{n-r+1}.

A geometric interpretation is as follows. Each intermediate control polygon leg $\mathbf{d}_i^r, \mathbf{d}_{i+1}^r$ may be viewed as an affine image of U_{i+1}^{n-r+1}. The point $\mathbf{d}_i^{r+1}$ is then the image of u under that affine map.

For the special knot sequence $0^{<n>}, 1^{<n>}$ and $U = [0, 1]$, the de Boor algorithm becomes

$$\mathbf{d}_i^r(u) = \mathbf{b}[u^{<r>}, 0^{<n-r-i>}, 1^{<i>}]; \quad r = 1, \ldots, n; \quad i = 0, \ldots, n-r, \tag{8.4}$$

which is simply the de Casteljau algorithm.

If the parameter u happens to be one of the knots, the algorithm proceeds as before, except that we do not need as many levels of the algorithm. For example, if a quadratic curve segment is defined by $\mathbf{b}[u_0, u_1], \mathbf{b}[u_1, u_2], \mathbf{b}[u_2, u_3]$ and we want to evaluate at $u = u_2$, then two of the intermediate points in the de Boor algorithm are already known, namely, $\mathbf{b}[u_1, u_2]$ and $\mathbf{b}[u_2, u_3]$. From these two, we immediately calculate the desired point $\mathbf{b}[u_2, u_2]$, thus the de Boor algorithm now needs only one level instead of two.

Derivatives of a B-spline curve segment are computed in analogy to the Bézier curve case (5.17)

$$\dot{\mathbf{x}}(u) = n\mathbf{b}[u^{<n-1>}, \vec{1}]. \tag{8.5}$$

Expanding this expression and using the control point notation, this becomes

$$\dot{\mathbf{x}}(u) = \frac{n}{|U|}(\mathbf{d}_1^{n-1} - \mathbf{d}_0^{n-1}), \tag{8.6}$$

[2] This notation is different from the one used in previous editions of this book.

Example 8.2 **The de Boor algorithm for $n = 3$.**

Let part of a knot sequence be given by

$$\ldots u_3, u_4, u_5, u_6, u_7, u_8, \ldots$$

and let $U = [u_5, u_6]$. The standard blossom computation of $\mathbf{b}[u, u, u]$ proceeds as follows:

$$\begin{array}{llll}
\mathbf{b}[u_3, u_4, u_5] & & & \\
\mathbf{b}[u_4, u_5, u_6] & \mathbf{b}[u, u_4, u_5] & & \\
\mathbf{b}[u_5, u_6, u_7] & \mathbf{b}[u, u_5, u_6] & \mathbf{b}[u, u, u_5] & \\
\mathbf{b}[u_6, u_7, u_8] & \mathbf{b}[u, u_6, u_7] & \mathbf{b}[u, u, u_6] & \mathbf{b}[u, u, u].
\end{array}$$

We now write this as

$$\begin{array}{llll}
\mathbf{b}[U_0^2] & & & \\
\mathbf{b}[U_1^2] & \mathbf{b}[u, U_0^1] & & \\
\mathbf{b}[U_2^2] & \mathbf{b}[u, U_1^1] & \mathbf{b}[u, u, U_0^0] & \\
\mathbf{b}[U_3^2] & \mathbf{b}[u, U_2^1] & \mathbf{b}[u, u, U_1^0] & \mathbf{b}[u, u, u].
\end{array}$$

In terms of control points:

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

The labeling in the first scheme depends on the subscripts of the knots, whereas the last two employ a relative numbering.

---

where $|U| = U_1^0 - U_0^0$ denotes the length of the interval U. Thus the last two intermediate points $\mathbf{d}_0^{n-1}$ and $\mathbf{d}_1^{n-1}$ span the curve's tangent, in complete analogy to the de Casteljau algorithm.

Higher derivatives follow the same pattern:

$$\frac{\mathrm{d}^r}{\mathrm{d}u^r}\mathbf{x}(u) = \frac{n!}{(n-r)!}\mathbf{b}[u^{<n-r>}, \vec{1}^{<r>}]. \tag{8.7}$$

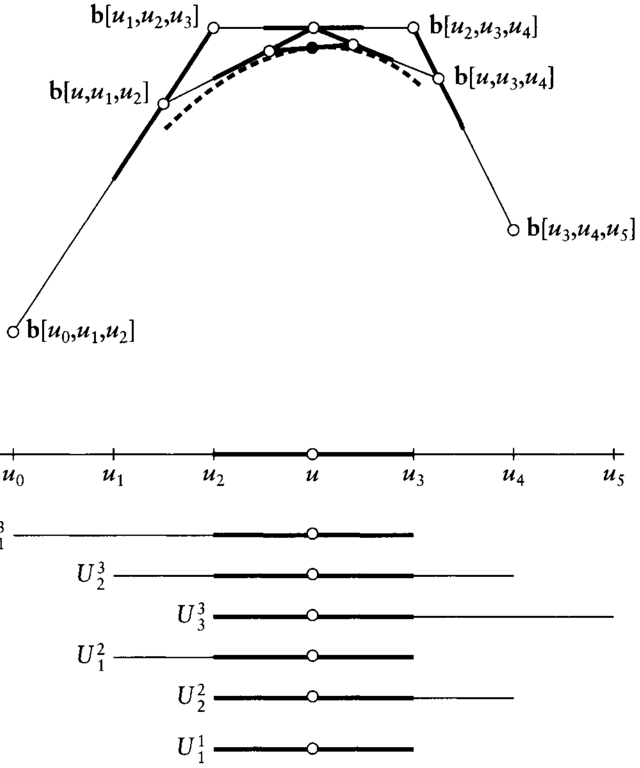


Figure 8.2 The de Boor algorithm: a cubic example. The solid point is the result $\mathbf{b}[u,u,u]$; it is on the line through $\mathbf{b}[u,u,u_2]$ and $\mathbf{b}[u,u,u_3]$.

In the case of Bézier curves, we could use the de Casteljau algorithm for curve evaluation, but we could also write a Bézier curve explicitly using Bernstein polynomials. Since we changed the domain geometry, we will now obtain a different explicit representation, using polynomials[3] P_i^n:

$$\mathbf{x}(u) = \sum_{i=0}^{n} \mathbf{d}_i P_i^n(u). \tag{8.8}$$

The polynomials P_i^n satisfy a recursion similar to the one for Bernstein polynomials, and the following derivation is very similar to that case:

[3] These will later become building blocks of B-splines.

$$
\begin{aligned}
\mathbf{x}(u) &= \sum_{i=0}^{n-1} \mathbf{d}_i^1 P_i^{n-1}(u) \\
&= \sum_{i=0}^{n-1} (1 - t_{i+1}^n)\mathbf{d}_i P_i^{n-1}(u) + \sum_{i=0}^{n-1} t_{i+1}^n \mathbf{d}_{i+1} P_i^{n-1}(u) \qquad (8.9) \\
&= \sum_{i=0}^{n} (1 - t_{i+1}^n)\mathbf{d}_i P_i^{n-1}(u) + \sum_{i=1}^{n} t_i^n \mathbf{d}_i P_{i-1}^{n-1}(u)
\end{aligned}
$$

For the first step, we used the de Boor algorithm, letting t_i^n be the local parameter in the interval U_{i+1}^n. For the second step, we used the convention $P_n^{n-1} \equiv 0$ to modify the first term. Using a similar argument: $P_{-1}^{n-1} \equiv 0$, we may extend the second term to start with $i = 0$. We conclude

$$P_i^n(u) = (1 - t_{i+1}^n)P_i^{n-1}(u) + t_i^n P_{i-1}^{n-1}(u). \qquad (8.10)$$

This recursion has to be anchored in order to be useful. This is straightforward for the case $n = 1$:

$$P_0^1(u) = \frac{u_r - U_1^0}{|U|}, \quad P_1^1(u) = \frac{u - U_0^0}{|U|},$$

where $|U| = U_1^0 - U_0^0$. For the special knot sequence $0^{<n>}, 1^{<n>}$ and $U = [0, 1]$, this is the Bernstein recursion.

8.3 B-Spline Curves

A B-spline curve consists of several polynomial curve segments, each of which may be evaluated using a de Boor algorithm. A B-spline curve is defined by

the degree n of each curve segment,

the knot sequence $u_0, \ldots, u_K$, consisting of $K + 1$ knots $u_i \leq u_{i+1}$,

the control polygon $\mathbf{d}_0, \ldots, \mathbf{d}_L$ with $L = K - n + 1$.

Some comments: the numbering of the control points $\mathbf{d}_i$ in this definition is *global*, whereas in Section 8.2 it was *local* relative to an interval U. Each $\mathbf{d}_i$ may be written as a blossom value with n subsequent knots as arguments. Hence the number $L + 1$ of control points equals the number of n-tuples in the knot sequence.

Example 8.3 **Some examples of B-spline curve definitions.**

Let $n = 1$, and let the knot sequence be $0, 1, 2$, hence $K = 2$. There will be control points $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_3$. The curve's domain is $[u_0, u_3]$, and there are two linear curve segments.

Let $n = 2$ with the knot sequence $0, 0.2, 0.4, 0.5, 0.7, 1$, hence $K = 5$. There will be control points $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4$ and three quadratic curve segments. If we now change the knot sequence to $0, 0.2, 0.45, 0.45, 0.7, 1$, then the number of curve segments will drop to two.

Each knot may be repeated in the knot sequence up to n times. In some cases it is approriate to simply list those knots multiple times. For other applications, it is better to list the knot only once and record its *multiplicity* in an integer array. For example, the knot sequence $0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 4.0, 4.0$ could be stored as $0.0, 1.0, 2.0, 3.0, 4.0$ and a multiplicity array $2, 1, 1, 2, 2$.

There is a different de Boor algorithm for each curve segment. Each is "started" with a set of U_i^n, that is, by sequences of $n + 1$ knots. In order for local coordinates to be defined in (8.3), no successive $n + 1$ knots may coincide.

In Section 8.2, we assumed that we could find the requisite U_i^n for each interval U. This is possible only if U is "in the middle" of the knot sequence; more precisely, the first possible de Boor algorithm is defined for $U = [u_{n-1}, u_n]$ and the last one is defined for $U = [u_{K-n}, u_{K-n+1}] = [u_{K-n}, u_L]$. We thus call $[u_{n-1}, u_L]$ the *domain* of the B-spline curve. A B-spline curve has as many curve segments as there are nonzero intervals U in the domain. Example 8.3 illustrates these comments.

For more examples of B-spline curves, see Figure 8.3.

Since a B-spline curve consists of a number of polynomial segments, one might ask for the Bézier form of these segments. For a segment $U = [u_I, u_{I+1}]$ of the curve, we simply evaluate its blossom $\mathbf{b}^U$ and obtain the Bézier points $\mathbf{b}_0^U, \ldots, \mathbf{b}_n^U$ as

$$\mathbf{b}_k^U = \mathbf{b}^U[u_I^{<n-k>}, u_{I+1}^{<k>}].$$

An example is shown in Figure 8.4. Several constituent curve pieces of the same curve are shown in Figure 8.5.

When dealing with B-spline curves, it is convenient to treat it as one curve, not just as a collection of polynomial segments. A point on such a curve is denoted by $\mathbf{d}(u)$, with $u \in [u_{n-1}, u_{K-n+1}]$. In order to evaluate, we perform the following steps:

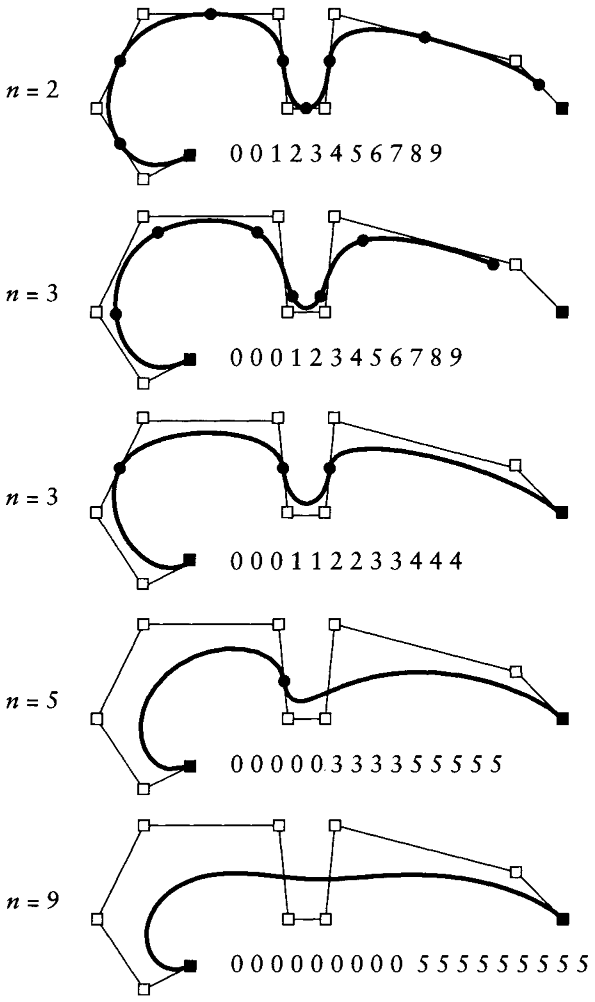


Figure 8.3 B-spline curves: several examples.

1. Find the interval $U = [u_I, u_{I+1})$ that contains u.
2. Find the $n + 1$ control points that are relevant for the interval U. They are, using the global numbering, given by $\mathbf{d}_{I-n+1}, \ldots, \mathbf{d}_{I+1}$.
3. Renumber them as $\mathbf{d}_0, \ldots, \mathbf{d}_n$ and evaluate using the de Boor algorithm (8.3).

In terms of the global numbering of knots, we observe that the intervals U_i^k from the previous section are given by

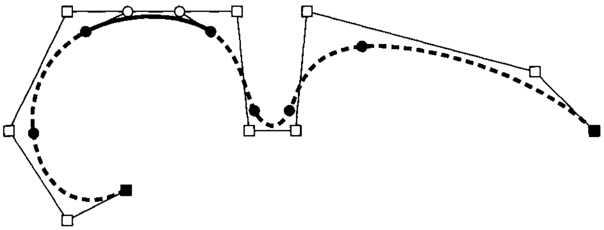


Figure 8.4 Conversion to Bézier form: the Bézier points of a segment of a cubic B-spline curve.

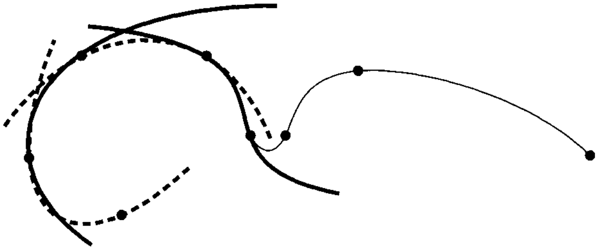


Figure 8.5 Individual curve segments: the first four cubic segments of a cubic B-spline curve are shown, alternating between dashed and black.

$$U_i^k = [u_{I-k+i}, u_{I+i}].$$

The steps in the de Boor algorithm then become

$$\mathbf{d}_i^k(u) = (1 - \alpha_i^k)\mathbf{d}_i^{k-1}(u) + \alpha_i^k \mathbf{d}_{i+1}^{k-1}(u)$$

with

$$\alpha_i^k = \frac{u - u_{I-k+i}}{u_{I+i} - u_{I\_k+i}}$$

for $k = r+1, \ldots, n$, and $i = 0, \ldots, n-k$. Here, r denotes the multiplicity of u. (Normally, u is not already in the knot sequence; then, $r = 0$.)

The fact that each curve segment is only affected by $n+1$ control points is called the *local control property.*

We also use the notion of a B-spline blossom $\mathbf{d}[v_1, \ldots, v_n]$, keeping in mind that each domain interval U has its own blossom $\mathbf{b}^U$ and that consequently $\mathbf{d}[v_1, \ldots, v_n]$ is piecewise defined.

B-spline curves enjoy all properties of Bézier curves, such as affine invariance, variation diminution, etc. Some of these properties are more pronounced now because of the local control property. Take the example of the convex hull

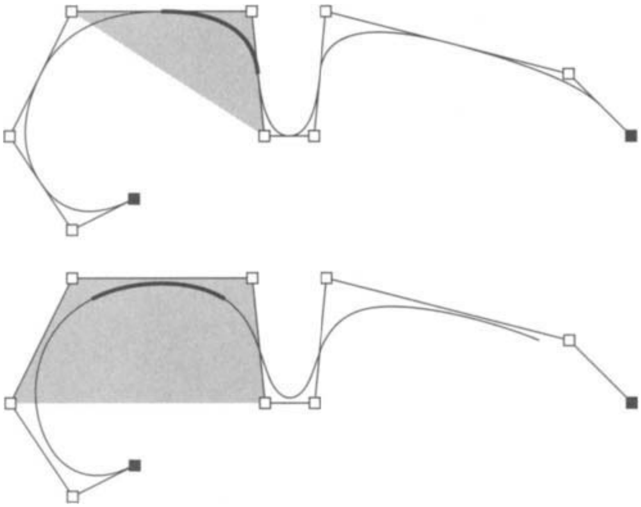


Figure 8.6 The local convex hull property: top, quadratic; bottom, cubic.

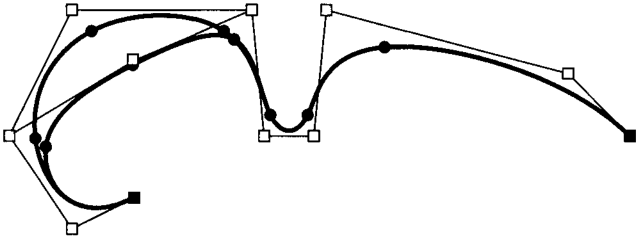


Figure 8.7 The local control property: changing one control point of a cubic B-spline curve has only a local effect.

property: the curve is in the convex hull of its control polygon, but also each segment is in the convex hull of its defining $n + 1$ control points; see Figure 8.6.

A consequence of the local control property is that changing one control point will only affect the curve locally. This is illustrated in Figure 8.7.

8.4 Knot Insertion

Consider a B-spline curve segment defined over an interval U. It is defined by all blossom values $\mathbf{d}[U_i^{n-1}]$; $i = 0, \ldots, n$ where each n-tuple of successive knots U_i^{n-1} contains at least one of the endpoints of U. If we now split U into two segments by inserting a new knot $\hat{u}$, the curve will have two corresponding

Example 8.4 **Knot insertion.**

In Figure 8.8, just one de Boor step is carried out for the parameter value $\hat{u}$. The two new resulting points, in blossom notation, are $\mathbf{b}[u_1, \hat{u}]$ and $\mathbf{b}[\hat{u}, u_2]$. Let us now consider the points

$$\mathbf{b}[u_0, u_1], \mathbf{b}[u_1, \hat{u}], \mathbf{b}[\hat{u}, u_2].$$

These are the B-spline control points of our curve $\mathbf{b}[u, u]$ for the interval $[u_1, \hat{u}]$! Similarly, the B-spline control points for the interval $[\hat{u}, u_2]$ are given by

$$\mathbf{b}[u_1, \hat{u}], \mathbf{b}[\hat{u}, u_2], \mathbf{b}[u_2, u_3].$$

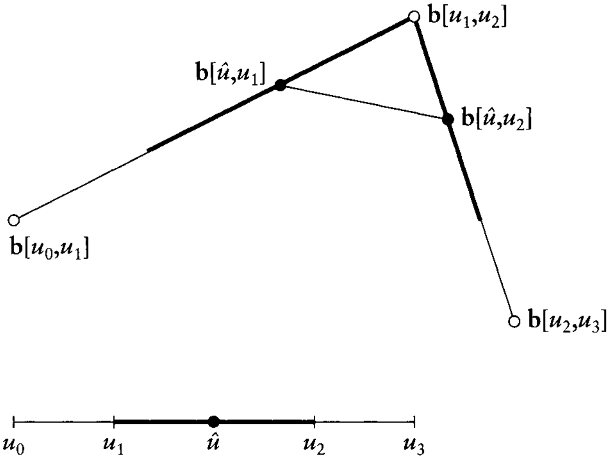


Figure 8.8 Knot insertion: a quadratic example.

segments. What are the control points for these two segments? The answer is surprisingly simple: all blossom values $\mathbf{d}[\hat{U}_i^{n-1}]$; $i = 0, \ldots, n+1$ where each n-tuple of successive knots $\hat{U}_i^{n-i}$ contains at least one of the endpoints of U. This result is due to W. Boehm [68], although it was not originally derived using blossoms. See Example 8.4.

Knot insertion works since B-spline control points are nothing but blossom values of successive knots—now they involve the new knot $\hat{u}$. We may also view the process of knot insertion as one level of the de Boor algorithm, as illustrated in Example 8.4.

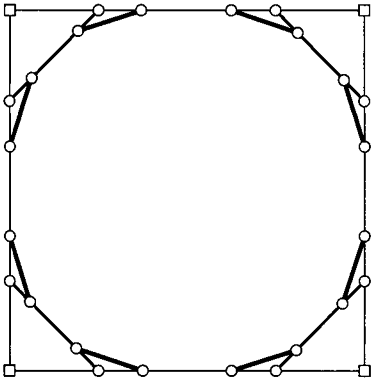


Figure 8.9 Chaikin's algorithm: starting with a (closed) control polygon B-spline curve, two levels of the algorithm are shown.

An interesting application of repeated knot insertion is due to G. Chaikin [105]. 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

$$\mathbf{d}_{2i-1}^{(1)} = \frac{3}{4}\mathbf{d}_i + \frac{1}{4}\mathbf{d}_{i-1} \quad \text{and} \quad \mathbf{d}_{2i}^{(1)} = \frac{3}{4}\mathbf{d}_i + \frac{1}{4}\mathbf{d}_{i+1}.$$

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 8.9 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. 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, and was first proved by de Boor [140]. One may thus use a corner-cutting procedure to *define* a curve—and only very few of the curves thus generated are piecewise polynomial! Recent work has been carried out by Prautzsch and Micchelli [495] and [426], based on earlier results by de Rham [150], [151].

R. Riesenfeld [508] 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 [121]. This so-called Oslo algorithm needs a theory of discrete B-splines for its development (see Bartels, Beatty, and Barsky [47]).

8.5 Degree Elevation

We may degree elevate in (almost) the same way we could degree elevate Bézier curves using (6.2). The difference: a given B-spline is a piecewise degree n curve over a given knot sequence. Its differentiability is determined by the knot multiplicities. If we write it as a piecewise degree $n+1$ curve, we need to increase the multiplicity of every knot by one, thus maintaining the original differentiability properties. For example, if we degree elevate a C^0 piecewise linear curve to piecewise quadratic, it is still C^0. But for a piecewise quadratic to be C^0, it has to have double knots. Let us denote the knots in this augmented knot sequence by $\hat{u}_i$.

Let V^n be a sequence of $n+1$ real numbers $v_1, \ldots, v_{n+1}$. Let $V^n|v_i$ denote the sequence V^n with the value v_i removed. Then the degree $n+1$ blossom $\hat{\mathbf{b}}$ may be expressed in terms of the degree n blossom $\mathbf{b}$ via

$$\hat{\mathbf{b}}[V^{(n+1)}] = \frac{1}{n+1}\left(\mathbf{b}[V^{(n+1)}|v_1] + \ldots + \mathbf{b}[V^{(n+1)}|v_{n+1}]\right). \qquad (8.11)$$

The proof is identical to that for degree elevation of Bézier curves. The control points are then recovered from the blossom as before (see Example 8.5).

The inverse process—*degree reduction* is more important for practical applications. Following the example of the analogous Bézier case, we write the elevation process as a matrix product and invert it by a least squares technique for the reduction process; see Section 6.4. This method is described in detail in [617]. Other methods exist, see [481] and [624].

Example 8.5 B-spline degree elevation and blossoms.

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 $[\hat{u}_7, \hat{u}_8]$. We denote the corresponding blossoms by $\mathbf{d}_4[a, b, c]$ and $\hat{\mathbf{d}}_7[a, b, c, d]$. The new control point $\hat{\mathbf{d}}_4$ is computed as follows:

$$\begin{aligned}\hat{\mathbf{d}}_4 &= \hat{\mathbf{d}}_7[\hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7]\\ &= \frac{1}{4}\left(\mathbf{d}_4[\hat{u}_4, \hat{u}_5, \hat{u}_6] + \mathbf{d}_4[\hat{u}_4, \hat{u}_5, \hat{u}_7] + \mathbf{d}_4[\hat{u}_4, \hat{u}_6, \hat{u}_7] + \mathbf{d}_4[\hat{u}_5, \hat{u}_6, \hat{u}_7]\right)\\ &= \frac{1}{2}\left(\mathbf{d}_4[u_3, u_3, u_4] + \mathbf{d}_4[u_3, u_4, u_4]\right).\end{aligned}$$

For the last step, we have used $\hat{u}_4 = \hat{u}_5 = u_3$ and $\hat{u}_6 = \hat{u}_7 = u_4$.

8.6 Greville Abscissae

Let $l[u] = u$ be the blossom of the (nonparametric) linear function u. If we want to write this linear blossom as a quadratic one: $l^2[u, v] = l[u]$, we easily see that

$$l^{(2)}[u, v] = \frac{1}{2}l[u] + \frac{1}{2}l[v]$$

gives the desired quadratic form of our linear blossom. If we asked for a cubic form $l^{(3)}[u, v, w]$ of $l[u]$, we find that

$$l^{(3)}[u, v, w] = \frac{1}{3}l^2[v, w] + \frac{1}{3}l^2[u, w] + \frac{1}{3}l^2[u, v].$$

If we denote a degree n version of the linear blossom by $l^{(n)}[V^n]$ with $V^n = v_1, \ldots, v_n$, it follows that

$$l^{(n)}[V^n] = \frac{1}{n}(v_1 + \ldots + v_n).$$

The proof is by induction and was anchored by the earlier examples. The inductive step starts with the degree elevation formula (8.11):[4]

$$l^{(n+1)} = \frac{1}{n+1}\sum_{i=1}^{n+1}\frac{1}{n}[V^{(n+1)}|v_i]$$

This is easily transformed to

$$l^{n+1} = \frac{1}{n+1}(v_1 + \ldots + v_{n+1}),$$

thus finishing the proof.

If we are given a knot sequence $u_0, \ldots, u_K$ and a degree n, then we know that any B-spline function $d(u)$ has control vertices $d[u_0, \ldots, u_{n-1}], \ldots, d[u_{K-n+1}, \ldots, u_K]$. In the case of a linear function l, we thus have control vertices

$$\frac{1}{n}(u_0 + \ldots + u_{n-1}), \ldots, \frac{1}{n}(u_{K-n+1} + \ldots + u_n).$$

[4] We do not have to work with augmented knot sequences here since we always deal with *one* linear function.

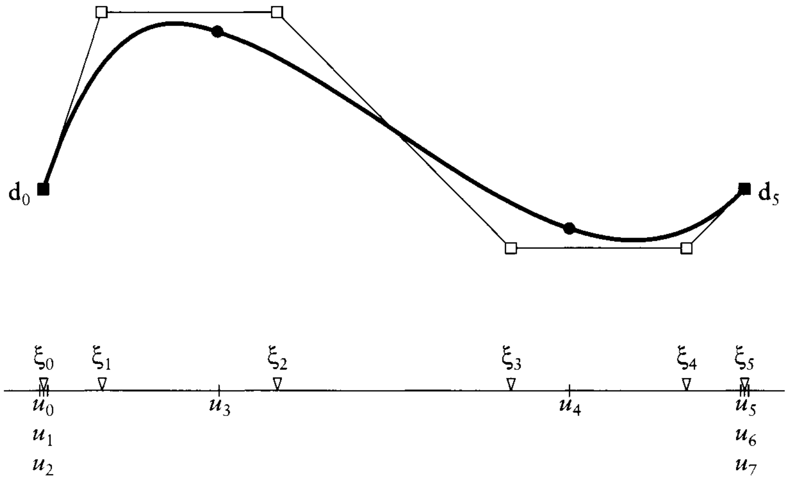


Figure 8.10 Nonparametric B-spline curves: a cubic example.

These terms are called *Greville abscissae* and are abbreviated as

$$\xi_i = \frac{1}{n}(u_i + \ldots + u_{i+n-1}).$$

A *nonparametric* B-spline function $d(u)$ may thus be written as a parametric curve with points

$$\mathbf{d}_i = \begin{bmatrix} \xi_i \\ d_i \end{bmatrix}; \quad i = 0, \ldots, L$$

with the usual $L = K - n + 1$. Figure 8.10 gives an example.

For the special case of the knot sequence $0^{<n>}, 1^{<n>}$, we obtain $\xi_i = \frac{i}{n}$, as already encountered in Section 6.5.

8.7 Smoothness

A B-spline curve consists of several polynomial segments, one for each domain interval U. What is the smoothness of this piecewise curve?

Figures 8.11, 8.12, and 8.13 show how knot multiplicities affect smoothness. In general, if a knot $\hat{u}$ is of multiplicity r, then a B-spline curve of degree n has smoothness C^{n-r} at that knot. This follows from considering the osculants

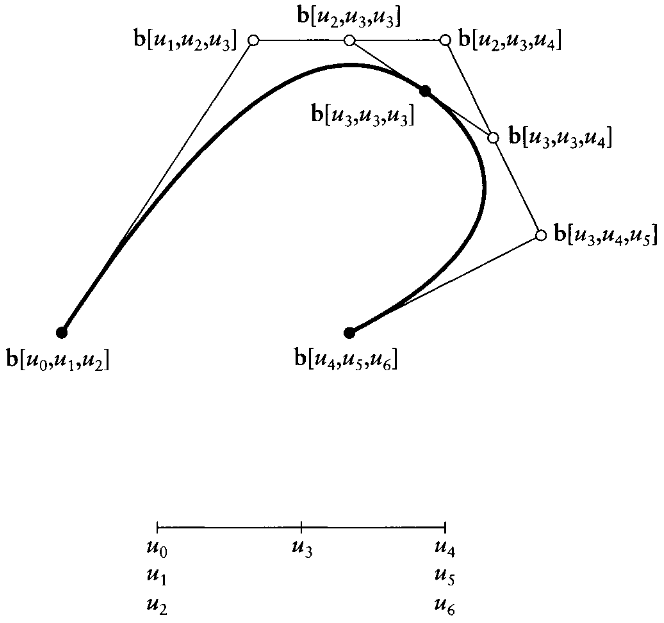

$\mathbf{b}[u_2,u_3,u_3]$
$\mathbf{b}[u_1,u_2,u_3]$
$\mathbf{b}[u_2,u_3,u_4]$
$\mathbf{b}[u_3,u_3,u_3]$
$\mathbf{b}[u_3,u_3,u_4]$
$\mathbf{b}[u_3,u_4,u_5]$
$\mathbf{b}[u_0,u_1,u_2]$
$\mathbf{b}[u_4,u_5,u_6]$
u_0
u_1
u_2
u_3
u_4
u_5
u_6

Figure 8.11 Smoothness: an interior knot of multiplicity one results in a C^2 piecewise cubic curve.

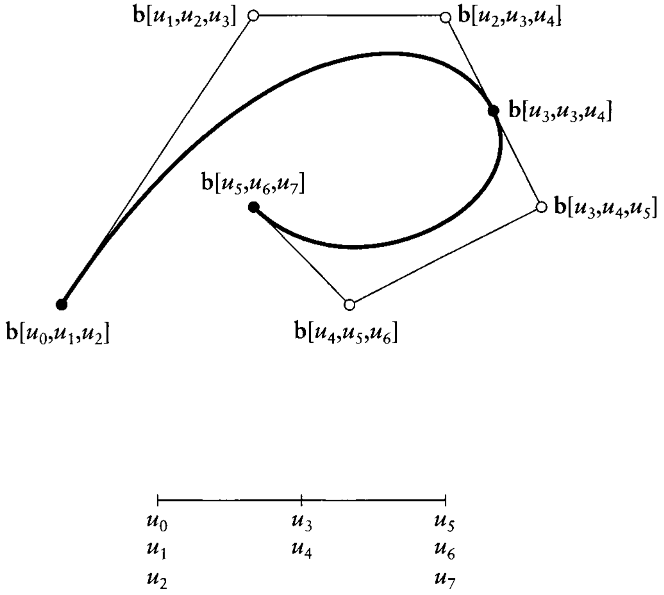

$\mathbf{b}[u_1,u_2,u_3]$
$\mathbf{b}[u_2,u_3,u_4]$
$\mathbf{b}[u_3,u_3,u_4]$
$\mathbf{b}[u_5,u_6,u_7]$
$\mathbf{b}[u_3,u_4,u_5]$
$\mathbf{b}[u_0,u_1,u_2]$
$\mathbf{b}[u_4,u_5,u_6]$
u_0
u_1
u_2
u_3
u_4
u_5
u_6
u_7

Figure 8.12 Smoothness: an interior knot of multiplicity two results in a C^1 piecewise cubic curve.

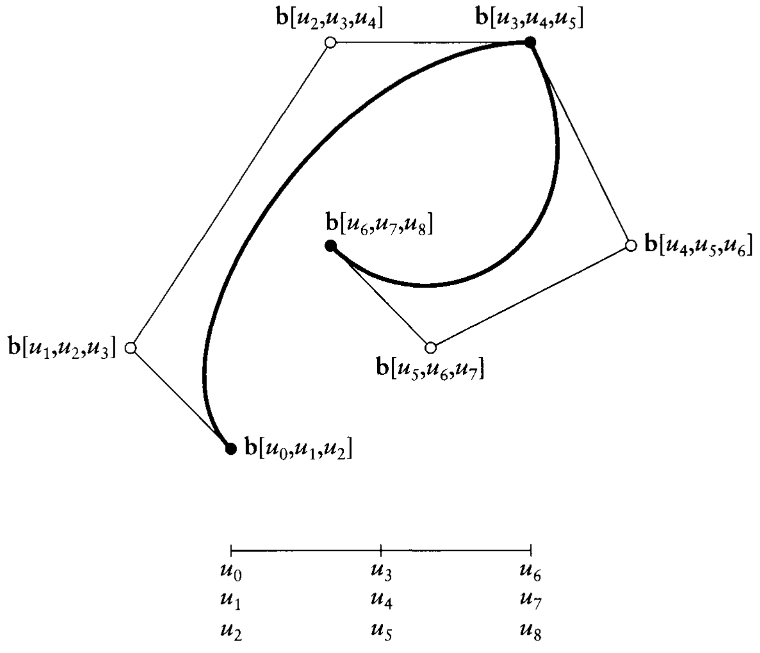


Figure 8.13 Smoothness: an interior knot of multiplicity three results in a C^0 piecewise cubic curve.

at $\hat{u}$.[5] The highest-order osculant is given by

$$o^{n-r}(u) = \mathbf{b}[\hat{u}^{<r>}, u^{<n-r>}],$$

assuring continuity of derivatives up to order $n - r$. Higher-order continuity is possible, but cannot be guaranteed.

An important special case is given by *piecewise Bézier curves*. These are B-spline curves of degree n where each knot is of full multiplicity n. In general, such curves will only be C^0, but under certain conditions, they may be smoother.

For concreteness, take two cubic Bézier curves with control polygons $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ and $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_2$, defined over a knot sequence $u_0, u_0, u_0, u_1, u_1, u_1, u_2, u_2, u_2$. They are C^0 if $\mathbf{b}_3 = \mathbf{c}_0$, or, in terms of the associated blossoms, if $\mathbf{b}[u_1, u_1, u_1] = \mathbf{c}[u_1, u_1, u_1]$. Two such curves are shown in Figure 8.14.

The two curves are C^1 if they may be written as a B-spline curve with a double, not a triple knot u_1. Then our triple knot at u_1 is the result of knot insertion and the three points $\mathbf{b}_2, \mathbf{b}_3, \mathbf{c}_1$ are collinear and in the ratio $\Delta_0 : \Delta_1$ with $\Delta_0 = u_1 - u_0$

[5] The osculant of order r of an n^{th} degree polynomial curve $\mathbf{x}(u)$ at paramter value $\hat{u}$ is the degree r polynomial that agrees with $\mathbf{x}$ for all derivatives up to order r.

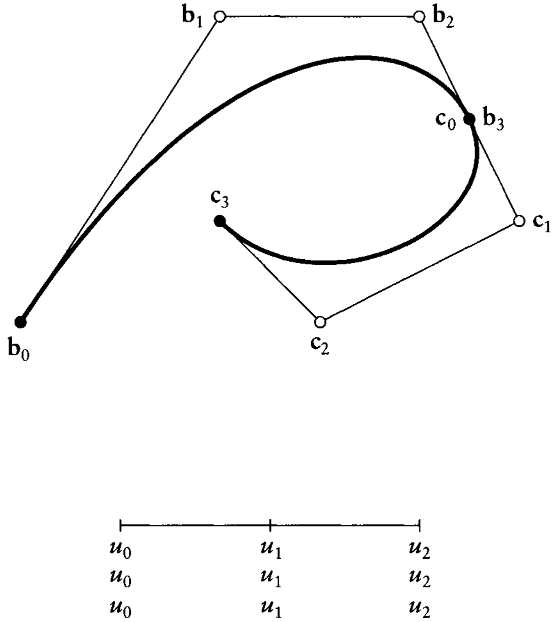


Figure 8.14 Smoothness of Bézier curves: the C^1 case.

and $\Delta_1 = u_2 - u_1$. In terms of blossoms:

$$\mathbf{c}_1 = \mathbf{b}[u_1, u_1, u_2] \quad \text{and} \quad \mathbf{b}_2 = \mathbf{c}[u_0, u_1, u_1].$$

For C^2 smoothness, the knot u_1 must have been the result of *two* knot insertions. It follows that

$$\mathbf{b}[u_0, u_1, u_2] = \mathbf{c}[u_0, u_1, u_2]. \tag{8.12}$$

is our desired C^2 condition. It is illustrated in Figure 8.15.

If we are to check if two given Bézier curves are C^2 or not, all we have to do is construct the two points appearing in (8.12). If they disagree, as in Figure 8.16, we conclude that the given curve is not C^2.

In most practical cases, a C^2 check would have to check for *approximate* satisfaction of (8.12), since reals or floats are rarely equal. In other words, a tolerance has to be used. The practical value of (8.12) lies in the fact that it is amenable to using a point tolerance that determines when two distinct points are to be considered the same point. Checking for C^2 smoothness by comparing second derivatives would require a different, and less intuitive, tolerance.

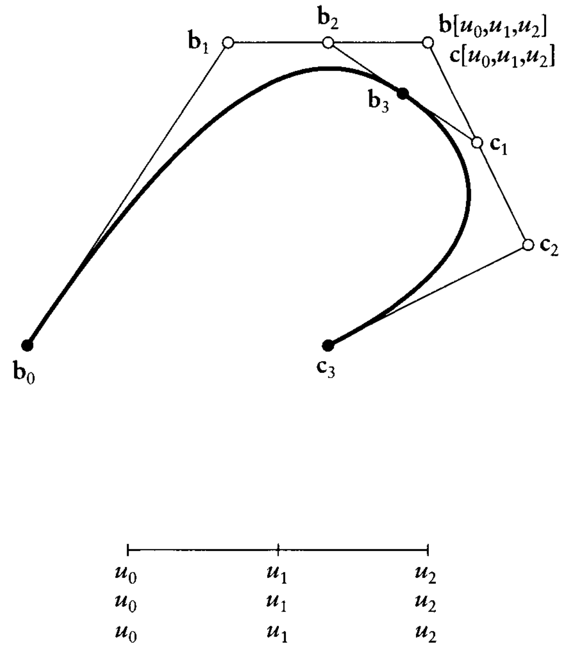


Figure 8.15 Smoothness of Bézier curves: the C^2 case.

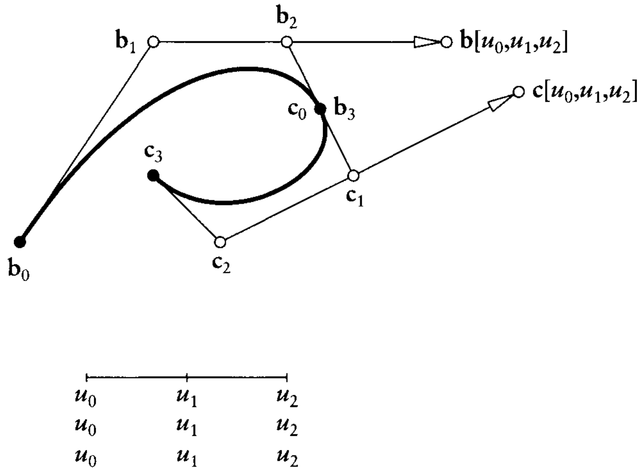


Figure 8.16 Smoothness of Bézier curves: the C^2 condition is violated.

8.8 B-Splines

Consider a knot sequence $u_0, \ldots, u_M$ 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

$$\dim = (n+1) + \sum_{i=1}^{M-1} r_i. \tag{8.13}$$

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 (8.13).

We are interested in B-spline curves that are piecewise polynomials over the special knot sequence $[u_{n-1}, u_L]$. The dimension of the linear space that they form is $L+1$, which also happens to be the number of B-spline vertices for a curve in this space. If we can define $L+1$ 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 $[u_{n-1}, u_L]$. 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 abscissae 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 (8.13). 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

$$\sum_{j=0}^{L} c_j N_j^n(u) \equiv 0 \tag{8.14}$$

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, (8.14) reduces to

$$\sum_{j=I-n+1}^{I+1} c_j N_j^n(u) \equiv 0 \quad \text{for } u \in [u_I, u_{I+1}].$$

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.

Because the B-splines N_i^n are linearly independent, every piecewise polynomial s over $[u_{n-1}, u_L]$ may be written uniquely in the form

$$s(u) = \sum_{j=0}^{L} d_j N_j^n(u). \tag{8.15}$$

The B-splines thus form a *basis* for this space. This reveals the origin of their name, which is short for Basis splines. Figure 8.17 gives examples of some cubic B-splines.

If we set all $d_i = 1$ in (8.15), the function $s(u)$ will be identically equal to 1, thus asserting that B-splines form a *partition of unity.*

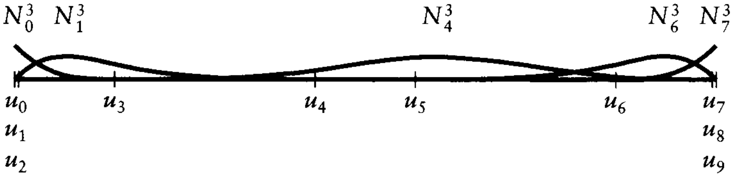


Figure 8.17 B-splines: some cubic examples.

B-spline curves are simply the parametric equivalent of (8.15):

$$\mathbf{x}(u) = \sum_{j=0}^{L} \mathbf{d}_j N_j^n(u).$$

Just as the de Casteljau algorithm for Bézier curves is related to the recursion of Bernstein polynomials, the de Boor algorithm yields a recursion for B-splines. It is given by

$$N_l^n(u) = \frac{u - u_{l-1}}{u_{l+n-1} - u_{l-1}} N_l^{n-1}(u) + \frac{u_{l+n} - u}{u_{l+n} - u_l} N_{l+1}^{n-1}(u), \tag{8.16}$$

with the "anchor" for the recursion being given by

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_{i-1} \le u < u_i \\ 0 & \text{else} \end{cases}. \tag{8.17}$$

Its proof relates the local recursion (8.10) to the global indexing scheme. An example is shown in Figure 8.18.

Equation (8.16) is due to L. Mansfield, C. de Boor, and M. Cox; see de Boor [137] and Cox [129]. For an illustration of (8.16), see Figure 8.18. This formula 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 same parameter values u_k, it is a good idea to compute the values for $N_i^n(u_k)$ using (8.16) and then to store them.

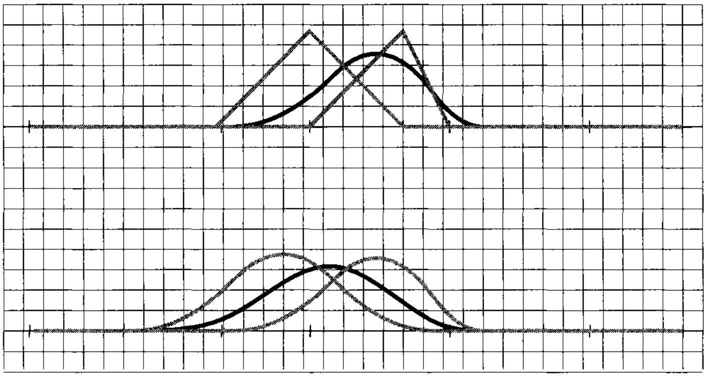


Figure 8.18 The B-spline recursion: top, two linear B-splines yield a quadratic one; bottom, two quadratic B-splines yield a cubic one.

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 like (8.16) seemingly require the presence of these knots. Since they are multiplied only by zero factors, their values have no influence on any computation. There is no reason to store completely inconsequential data, and hence the "leaner" notation of this chapter.

8.9 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, $L+1$ is the number of control points, and K is the number of intervals.

Knot sequence: $\{u_0, \ldots, u_K\}$.

Control points: $\mathbf{d}_0, \ldots, \mathbf{d}_L$, with $L = K - n + 1$.

Domain: Curve is only defined over $[u_{n-1}, \ldots, u_L]$.

Greville abscissae: $\xi_i = \frac{1}{n}(u_i + \cdots + u_{i+n-1})$.

Support: N_i^n is nonnegative over $[u_{i-1}, u_{i+n}]$.

Knot insertion: To insert $u_I \le u < u_{I+1}$, first find new Greville abscissae $\hat{\xi}_i$, then set new $d_i = P(\hat{\xi}_i)$.

de Boor algorithm: Given $u_I \le u < u_{I+1}$, renumber the relevant control points $\mathbf{d}_{I-n+1}, \ldots, \mathbf{d}_{I+1}$ as $\mathbf{d}_0, \ldots, \mathbf{d}_n$ and then set

$$\mathbf{d}_i^k(u) = (1 - \alpha_i^k)\mathbf{d}_i^{k-1}(u) + \alpha_i^k \mathbf{d}_{i+1}^{k-1}(u)$$

with

$$\alpha_i^k = \frac{u - u_{I+i+1}}{u_{I-n+k+i} - u_{I+i+1}}$$

for $k = r+1, \ldots, n$, and $i = 0, \ldots, n-k$. Here, r denotes the multiplicity of u. (Normally, u is not already in the knot sequence; then, $r = 0$.)

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:

$$\frac{\mathrm{d}}{\mathrm{d}u}N_l^n(u) = \frac{n}{u_{n+l-1} - u_{l-1}} N_l^{n-1}(u) - \frac{n}{u_{l+n} - u_l} N_{l+1}^{n-1}(u).$$

Derivative of B-spline curve:

$$\frac{\mathrm{d}}{\mathrm{d}u}s(u) = n \sum_{i=0}^{L-1} \frac{\Delta \mathrm{d}_i}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u).$$

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 [493]. Another reference is Barry and Goldman [39].

8.10 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 5.11.

As a second example, the input data for the *y*-values of the curve in Figure 8.10 are

```
degree = 3; l = 3; coeff = 1,4,4,0,0,1;
knot = 0,0,0,3,9,12,12,12; 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

          */
```

8.11 Problems

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 Derive (8.16) from (8.10).

*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 and (b) uniform knot spacing and knots all with multiplicity two.

P1 Use `de_boor_blossom` to program degree elevation for B-spline curves.

Constructing Spline Curves

9

A *spline* is a flexible rod of wood or plastic. It has its origins in shipbuilding, where splines were used to draft the curves (ribs) that define a ship body. Early uses go back to the 1600s, and are documented in [450]. Although mechanical splines are used less frequently now, the underlying principle still gives rise to new algorithms.

9.1 Greville Interpolation

In Chapter 7, we saw how to pass a polynomial curve of degree n through $n+1$ data points $\mathbf{p}_0, \ldots, \mathbf{p}_n$ with parameter values $t_0, \ldots, t_n$. The key to the solvability of the problem was simple: the number of knowns (the data points with parameter values) had to equal the number of unknowns (the polynomial coefficients).

Something quite analogous happens in a spline context. A spline curve of degree n is defined over a knot sequence $u_0, \ldots, u_K$. Such a knot sequence has $K-n+2$ Greville abscissae ξ_i and hence the spline curve has $L+1=K-n+2$ B-spline control points $\mathbf{d}_0, \ldots, \mathbf{d}_L$.

In view of these numbers, the following is a meaningful interpolation problem:

Given: A knot sequence $u_0, \ldots, u_K$ and a degree n, also a set of data points $\mathbf{p}_0 \cdots \mathbf{p}_L$ with $L=K-n+1$.

Find: A set of B-spline control points $\mathbf{d}_0, \ldots, \mathbf{d}_L$ such that the resulting curve $\mathbf{x}(u)$ satisfies

$$\mathbf{x}(\xi_i)=\mathbf{p}_i; \quad i=0, \ldots, L. \tag{9.1}$$

In this case, the parameter values associated with the data points are not the knots u_i but rather the Greville abscissae ξ_i. This gives us a problem in which the number of unknowns equals the number of knowns.

The solution is obtained in complete analogy to the polynomial case: write out (9.1) as

$$\mathbf{p}_i = \sum_{j=0}^{L} \mathbf{d}_j N_j^n(\xi_i); \quad i = 0, \ldots, L \tag{9.2}$$

and collect them in matrix form:

$$\begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_L \end{bmatrix} = \begin{bmatrix} N_0^n(\xi_0) & \ldots & N_L^n(\xi_0) \\ & \vdots & \\ N_0^n(\xi_L) & \ldots & N_L^n(\xi_L) \end{bmatrix} \begin{bmatrix} \mathbf{d}_0 \\ \vdots \\ \mathbf{d}_L \end{bmatrix}. \tag{9.3}$$

There is a significant difference to the polynomial case: the matrix in (9.3) has nonzero entries only near the diagonal. Because of the local support property of B-splines, most of the $N_j^n(\xi_i)$ are zero; at most $n + 1$ of them are nonzero for any ξ_i. This means that the matrix in (9.3) is *banded* and thus much easier to handle than a full matrix as encountered in polynomial interpolation. The cubic case (with triple end knots and simple interior knots) looks like this:

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \\ \mathbf{p}_5 \\ \mathbf{p}_6 \end{bmatrix} = \begin{bmatrix} \star & & & & & & & \\ \star & \star & \star & \star & & & & \\ & \star & \star & \star & \star & & & \\ & & \star & \star & \star & \star & & \\ & & & \star & \star & \star & \star & \\ & & & & \star & \star & \star & \star \\ & & & & & & & \star \end{bmatrix} \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \\ \mathbf{d}_4 \\ \mathbf{d}_5 \\ \mathbf{d}_6 \end{bmatrix}. \tag{9.4}$$

The $\star$ elements represent the nonzero $N_j^3(\xi_i)$; zero matrix entries are left blank. Instead of employing a general-purpose linear system solver, routines for banded matrices may be used—they are much more efficient.

Greville interpolation works well where the given data points correspond to the Greville abscissae of a knot sequence. It is the most commonly used method for quadratic spline interpolation; see [138] and [156].

In most practical situations, however, it will be hard to come up with such a knot sequence, and different methods are employed.

9.2 Least Squares Approximation

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. We encountered situations like this in Section 7.8.

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.[1] 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.

To make matters more precise, assume that we are given data points $\mathbf{p}_i$ with $i = 0, \ldots, P$.[2] We wish to find an approximating B-spline curve $\mathbf{p}(u)$ of degree n with $K + 1$ knots $u_0, \ldots, u_K$. We want the curve to be close to the data points in the least squares sense. Suppose the data point $\mathbf{p}_i$ is associated with a data parameter value w_i.[3] 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

$$\text{minimize} \sum_{i=0}^{P} \|\mathbf{p}_i - \mathbf{p}(w_i)\|^2 . \tag{9.5}$$

The squared distances are introduced to simplify our subsequent computations. They gave the name to this method: *least squares approximation*. We shall minimize (9.5) by finding suitable B-spline control vertices $\mathbf{d}_j$:

$$\text{minimize } f(\mathbf{d}_0, \ldots, \mathbf{d}_L) = \sum_{i=0}^{P} \left\| \mathbf{p}_i - \sum_{j=0}^{L} \mathbf{d}_j N_j^n(w_i) \right\|^2 . \tag{9.6}$$

[1] Many explicit wing section equations are given by the so-called NACA profiles.

[2] We are thus assuming that the data points are numbered in a meaningful order. The problem changes completely if this assumption is not valid, see [382].

[3] Note that w_i does not have to be one of the knots!

The least squares approach—identical to our development of the polynomial case in Section 7.8—produces the *normal equations*

$$\sum_{j=0}^{L} \mathbf{d}_j \sum_{i=0}^{P} N_j^n(w_i) N_k^n(w_i) = \sum_{i=0}^{P} \mathbf{p}_i N_k^n(w_i); \quad k = 0, \ldots, L. \tag{9.7}$$

This is a linear system of $L+1$ equations for the unknowns $\mathbf{d}_k$, with a coefficient matrix M whose elements $m_{j,k}$ are given by

$$m_{j,k} = \sum_{i=0}^{P} N_j^n(w_i) N_k^n(w_i); \quad 0 \le j, k \le L.$$

The symmetric matrix M 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 [376].

The matrix M is singular if and only if there is a span $[u_{j-1}, u_{j+n}]$ that contains no w_i. This fact is known as the *Schoenberg–Whitney theorem.*

In cases where there are gaps in the data points, there is still a remedy: we may employ *smoothing equations* in exactly the same way as was done in Section 7.9. The addition of these equations, now applied to B-spline control vertices instead of Bézier control vertices, will guarantee a solution. In cases where there is noise in the data, these equations will also help in obtaining a better shape of the least squares curve.

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 9.6 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.

Figures 9.1 through 9.4 give some examples. In all four figures, 1,000 points were sampled from a spiral, and noise was added. The parameter values were assigned according to the centripetal parametrization; the knots were assigned uniformly. The best fit and shape is obtained, not surprisingly, by using a relatively high degree and many intervals; see Figure 9.4. The corresponding curvature plots are shown in Figure 23.1.

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 $\mathbf{p}(w_i)$

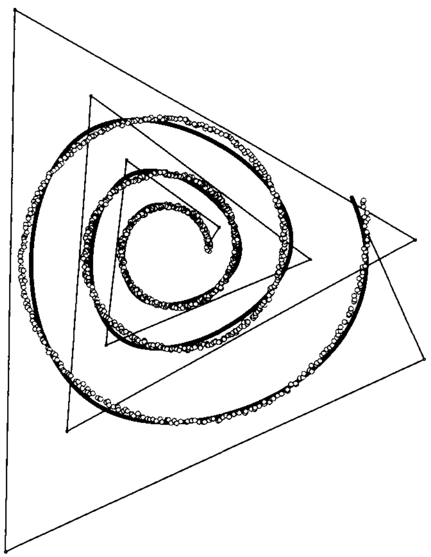


Figure 9.1 Least squares approximation: $n = 3, K = 9$. Figure courtesy M. Jeffries.

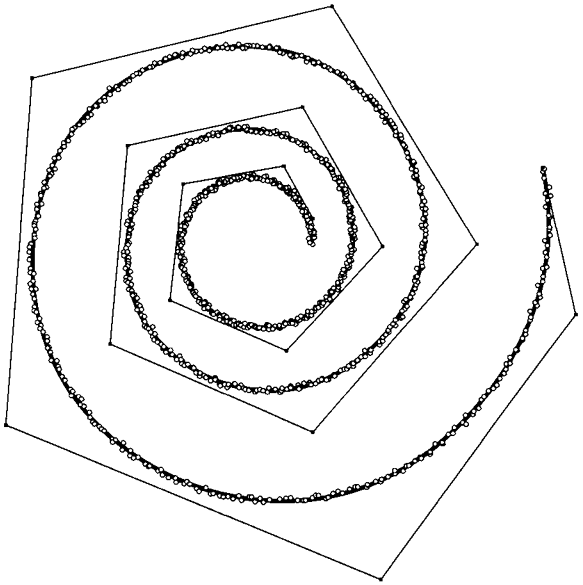


Figure 9.2 Least squares approximation: $n = 3, K = 15$. Figure courtesy M. Jeffries.

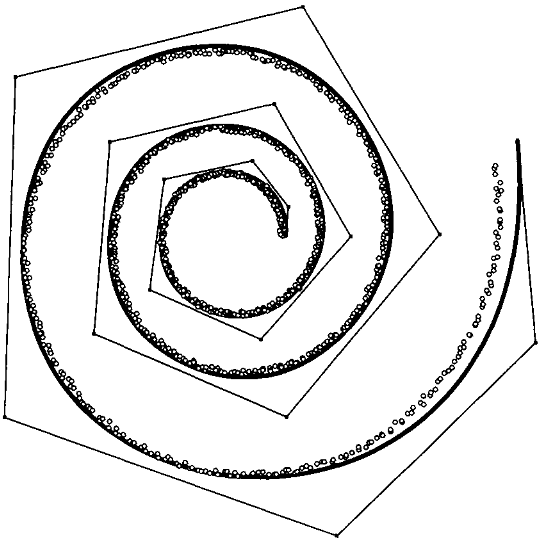


Figure 9.3 Least squares approximation: $n = 3, K = 15$, smoothing factor $\alpha = 0.05$. Figure courtesy M. Jeffries.

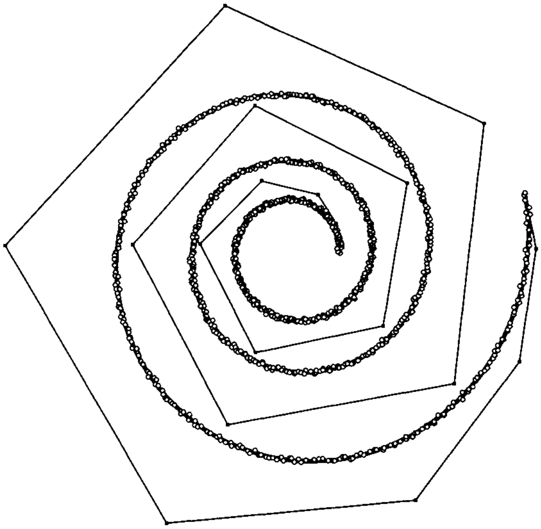


Figure 9.4 Least squares approximation: $n = 6, K = 15$. Figure courtesy M. Jeffries.

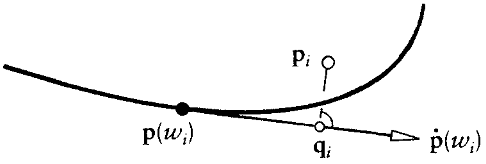


Figure 9.5 Parameter correction: the connection of $\mathbf{p}_i$ and $\mathbf{p}(w_i)$ is typically not perpendicular to the tangent at $\mathbf{p}(w_i)$. A better value for w_i is found by projecting $\mathbf{p}_i$ onto the tangent.

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 9.5. 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.[4] It was named *parameter correction* by J. Hoschek [337], [577].

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 $\|\mathbf{q}_i - \mathbf{p}(w_i)\|/\|\dot{\mathbf{p}}(w_i)\|$ is a measure for the adjustment of w_i. The actual Newton iteration step looks like this:

$$\hat{w}_i = w_i + [\mathbf{p}_i - \mathbf{p}(w_i)]\frac{\dot{\mathbf{p}}(w_i)}{\|\dot{\mathbf{p}}(w_i)\|^2}\frac{\Delta u_k}{s_k}. \tag{9.8}$$

In this equation, s_k denotes the arc length of the segment that w_i is in, that is, $u_k < w_i \leq u_{k+1}$. This length may safely (and cheaply) be overestimated by the length of the Bézier polygon of the k^{th} segment.

We finally note that (9.8) should not be used to compute the point on a curve closest to an arbitrary point $\mathbf{p}_i$. It works only 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.

9.3 Modifying B-Spline Curves

The use of B-spline curves is often described like this: pick a control point, move it, observe the shape of the resulting curve, and stop once a desired shape is obtained. It is more intuitive, however, to let a user pick a point *on the curve*, move it to a

[4] In theory, more iterations should produce better fits. In practice, however, the fit often deteriorates after more than four iterations.

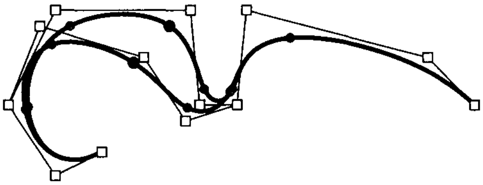


Figure 9.6 Modifying B-spline curves: a point on a cubic B-spline curve is moved and a new polygon is computed that passes through the new point.

new position, and then compute a control polygon that will accommodate this change. We already did this for the case of Bézier curves in Section 7.10.

Having solved the Bézier case makes the corresponding B-spline problem a relatively easy task. Suppose we want to change a point $\mathbf{x}(\hat{u})$ on a B-spline curve. Because of the local control property of B-spline curves, this point is determined by $n+1$ B-spline control points $\mathbf{d}_i$. With some relabeling, we can write it as

$$\mathbf{x}(\hat{u}) = \sum_{i=0}^{n} \mathbf{d}_i N_i^n(\hat{u}).$$

We wish to find new control points $\hat{\mathbf{d}}_i$ such that the new curve goes through a point $\mathbf{y}$ for parameter value $\hat{u}$. This is exactly the same problem as for the Bézier case if we want to change only $\mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$. If changing all vertices $\mathbf{d}_0, \ldots, \mathbf{d}_n$ is desired, the linear system has to be changed to

$$[\, N_0^n(\hat{u}) \quad \ldots \quad N_n^n(\hat{u}) \,] \begin{bmatrix} \mathbf{d}_0 \\ \vdots \\ \mathbf{d}_n \end{bmatrix} = \mathbf{x}(\hat{u}). \tag{9.9}$$

This is a linear system consisting of one equation for $n+1$ unknowns and is solved as in Section 7.10. Figure 9.6 gives an example.

It is possible to change not only a point on a curve. Since this results in an underdetermined system, more constraints such as more point changes or prescription of derivatives may be added. This was covered in detail by Bartels and Forsey [48].

9.4 C^2 Cubic Spline Interpolation

This is the most popular of all spline algorithms. For this section, we will use the condensed form of the knot sequence, denoting each knot as τ_i. We will be

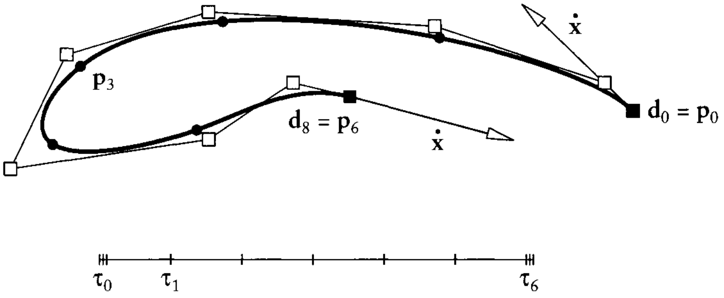


Figure 9.7 Cubic spline interpolation: the case of clamped end conditions with six intervals.

dealing only with knot sequences that are simple ($\mu_i = 1$) except for the end knots that are of multiplicity three: $\mu_0 = 3, \mu_K = 3$.

The problem statement is as follows:

Given: A set of data points $\mathbf{p}_0, \ldots, \mathbf{p}_K$ and a knot sequence $\tau_0, \ldots, \tau_K$ and a multiplicity vector $3, 1, 1, \ldots, 1, 1, 3$.

Find: A set of B-spline control points $\mathbf{d}_0, \ldots, \mathbf{d}_L$ with $L = K + 2$ such that the resulting C^2 piecewise cubic curve $\mathbf{x}(u)$ satisfies

$$\mathbf{x}(\tau_i) = \mathbf{p}_i; \quad i = 0, \ldots, K. \tag{9.10}$$

Consult Figure 9.7 for an example of the numbering scheme.

As it turns out, the preceding problem is ill posed: the number of unknowns is $K + 3$, whereas the number of given data points is $K + 1$. This is an underdetermined problem; two more conditions are needed in order to have a uniquely solvable problem. We will discuss several possibilities for this; to begin with, we consider *clamped end conditions.*

A clamped end condition corresponds to the prescription of two derivatives $\dot{\mathbf{x}}(\tau_0)$ and $\dot{\mathbf{x}}(\tau_K)$. They are given by

$$\dot{\mathbf{x}}(\tau_0) = \frac{3}{\tau_1 - \tau_0}[\mathbf{d}_1 - \mathbf{d}_0], \qquad \dot{\mathbf{x}}(\tau_K) = \frac{3}{\tau_K - \tau_{K-1}}[\mathbf{d}_L - \mathbf{d}_{L-1}].$$

The geometry of this interpolation problem is illustrated in Figure 9.7

Because of the triple end knots, we immediately have

$$\mathbf{d}_0 = \mathbf{p}_0 \quad \text{and} \quad \mathbf{d}_L = \mathbf{p}_K;$$

this takes care of equations #0 and #K of (9.10). We will not use these in setting up our linear system.

The clamped end conditions yield

$$\mathbf{d}_1 = \mathbf{d}_0 + \frac{\tau_1 - \tau_0}{3}\dot{\mathbf{x}}(\tau_0) \quad \text{and} \quad \mathbf{d}_{L-1} = \mathbf{d}_L - \frac{\tau_K - \tau_{K-1}}{3}\dot{\mathbf{x}}(\tau_K).$$

Thus, the first and last equations of our linear system become

$$\mathbf{d}_2 N_2^3(\tau_1) + \mathbf{d}_3 N_3^3(\tau_1) = \mathbf{r}_s \quad \text{and} \quad \mathbf{d}_{L-3} N_{L-3}^3(\tau_{K-1}) + \mathbf{d}_{L-2} N_{L-2}^3(\tau_{K-1}) = \mathbf{r}_e$$

with

$$\mathbf{r}_s = \mathbf{p}_1 - \mathbf{d}_1 N_1^3(\tau_1) \quad \text{and} \quad \mathbf{r}_e = \mathbf{p}_{K-1} - \mathbf{d}_{L-1} N_{L-1}^3(\tau_{K-1}).$$

Each of the remaining unknowns $\mathbf{d}_2, \ldots, \mathbf{d}_{L-2}$ is related to the data points by one equation. Because of the local support property of cubic B-splines, it is of the form

$$\mathbf{p}_i = \mathbf{d}_i N_i^3(\tau_i) + \mathbf{d}_{i+1} N_{i+1}^3(\tau_i) + \mathbf{d}_{i+2} N_{i+2}^3(\tau_i); \quad i = 2, \ldots, K-2. \tag{9.11}$$

Together, these are $K-1$ equations for the $K-1$ unknown control points. In matrix form, we have

$$\begin{bmatrix} N_2^3(\tau_1) & N_3^3(\tau_1) & & \\ N_1^3(\tau_2) & N_2^3(\tau_2) & N_3^3(\tau_2) & \\ & \vdots & & \\ & & \vdots & \\ & N_{L-3}^3(\tau_{K-2}) & N_{L-2}^3(\tau_{K-2}) & N_{L-1}^3(\tau_{K-2}) \\ & & N_{L-3}^3(\tau_{K-1}) & N_{L-2}^3(\tau_{K-1}) \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{d}_2 \\ \vdots \\ \vdots \\ \mathbf{d}_{L-2} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_s \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{K-2} \\ \mathbf{r}_e \end{bmatrix} \tag{9.12}$$

Schematically, the case $K = 5$ looks like this:

$$\begin{bmatrix} \star & \star & & \\ \star & \star & \star & \\ & \star & \star & \star \\ & & \star & \star \end{bmatrix} \begin{bmatrix} \mathbf{d}_2 \\ \mathbf{d}_3 \\ \mathbf{d}_4 \\ \mathbf{d}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{r}_s \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{r}_e \end{bmatrix}$$

The entries in this tridiagonal matrix are easily computed from the definitions of the cubic B-splines N_i^3.

In the case of uniform knots $u_i = i$, the interpolation conditions (9.11) take on a particularly simple form:

$$6\mathbf{p}_i = \mathbf{d}_i + 4\mathbf{d}_{i+1} + \mathbf{d}_{i+2}; \quad i = 2, \ldots, K-2. \tag{9.13}$$

We conclude with a method for cubic spline interpolation that occasionally appears in the literature (e.g., in Yamaguchi [621]). It is possible to solve the interpolation problem without setting up a linear system! Just do the following: define $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_{L-1}, \mathbf{d}_L$ as before, and set $\mathbf{d}_i = \mathbf{p}_{i-1}$ for all other control points. Then, for $i = 2, \ldots, L-2$, correct the location of $\mathbf{d}_i$ such that the curve passes through $\mathbf{p}_{i-1}$. 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 earlier iterative method amounts to solving the system via Gauss–Seidel iteration. So though geometrically appealing, the iterative method needs more computation time than the direct method.

9.5 More End Conditions

For cubic spline interpolation, the choice of end conditions is important for the shape of the interpolant near the endpoints—they do not matter much in the interior. We have seen a clamped end condition earlier. It works well in situations where the end derivatives are actually known. But in most applications, we do not have this knowledge. Still, two extra equations are needed in addition to the basic interpolation conditions (9.10).

We list several possibilities:

The *natural end conditions* are derived from the physical analogy of a wooden beam that is clamped at some positions; see Figure 9.22. Beyond the first and last clamps, such a beam assumes the shape of a straight line. A line is characterized by having a zero second derivative, and hence the end conditions

$$\ddot{\mathbf{x}}(\tau_0) = 0, \qquad \ddot{\mathbf{x}}(\tau_K) = 0$$

are called natural end conditions.

The second derivative of a Bézier curve at $\mathbf{b}_0$ is given by $\mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2$. In terms of B-spline control vertices (using triple end knots), this becomes

$$\mathbf{d}_0 - 2\mathbf{d}_1 + \frac{\Delta_1}{\Delta_0 + \Delta_1}\mathbf{d}_1 + \frac{\Delta_0}{\Delta_0 + \Delta_1}\mathbf{d}_2 = 0,$$

where we set $\Delta_i = \tau_{i+1} - \tau_i$. We rearrange and obtain

$$(\Delta_0 + \Delta_1)\mathbf{d}_0 - (2\Delta_0 + \Delta_1)\mathbf{d}_1 + \Delta_0\mathbf{d}_2 = \mathbf{0}. \tag{9.14}$$

A similar condition holds at the other endpoint. Unless required by a specific application, this end condition should be avoided since it forces the curve to behave linearly near the endpoints.

Typically another end condition, called Bessel end condition, yields better results. It works as follows: the first three data points and their parameter values determine an interpolating quadratic curve. Its first derivative at $\mathbf{p}_0$ is taken to be the one for the spline curve. If we set

$$\alpha = \frac{\Delta_1}{\tau_2 - \tau_0} \quad \text{and} \quad \beta = 1 - \alpha$$

and

$$\mathbf{a} = \frac{1}{2\alpha\beta}(\mathbf{p}_1 - \alpha^2\mathbf{p}_0 - \beta^2\mathbf{p}_2),$$

then

$$\mathbf{d}_1 = \frac{2}{3}(\alpha\mathbf{p}_0 + \beta\mathbf{a}) + \frac{1}{3}\mathbf{p}_0.$$

This value for $\mathbf{d}_1$ is then used in the earlier clamped end condition. The control point $\mathbf{d}_{L-1}$ is obtained in complete analogy; it is then also used for a clamped end condition.

Other end conditions exist: requiring $\ddot{\mathbf{x}}(\tau_0) = \ddot{\mathbf{x}}(\tau_1)$ is called a *quadratic* end condition. If all data points and parameter values were read off from a quadratic curve, then this condition would ensure that the spline interpolant reproduces that quadratic. The same is true, of course, for the Bessel end conditions. If the first and second cubic segment are parts of *one* cubic, then their third derivatives at τ_1 would agree. The corresponding end condition is called *not-a-knot* condition since the knot τ_1 does not act as a breakpoint between two distinct cubic segments.

We finish this section with a few examples, courtesy T. Foley, using uniform parameter values in all examples.[5] Figure 9.8 shows equally spaced data

[5] Owing to 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.

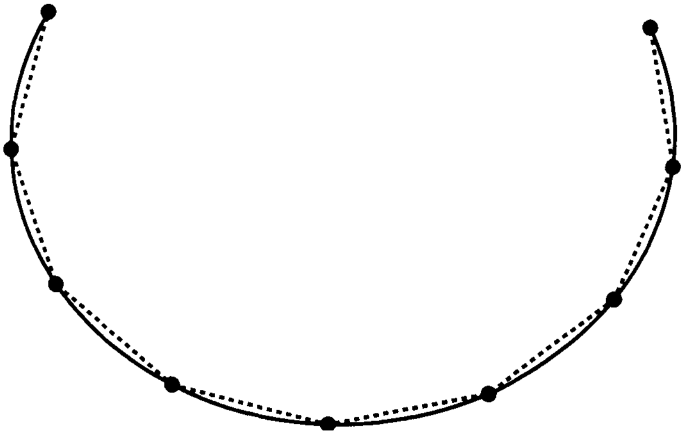


Figure 9.8 Exact clamped end condition spline.

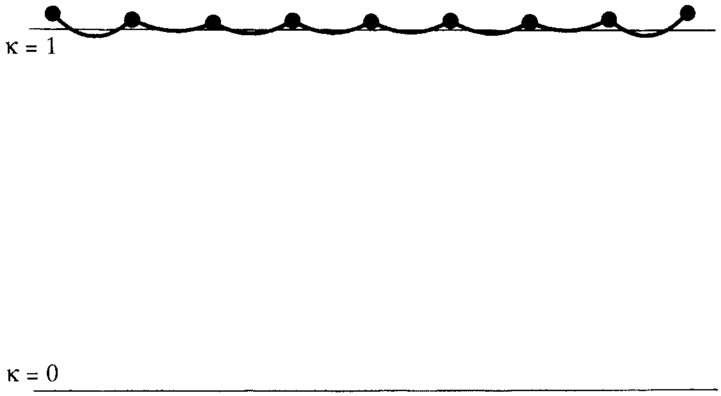


Figure 9.9 Curvature plot of exact clamped end condition spline.

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 9.9 shows the curvature plot[6] of the spline curve. Ideally, the curvature should be constant, and the spline curvature is quite close to this ideal.

[6] The graph of curvature versus arc length; see also Chapter 23.

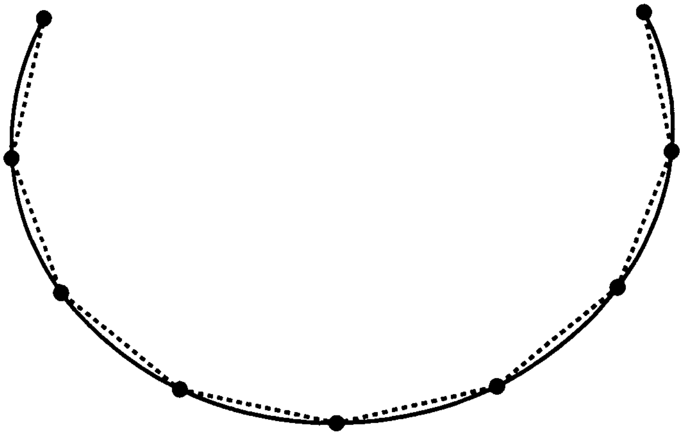


Figure 9.10 Bessel end condition spline.

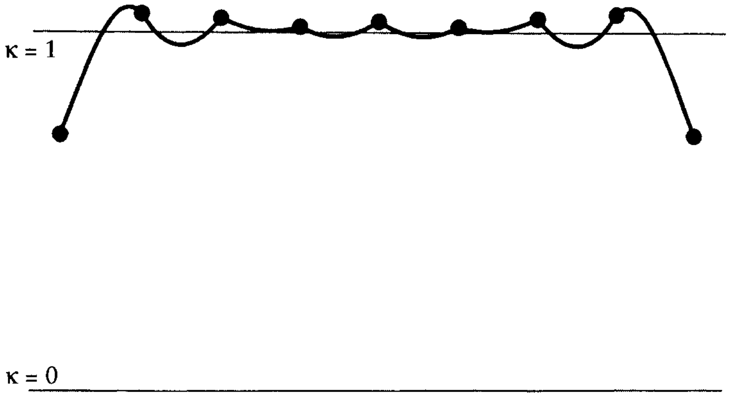


Figure 9.11 Curvature plot of Bessel end condition spline.

Figure 9.10 shows the same data, but now using Bessel end conditions. Near the endpoints, the curvature deviates from the ideal value, as shown in Figure 9.11.

Finally, Figure 9.12 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 9.13. This end condition should be avoided unless the linear behavior near the ends is desired.

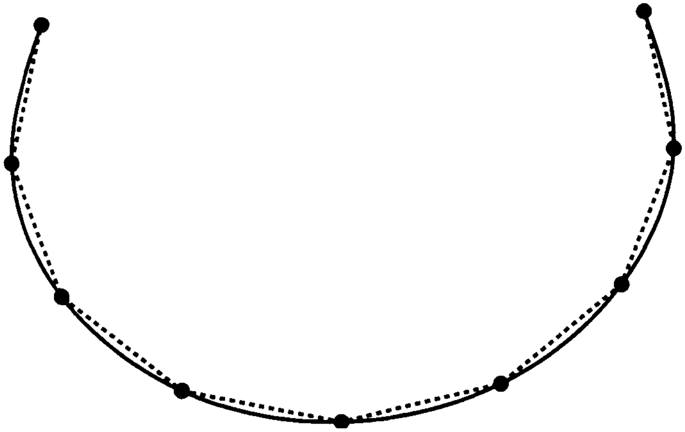


Figure 9.12 Natural end condition spline.

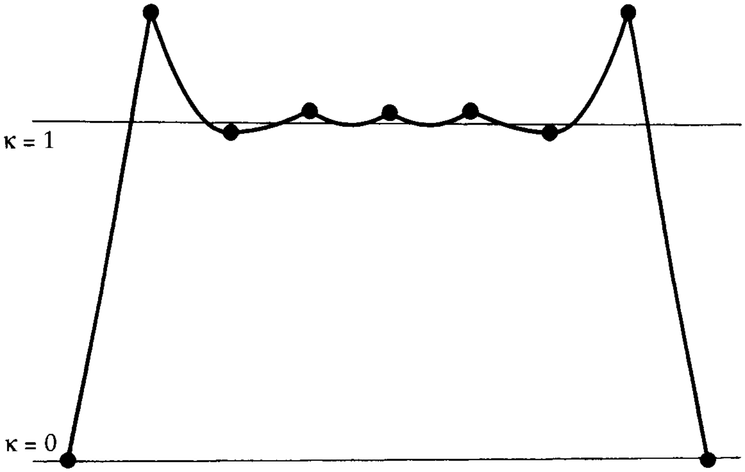


Figure 9.13 Curvature plot of natural end condition spline.

9.6 Finding a Knot Sequence

The spline interpolation problem is usually stated as "given data points $\mathbf{p}_i$ and parameter values u_i, ..." 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[7] 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 since 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:

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{\|\Delta \mathbf{p}_i\|}{\|\Delta \mathbf{p}_{i+1}\|}. \tag{9.15}$$

A knot sequence satisfying (9.15) is called *chord length parametrization.* Equation (9.15) 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 [186]) that chord length parametrization (in connection with natural end conditions) cannot produce curves with corners[8] at the data points, which gives it some theoretical advantage over the uniform choice.

Another parametrization has been named "centripetal" by E. Lee [378]. It is derived from the physical heuristics presented earlier. If we set

[7] There are cases in which uniform parametrization fares better than other methods. An interesting example is in Foley [239], p. 86.

[8] 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 using the chord length parametrization.

$$\frac{\Delta_i}{\Delta_{i+1}} = \left[\frac{\|\Delta \mathbf{p}_i\|}{\|\Delta \mathbf{p}_{i+1}\|} \right]^{1/2}, \tag{9.16}$$

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 [449]. It sets

$$\Delta_i = d_i \left[1 + \frac{3}{2} \frac{\hat{\Theta}_i d_{i-1}}{d_{i-1} + d_i} + \frac{3}{2} \frac{\hat{\Theta}_{i+1} d_{i+1}}{d_i + d_{i+1}} \right], \tag{9.17}$$

where $d_i = \|\Delta \mathbf{p}_i\|$ and

$$\hat{\Theta}_i = \min \left(\pi - \Theta_i, \frac{\pi}{2} \right),$$

and Θ_i is the angle formed by $\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}$. Thus $\hat{\Theta}_i$ is the "adjusted" exterior angle formed by the vectors $\Delta \mathbf{p}_i$ and $\Delta \mathbf{p}_{i-1}$. 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 [446].[9]

For more literature on parametrizations, see Cohen and O'Dell [122], Hartley and Judd [314], [315], McConalogue [421], and Foley [239].

Figures 9.14 to 9.21[10] 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.

Although the figures are self-explanatory, some comments are in order. 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

[9] The Foley parametrization was in fact first formulated in terms of that modified length measure.

[10] Kindly provided by T. Foley.

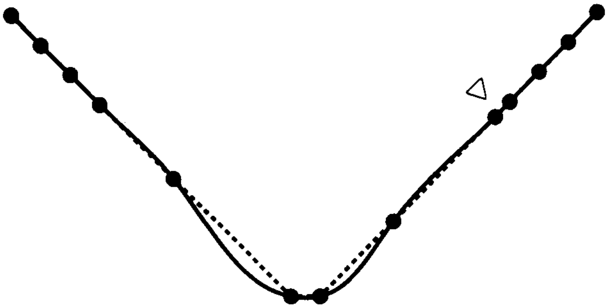


Figure 9.14 Chord length spline.

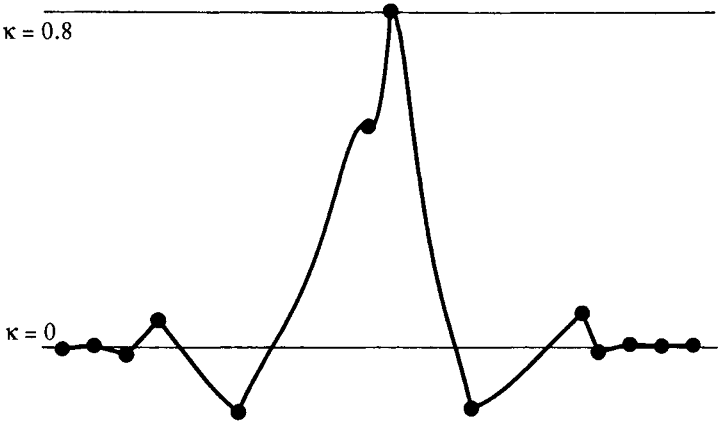


Figure 9.15 Curvature plot of chord length spline.

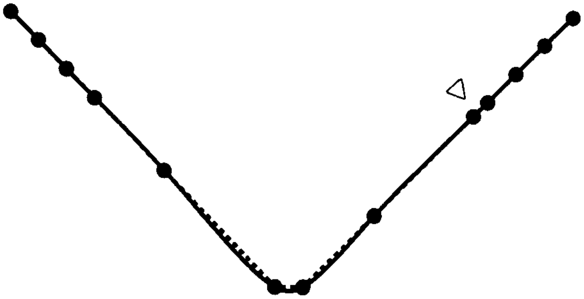


Figure 9.16 Foley spline.

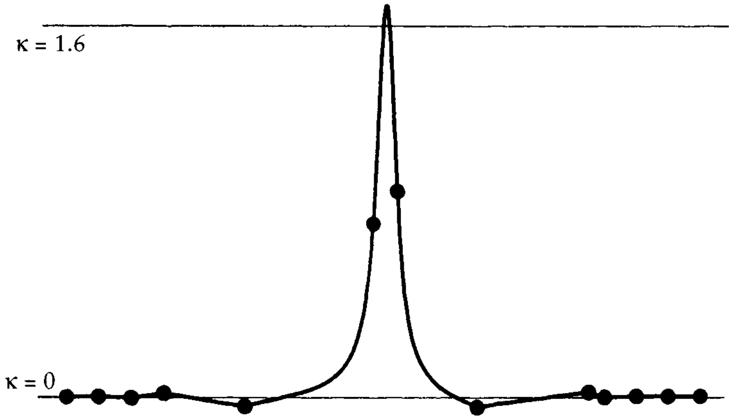


Figure 9.17 Curvature plot of Foley spline.

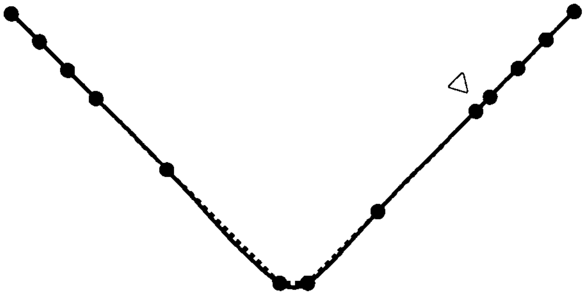


Figure 9.18 Centripetal spline.

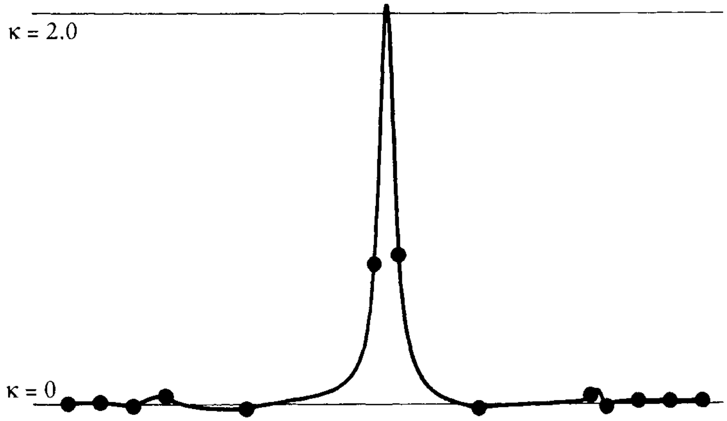


Figure 9.19 Curvature plot of centripetal spline.

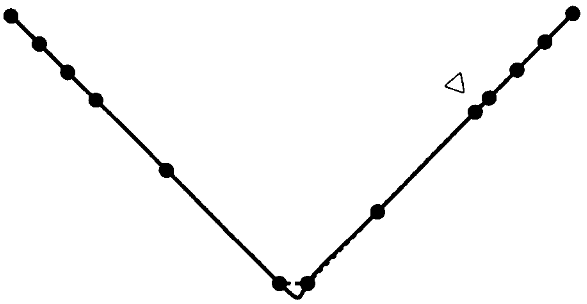


Figure 9.20 Uniform spline.

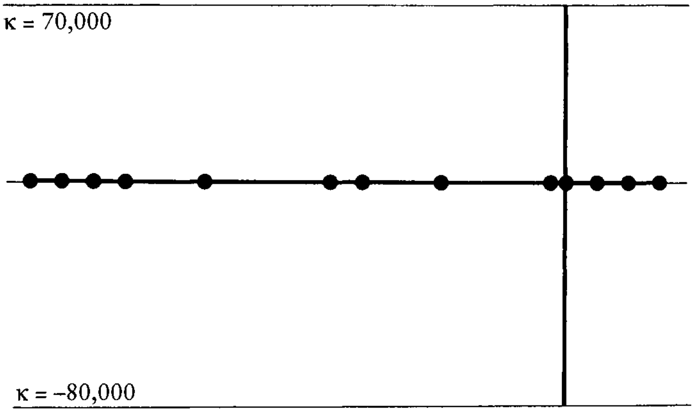


Figure 9.21 Curvature plot of uniform spline.

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.

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 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.

9.7 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 9.22.

The plastic or wooden beam assumes a position that minimizes its strain energy. The mathematical model of the beam is a curve s, and its strain energy E is given by

$$E = \int (\kappa(s))^2 ds,$$

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 with a simpler one:

$$\hat{\mathrm{E}} = \int \left[\frac{d^2}{du^2} \mathrm{s}(u) \right]^2 du. \qquad (9.18)$$

Note that $\hat{\mathrm{E}}$ is a vector; it is obtained by performing the integration on each component of s.

Equation (9.18) 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 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

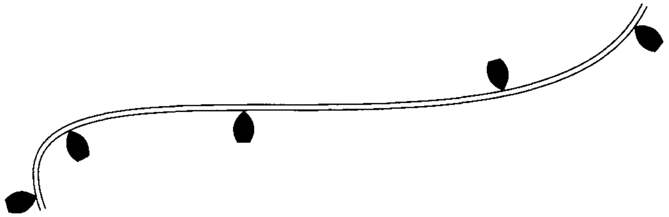


Figure 9.22 Spline interpolation: a plastic beam, the spline, is forced to pass through data points, marked by metal weights, the ducks.

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![11]

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 $\hat{\mathrm{E}}$. For a proof, let $\mathrm{s}(u)$ be the C^2 cubic spline and let $\mathrm{y}(u)$ be another C^2 interpolating curve. We can write y as

$$\mathrm{y}(u) = \mathrm{s}(u) + [\mathrm{y}(u) - \mathrm{s}(u)].$$

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 s and y, respectively. The "energy integral" $\hat{E}$ of 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 term vanishes because of the common end conditions. In the second term, $\dddot{s}$ is piecewise constant:

$$\int_{u_0}^{u_L} \dddot{s}(\dot{y} - \dot{s})\mathrm{d}u = \sum_{j=0}^{L-1} \dddot{s}_j (y - s)\Big|_{u_j}^{u_{j+1}}.$$

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 $\ddot{y} \neq \ddot{s}$,

$$\int_{u_0}^{u_L} (\ddot{y})^2 \mathrm{d}u \geq \int_{u_0}^{u_L} (\ddot{s})^2 \mathrm{d}u, \tag{9.19}$$

we have proved the claimed minimum property.

[11] I am grateful to P. Crouch for bringing the airplane analogy to my attention.

The minimum property of splines has spurred substantial research activity. The replacement of the actual strain energy measure E by $\hat{\mathrm{E}}$ 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 (10.7). This assumption is not very realistic for cubic splines in a design environment; see Problems.

While the classical spline curve merely minimizes an approximation to (9.18), methods have been developed that produce interpolants that minimize the true energy (9.18), see [425], [99]. Moreton and Séquin have suggested to minimize the functional $\int[\kappa'(t)]^2dt$ instead; see [431].

9.8 C^1 Piecewise Cubic Interpolation

Spline curves come in the B-spline form, but they may also be described as piecewise Bézier curves. We now consider that approach, applied to piecewise cubic interpolation. First, 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; see Figure 9.23.

Find: A C^1 piecewise cubic polynomial that passes through the given data points and is tangent to the given tangent directions there.

It is important to note that we only have tangent *directions*, that is, we have no vectors with a prescribed length since our problem statement did not involve

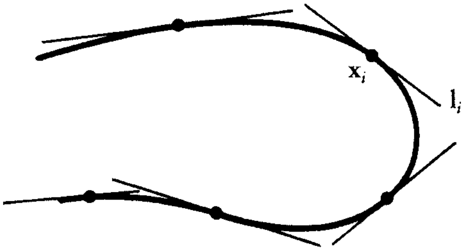


Figure 9.23 C^1 piecewise cubics: example data set.

a knot sequence. 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 $\mathbf{b}_{3i} = \mathbf{x}_i, \quad i = 0, \ldots, L$.

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 (5.30)].

In general, we must determine the inner Bézier points from

$$\mathbf{b}_{3i+1} = \mathbf{b}_{3i} + \alpha_i \mathbf{l}_i, \tag{9.20}$$

$$\mathbf{b}_{3i-1} = \mathbf{b}_{3i} - \beta_{i-1} \mathbf{l}_i, \tag{9.21}$$

so that the problem boils down to finding reasonable values for α_i and β_i. Although 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 $\mathbf{x}_i$, whereas 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

$$\alpha_i = \beta_i = 0.4\|\Delta\mathbf{x}_i\|. \tag{9.22}$$

(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

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{\beta_i}{\alpha_{i+1}} = \frac{\|\Delta\mathbf{x}_i\|}{\|\Delta\mathbf{x}_{i+1}\|}. \tag{9.23}$$

A more sophisticated solution is the following: if we consider the planar curve in Figure 9.24, we see that it can be interpreted as a function, where the parameter t varies along the straight line through $\mathbf{b}_{3i}$ and $\mathbf{b}_{3i+3}$. Then

$$\|\Delta\mathbf{b}_{3i}\| = \frac{\|\mathbf{b}_{3i+3} - \mathbf{b}_{3i}\|}{3\cos\Theta_i},$$

$$\|\Delta\mathbf{b}_{3i+2}\| = \frac{\|\mathbf{b}_{3i+3} - \mathbf{b}_{3i}\|}{3\cos\Psi_{i+1}}.$$

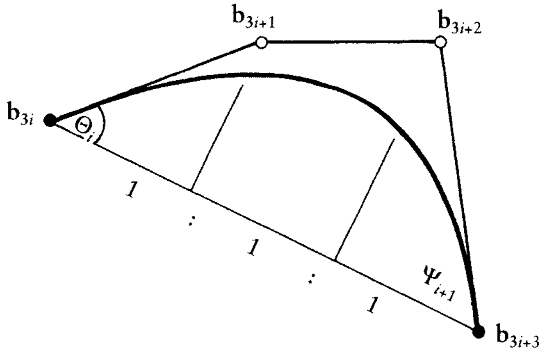


Figure 9.24 Inner Bézier points: this planar curve can be interpreted as a *function* in an oblique coordinate system with $\mathbf{b}_{3i}, \mathbf{b}_{3i+3}$ as the x-axis.

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 degrees, these expressions could be used 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° = 1/2, we can now make a case distinction:

$$\alpha_i = \begin{cases} \frac{\|\Delta\mathbf{x}_i\|^2}{3\mathbf{l}_i\Delta\mathbf{x}_i} & \text{if } |\Theta_i| \le 60^\circ \\ \frac{2}{3}\|\Delta\mathbf{x}_i\| & \text{otherwise} \end{cases} \tag{9.24}$$

and

$$\beta_i = \begin{cases} \frac{\|\Delta\mathbf{x}_i\|^2}{3\mathbf{l}_{i+1}\Delta\mathbf{x}_i} & \text{if } |\Psi_{i+1}| \le 60^\circ \\ \frac{2}{3}\|\Delta\mathbf{x}_i\| & \text{otherwise.} \end{cases} \tag{9.25}$$

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—(9.22)—does not preserve the ratios of the three points $\mathbf{b}_{3i-1}, \mathbf{b}_{3i}, \mathbf{b}_{3i+1}$ since

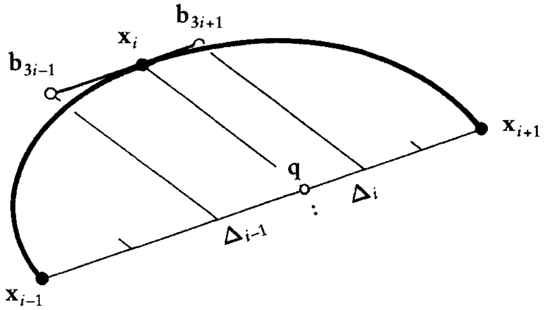


Figure 9.25 FMILL tangents: the tangent at $\mathbf{x}_i$ is parallel to the chord through $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$.

the ratios $\|\Delta\mathbf{x}_{i-1}\| : \|\Delta\mathbf{x}_i\|$ are not generally invariant under affine maps.[12] The second method uses angles, which are not preserved under affine transformations. However, both methods are invariant under euclidean transformations.

We now address a different version of piecewise cubic interpolation:

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 Section 9.4. Although those provide a global solution, a local one might be preferred for some applications where C^2 smoothness is not crucial. We are then faced with the problem of estimating tangents from the given data points and parameter values.

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 $\mathbf{x}_{i+1}$:

$$\mathbf{v}_i = \mathbf{x}_{i+1} - \mathbf{x}_{i-1}; \quad i = 1, \ldots, L-1. \tag{9.26}$$

Once the tangent direction $\mathbf{v}_i$ has been found,[13] the inner Bézier points are placed on it according to Figure 9.25:

[12] Recall that only the ratio of three *collinear* points is preserved under affine maps!

[13] Note that here we do not have $\|\mathbf{v}_i\| = 1$!

$$\mathbf{b}_{3i-1} = \mathbf{b}_{3i} - \frac{\Delta_{i-1}}{3(\Delta_{i-1} + \Delta_i)}\mathbf{v}_i, \tag{9.27}$$

$$\mathbf{b}_{3i+1} = \mathbf{b}_{3i} + \frac{\Delta_i}{3(\Delta_{i-1} + \Delta_i)}\mathbf{v}_i. \tag{9.28}$$

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.

The idea behind *Bessel tangents*[14] 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

$$\mathbf{m}_i = \frac{(1-\alpha_i)}{\Delta_{i-1}}\Delta\mathbf{x}_{i-1} + \frac{\alpha_i}{\Delta_i}\Delta\mathbf{x}_i; \quad i = 1, \ldots, L-1, \tag{9.29}$$

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 [452] (see also [91] and [154]). The i^{th} segment $\mathbf{s}_i$ of such a spline (defined over $[u_i, u_{i+1}]$) is defined by

$$\mathbf{s}_i(u) = \frac{u_{i+1} - u}{\Delta_i}\mathbf{q}_i(u) + \frac{u - u_i}{\Delta_i}\mathbf{q}_{i+1}(u); \quad i = 1, \ldots, L-2.$$

[14] They are also attributed to Ackland [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 Problems). A similar way of determining tangent vectors was developed by McConalogue [421], [422].

Finally, we mention a method created by H. Akima [3]. 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 curves are needed that oscillate only minimally.

9.9 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.
*/
```

In case 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[l], data_x[l+1] are not used! Same for data_y.
*/
```

A calling sequence that uses 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.

9.10 Problems

1 For the case of closed curves, C^1 quadratic spline interpolation with uniform knots does not always have a solution. Why?[15]

*2 Show that interpolating splines reproduce cubic polynomial curves—that they have *cubic precision.* This means that if all data points $\mathbf{p}_i$ are read off from a cubic, $\mathbf{p}_i = \mathbf{c}(u_i)$, and the end tangent vectors are read off from the cubic, then the interpolating spline equals the original cubic.

[15] T. DeRose pointed this out to me.

3 Show that Akima's interpolant always passes a straight line segment through three subsequent points if they happen to lie on a straight line.

***4** Show that Overhauser splines are piecewise cubics with Bessel tangents at the junction points.

***5** One can generalize the quintic Hermite interpolants from Section 7.6 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: [172], [260].

P2 Program the following: instead of prescribing end conditions for interpolating C^2 splines at both ends, prescribe first and second derivatives at τ_0. The interpolant can then be built segment by segment. Discuss the numerical aspects of this method (they will not be wonderful).

This Page Intentionally Left Blank

10

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 embedding into a surface.

10.1 Parametric Curves and Arc Length

A curve in $\mathbb{E}^3$ is given by the parametric representation

$$\mathbf{x} = \mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}, \quad t \in [a, b] \subset \mathbb{R}, \tag{10.1}$$

where its cartesian coordinates x, y, z are differentiable functions of t. (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

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} \neq 0, \quad t \in [a, b], \tag{10.2}$$

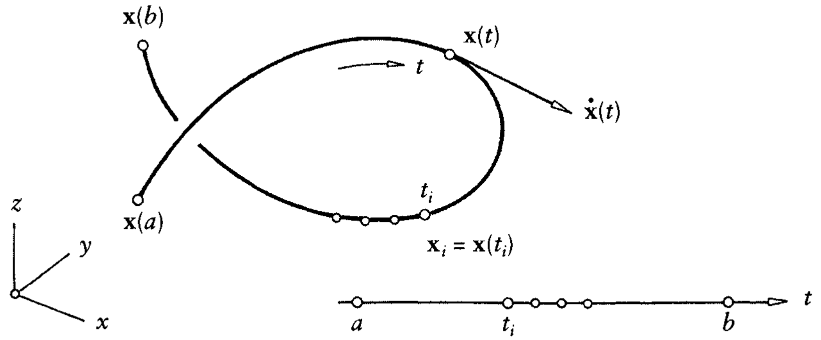


Figure 10.1 Parametric curve in space.

where dots denote derivatives with respect to t. Such a parametrization is called *regular*. An example is shown in Figure 10.1

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 $t \in [a, b]$, that is, we can find the inverse $t = t(\tau)$. Let

$$s = s(t) = \int_a^t \|\dot{\mathbf{x}}\| \mathrm{d}t \tag{10.3}$$

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

$$S = \sum_i \|\Delta \mathbf{x}_i\| = \sum_i \left\| \frac{\Delta \mathbf{x}_i}{\Delta t} \right\| \Delta t, \tag{10.4}$$

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 (10.4).

10.2 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 below do exist. The first terms of the Taylor expansion of $\mathbf{x}(t + \Delta t)$ at t are given by

$$\mathbf{x}(t + \Delta t) = \mathbf{x} + \dot{\mathbf{x}}\Delta t + \ddot{\mathbf{x}}\frac{1}{2}\Delta t^2 + \dddot{\mathbf{x}}\frac{1}{6}\Delta t^3 + \dots .^{1}$$

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*

$$\begin{bmatrix} \Delta t + \dots \\ \frac{1}{2}\Delta t^2 + \dots \\ \frac{1}{6}\Delta t^3 + \dots \end{bmatrix},$$

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 10.2:

$$\mathbf{t} = \frac{\dot{\mathbf{x}}}{\|\dot{\mathbf{x}}\|}, \qquad \mathbf{m} = \mathbf{b} \wedge \mathbf{t}, \qquad \mathbf{b} = \frac{\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}}{\|\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}\|}, \tag{10.5}$$

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.

[1] We use the abbreviation $\Delta t^2 = (\Delta t)^2$.

[2] One often sees the notation $\mathbf{n}$ for this vector. We use $\mathbf{m}$ to avoid confusion with surface normals, which are discussed later.

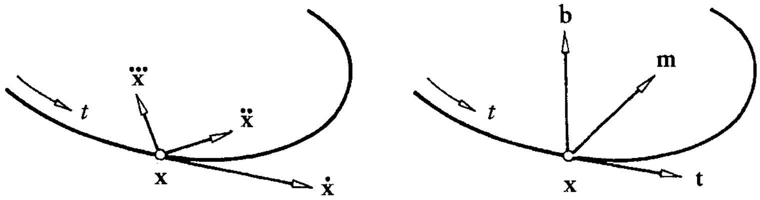


Figure 10.2 Local affine system (left) and Frenet frame (right).

The plane spanned by the point $\mathbf{x}$ and the two vectors $\mathbf{t}$, $\mathbf{m}$ is called the *osculating plane* 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 O. Its parametric form is

$$\mathrm{O}(u, v) = \mathbf{x} + u\dot{\mathbf{x}} + v\ddot{\mathbf{x}}.$$

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.

10.3 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, we find 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:

$$\begin{aligned} \mathbf{t}' &= \quad\quad +\kappa\mathbf{m} \\ \mathbf{m}' &= -\kappa\mathbf{t} \quad\quad\quad +\tau\mathbf{b}, \\ \mathbf{b}' &= \quad\quad -\tau\mathbf{m} \end{aligned} \tag{10.6}$$

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:

$$\begin{aligned} \kappa &= \kappa(s) = \|\mathbf{x}''\|, \\ \kappa &= \kappa(t) = \frac{\|\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}\|}{\|\dot{\mathbf{x}}\|^3}, \end{aligned} \tag{10.7}$$

$$\begin{aligned} \tau &= \tau(s) = \frac{1}{\kappa^2}\det[\mathbf{x}', \mathbf{x}'', \mathbf{x}'''], \\ \tau &= \tau(t) = \frac{\det[\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}}]}{\|\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}\|^2}. \end{aligned} \tag{10.8}$$

Figure 10.3 illustrates the formulas of (10.6).

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 $\ldots$ denotes terms of higher degree in Δs. Thus, when $\Delta s \to \mathrm{d}s$, we find that

$$\kappa = \frac{\mathrm{d}\alpha}{\mathrm{d}s}, \quad \tau = -\frac{\mathrm{d}\beta}{\mathrm{d}s}.$$

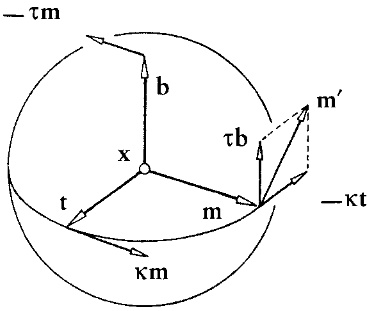


Figure 10.3 The geometric meaning of the Frenet–Serret formulas.

In other words, κ and $-\tau$ are the angular velocities of **t** and **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; that is, 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

$$\mathbf{x}(s+\Delta s) = \begin{bmatrix} \Delta s & & -\frac{1}{6}\kappa^2\Delta s^3 + \dots \\ & \frac{1}{2}\kappa\Delta s^2 & +\frac{1}{6}\kappa'\Delta s^3 + \dots \\ & & \frac{1}{6}\kappa\tau\Delta s^3 + \dots \end{bmatrix},$$

where ... again denotes terms of higher degree in Δs.

10.4 The Osculating Circle

The circle that has second-order contact with the curve at **x** is called the *osculating circle* (Figure 10.4). 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 Frenet–Serret formulas of (10.6), the Taylor expansion of $\mathbf{x}(s+\Delta s)$ can be written as

$$\mathbf{x}(s+\Delta s) = \mathbf{x}(s) + \mathbf{t}\Delta s + \frac{1}{2}\kappa\mathbf{m}\Delta s^2 + \dots.$$

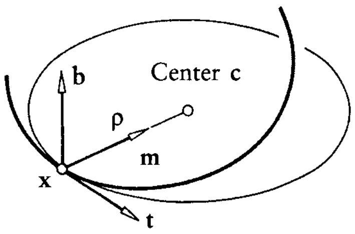


Figure 10.4 The osculating circle.

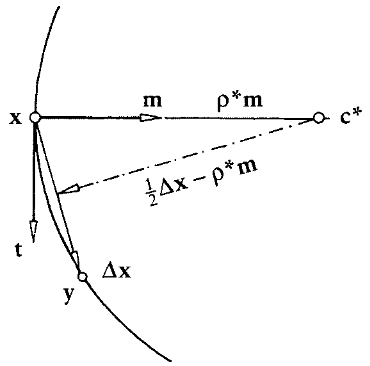


Figure 10.5 Construction of the osculating circle.

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 10.5). Note that $\mathbf{y}$ lies in the osculating plane O. Inspection of the figure reveals that $(\frac{1}{2}\Delta\mathbf{x} - \rho^*\mathbf{m})\Delta\mathbf{x} = 0$; that is, we obtain

$$\rho^* = \frac{1}{2}\frac{(\Delta\mathbf{x})^2}{\mathbf{m}\Delta\mathbf{x}}.$$

From the definition of $\Delta\mathbf{x}$, we obtain $(\Delta\mathbf{x})^2 = \Delta s^2 + \ldots$ and $\mathbf{m}\Delta\mathbf{x} = \frac{1}{2}\kappa(\Delta s)^2$. Thus $\rho^* = \frac{1}{\kappa} + \ldots$. 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 13. Its curvature and torsion at $\mathbf{b}_0$ are given by

$$\kappa = \frac{n-1}{n}\frac{w_0 w_2}{w_1^2}\frac{b}{a^2}, \quad \tau = \frac{n-2}{n}\frac{w_0 w_3}{w_1 w_2}\frac{c}{ab}, \tag{10.9}$$

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 10.6). 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 13.2).

Remark 7 An immediate application of (10.9) is the following: let $\mathbf{x}$ be a point on an integral quadratic Bézier curve, that is, a parabola. Let 2δ denote the length of a chord

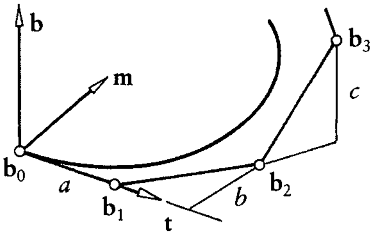


Figure 10.6 Frenet frame and geometric meaning of a, b, c.

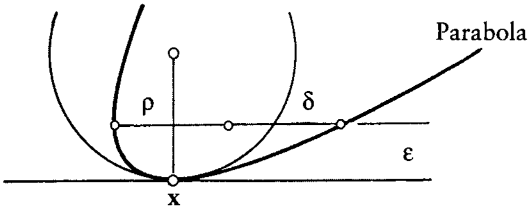


Figure 10.7 Curvature of a parabola.

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 10.7).

Remark 8 An equivalent way to formulate (10.9) is given by

$$\kappa = 2\frac{n-1}{n}\frac{w_0 w_2}{w_1^2}\frac{\text{area}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}{\text{dist}^3[\mathbf{b}_0, \mathbf{b}_1]} \qquad (10.10)$$

and

$$\tau = \frac{3}{2}\frac{n-2}{n}\frac{w_0 w_3}{w_1 w_2}\frac{\text{volume}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]}{\text{area}^2[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}. \qquad (10.11)$$

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.

10.5 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 this, we derive 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:

$$\kappa = \frac{\ddot{y}}{[1 + \dot{y}^2]^{\frac{3}{2}}},$$

and 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 (23.1).

Remark 10 For a nonparametric Bézier curve (see Section 6.5),

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

Where $u = u_0 + t\Delta u$ is a global parameter, we obtain

$$a = \frac{1}{n}\sqrt{\Delta u^2 + n^2(\Delta b_0)^2}, \qquad b = -\frac{\Delta u}{n}\frac{\Delta^2 b_0}{a},$$

as illustrated in Figure 10.8.

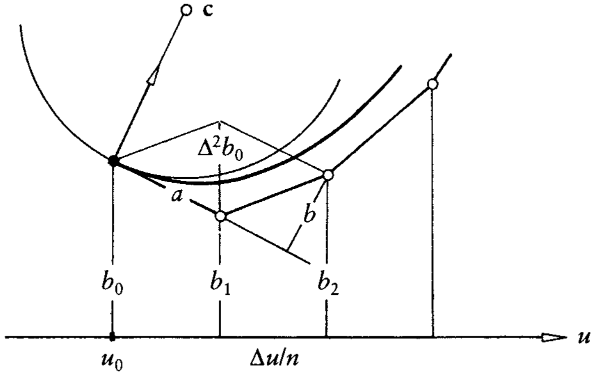


Figure 10.8 Curvature of nonparametric Bézier curve.

10.6 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

$$\dot{\mathbf{x}}_+ = \alpha \dot{\mathbf{x}}_-. \tag{10.12}$$

Moreover, it is curvature and osculating plane continuous if in addition

$$\ddot{\mathbf{x}}_+ = \alpha^2 \ddot{\mathbf{x}}_- + \alpha_{21} \dot{\mathbf{x}}_-, \tag{10.13}$$

and it is torsion continuous if in addition

$$\dddot{\mathbf{x}}_+ = \alpha^3 \dddot{\mathbf{x}}_- + \alpha_{32} \ddot{\mathbf{x}}_- + \alpha_{31} \dot{\mathbf{x}}_- \tag{10.14}$$

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,

$$\mathbf{x}''' = \mathbf{t}'' = \kappa' \mathbf{m} + \kappa(-\kappa \mathbf{t} + \tau \mathbf{b})$$

[3] Recall that $\kappa' = d\kappa(s)/ds$, where the prime denotes differentiation with respect to arc length s of the (composite) curve. A formula for κ' is provided by (23.2).

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; that is, 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

$$\ddot{\mathbf{x}}_+ = \alpha^2\ddot{\mathbf{x}}_- + \beta\dot{\mathbf{x}}_-,$$

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 rth 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 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, we find that a d-dimensional curve has $d - 1$ geometric invariants. Continuity of these invariants makes sense only 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 [76] and Goldman and Micchelli [267].

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 [88].

This Page Intentionally Left Blank

Geometric Continuity

11.1 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 11.1 shows three parabolas with junction points at the midpoints of an equilateral triangle. According to (10.10), 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 as is easily seen by sketching the second derivative vectors at the junction points.

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 10. 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 continuously varying binormal vectors and continuously varying curvature.

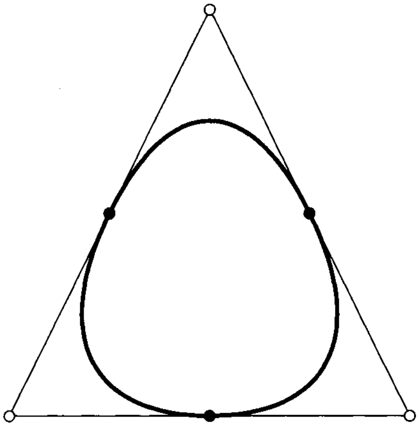


Figure 11.1 G^2 continuity: a closed quadratic G^2 spline curve.

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 pages, we present a unified treatment for most of these.

11.2 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 11.2, 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

$$r_- = \text{ratio}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{d}), \tag{11.1}$$

$$r_+ = \text{ratio}(\mathbf{d}, \mathbf{c}_1, \mathbf{c}_2), \tag{11.2}$$

$$r = \text{ratio}(\mathbf{b}_2, \mathbf{b}_3, \mathbf{c}_1). \tag{11.3}$$

[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.

[2] The G^2 conditions for general degrees will be identical, and so nothing is lost by concentrating on the cubic case.

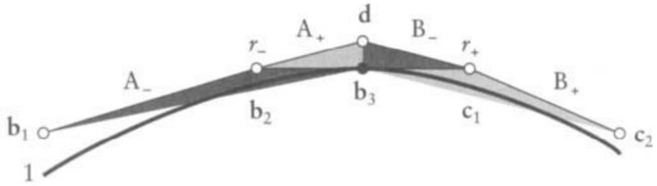


Figure 11.2 G^2 continuity: using the direct formulation.

Letting A_-, A_+, B_-, B_+ denote the triangle areas in Figure 11.2, we can invoke (10.10) in order to express the curvatures κ_- and κ_+ of the left and right segments at $\mathbf{b}_3$:

$$\kappa_- = \frac{4}{3}\frac{A_-}{\|\mathbf{b}_3 - \mathbf{b}_2\|^3}, \quad \kappa_+ = \frac{4}{3}\frac{A_+}{\|\mathbf{c}_1 - \mathbf{c}_0\|^3}.$$

If these two curvatures agree, we have that

$$\frac{A_-}{A_+} = r^3. \tag{11.4}$$

Referring to the figure again, we see that

$$\frac{A_-}{B_-} = r_-, \qquad \frac{B_+}{A_+} = r_+, \qquad \frac{B_-}{B_+} = r.$$

Inserting this into (11.4) yields our desired G^2 condition:

$$r^2 = r_- r_+. \tag{11.5}$$

With the notation of Figure 11.2, and setting $l_- = \|\mathbf{b}_3 - \mathbf{b}_2\|$, $l_+ = \|\mathbf{c}_1 - \mathbf{b}_3\|$, we have

$$\frac{\kappa_-}{\kappa_+} = \frac{A_-}{A_+}\frac{l_+^3}{l_-^3} = \frac{A_-}{A_+}\frac{B_+}{B_-}\frac{l_-}{l_+}\frac{l_+^3}{l_-^3}.$$

Thus

$$\frac{\kappa_-}{\kappa_+} = \frac{r_- r_+}{r^2}.$$

This is known as *Memke's theorem* and states that the ratio of left and right curvatures on any point of a curve is invariant under affine maps. This follows since we only use affinely invariant ratios in the formulation of κ_-/κ_+.

11.3 The γ, ν, and β Formulations

Using the setting of Section 11.2, we observe that our composite curve could be made C^1 if we introduced a knot sequence with interval lengths Δ_-, Δ_+ satisfying $\Delta_-/\Delta_+ = r$. Using (11.5), 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 11.3.

W. Boehm used this framework for his development of G^2 cubic splines; see [71].

If a C^1 curve $\mathbf{x}$ is curvature continuous at a point $\mathbf{x}(u)$, then we must have

$$\|\ddot{\mathbf{x}}_- \wedge \dot{\mathbf{x}}\| = \|\ddot{\mathbf{x}}_+ \wedge \dot{\mathbf{x}}\|.$$

Thus the three vectors involved must satisfy a relationship

$$\ddot{\mathbf{x}}_+ - \ddot{\mathbf{x}}_- = \nu\dot{\mathbf{x}}. \tag{11.6}$$

The constant ν depends on the parametrization of the curve; if we change u to ku, we will have to replace ν by $k\nu$. The ν formulation of G^2 continuity is due to G. Nielson [442].

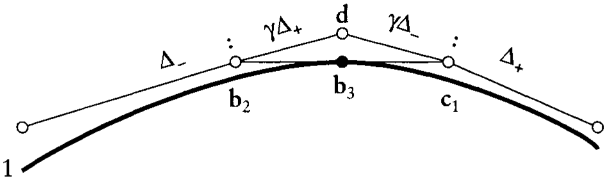


Figure 11.3 G^2 continuity: using the γ formulation.

There is a one-to-one relationship between the constants γ and ν:

$$\nu = 2\left(\frac{\Delta_- + \Delta_+}{\Delta_-\Delta_+}\right)\frac{1-\gamma}{\gamma}, \tag{11.7}$$

first found by W. Boehm [71].

A similar approach was taken by B. Barsky [40], [47];he uses

$$\beta_1 = \frac{\Delta_-}{\Delta_+} \quad \text{and} \quad \beta_2 = \nu \tag{11.8}$$

as the descriptors of G^2 continuity and calls them bias and tension, respectively.

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 Baer [12], Bézier [59], Geise [256], and Manning [414].

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.

Freeform curve design, on the other hand, will benefit more from the direct formulation since it is linked the closest 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.

11.4 G^2 Cubic Splines

We start with a control polygon $\mathbf{d}_0, \ldots, \mathbf{d}_L$. 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 11.4. Placing $\mathbf{b}_{3i-2}$ on the polygon leg $\overline{\mathbf{d}_i, \mathbf{d}_{i+1}}$ amounts to picking a number α_i (between 0 and 1) and then setting

$$\mathbf{b}_{3i-2} = (1-\alpha_i)\mathbf{d}_i + \alpha_i\mathbf{d}_{i+1}. \tag{11.9}$$

Similarly, we place $\mathbf{b}_{3i-1}$ by picking a number ω_i and setting

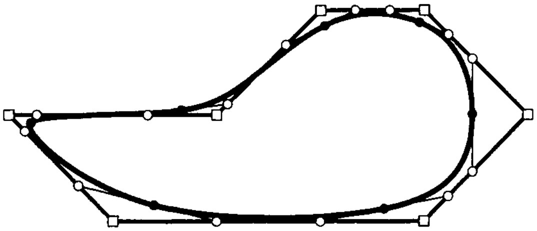


Figure 11.4 G^2 conditions: inner Bézier points may be placed on the control polygon legs. The junction points then may be found using the G^2 condition.

$$\mathbf{b}_{3i-1} = (1-\omega_i)\mathbf{d}_i + \omega_i\mathbf{d}_{i+1}. \tag{11.10}$$

In the same manner, by choosing numbers α_{i+1} and ω_{i+1}, 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 11.4 and 11.2, we see that we need the quantities $\lambda_i = \text{ratio}(\mathbf{b}_{3i-2}, \mathbf{b}_{3i-1}, \mathbf{d}_{i+1})$ and $\rho_i = \text{ratio}(\mathbf{d}_{i+1}, \mathbf{b}_{3i+1}, \mathbf{b}_{3i+2})$. Since

$$\mathbf{b}_{3i-1} = \frac{1-\omega_i}{1-\alpha_i}\mathbf{b}_{3i-2} + \frac{\omega_i - \alpha_i}{1-\alpha_i}\mathbf{d}_{i+1} \tag{11.11}$$

and

$$\mathbf{b}_{3i+1} = \frac{\omega_{i+1} - \alpha_{i+1}}{\omega_{i+1}}\mathbf{d}_{i+1} + \frac{\alpha_{i+1}}{\omega_{i+1}}\mathbf{b}_{3i+2}, \tag{11.12}$$

we have

$$\lambda_i = \frac{\omega_i - \alpha_i}{1-\omega_i}, \quad \rho_i = \frac{\alpha_{i+1}}{\omega_{i+1} - \alpha_{i+1}}. \tag{11.13}$$

Setting $r_i = \sqrt{\lambda_i\rho_i}/(1+\sqrt{\lambda_i\rho_i})$, we find the desired junction point to be

$$\mathbf{b}_{3i} = (1-r_i)\mathbf{b}_{3i-1} + r_i\mathbf{b}_{3i+1}. \tag{11.14}$$

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 (11.14).

For an open polygon, we set $\alpha_0 = 0$ and $\omega_{L-2} = 1$. This ensures the usual $\mathbf{b}_1 = \mathbf{d}_1$ and $\mathbf{b}_{3L-4} = \mathbf{d}_{L-1}$.

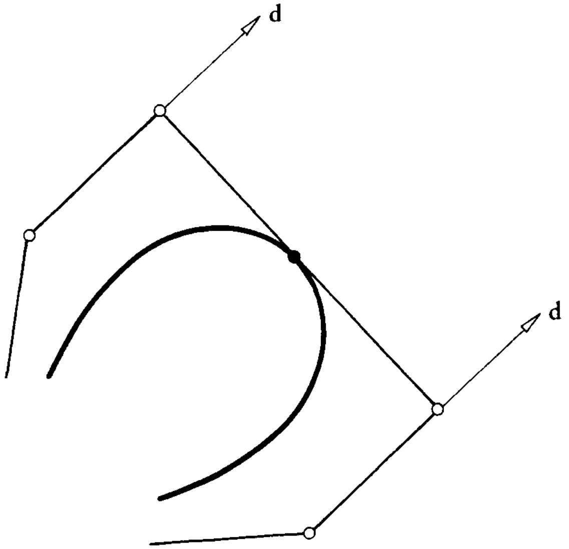


Figure 11.5 G^2 splines: these two cubics are a G^2 spline but do not possess a G^2 control polygon.

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 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 11.5 are curvature continuous, yet they cannot be obtained with the preceding construction: the control point **d** would have to be at infinity.

In interactive design, one would use 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 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 11.6 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 11.3. We then have

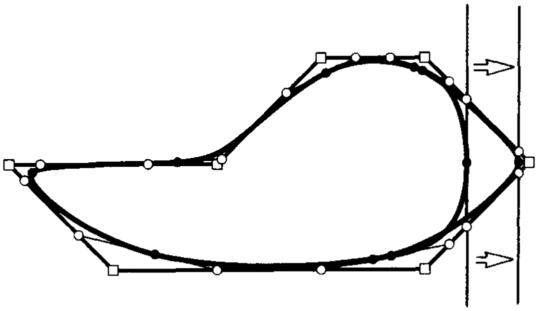


Figure 11.6 G^2 splines: a shape may be varied by prescribing tangents in addition to the control polygon.

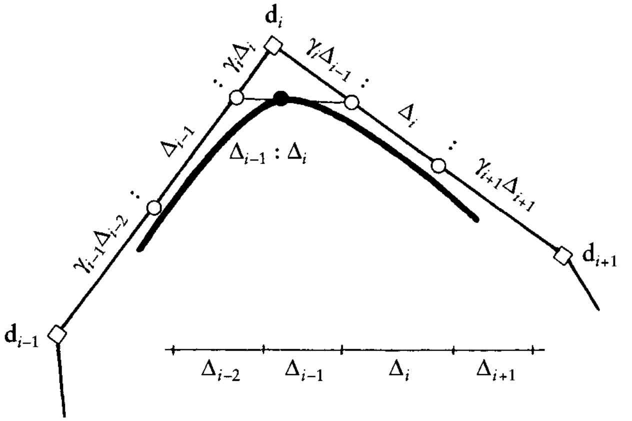


Figure 11.7 γ-splines: the Bézier points are connected to the G^2 control polygon by the shown ratios.

$$\alpha_{i-1} = \frac{\Delta_{i-1} + \gamma_i\Delta_i}{\gamma_i\Delta_{i-2} + \Delta_{i-1} + \gamma_{i+1}\Delta_i} \tag{11.15}$$

and

$$\omega_{i-1} = \frac{\gamma_i\Delta_i}{\gamma_i\Delta_{i-2} + \Delta_{i-1} + \gamma_{i+1}\Delta_i}. \tag{11.16}$$

The geometry of a γ-spline curve is shown in Figure 11.7. Note that for all $\gamma_i \to 0$, the curve will tend toward its control polygon.

11.5 Interpolating G^2 Cubic Splines

We may also use G^2 cubics to *interpolate* to given data points $\mathbf{x}_i; i = 0, \dots, L$. 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 $\mathbf{b}_{3i} = \mathbf{x}_i$, (11.14) yields

$$\begin{aligned}(\sqrt{\lambda_i} + \sqrt{\rho_i})\mathbf{x}_i = \sqrt{\rho_i}(1-\omega_i)\mathbf{d}_i &+ [\sqrt{\rho_i}\omega_i + \sqrt{\lambda_i}(1-\alpha_{i+1})] \\ &\mathbf{d}_{i+1} + \sqrt{\lambda_i}\alpha_{i+1}\mathbf{d}_{i+2}, i = 1, \dots, L-1. \qquad (11.17)\end{aligned}$$

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 $\mathbf{d}_1$ a linear combination of the first three data points: $\mathbf{d}_1 = u\mathbf{x}_0 + v\mathbf{x}_1 + w\mathbf{x}_2$. In our experience, $(u, v, w) = (5/6, 1/2, -1/3)$ has worked well. A similar equation then holds for $\mathbf{d}_{L+1}$. For the limiting case of $\alpha_i \to 0$ and $\omega_i \to 1$, the interpolating curve will approach the polygon formed by the data points. In terms of the γ formulation, this spline type was investigated in [209].

Nielson [442] derived the G^2 interpolating spline from the ν formulation. Assuming that the data points $\mathbf{x}_i$ have parameter values τ_i assigned to them, and using the piecewise cubic Hermite form, the interpolant becomes

$$\mathbf{x}(u) = \mathbf{x}_i H_0^3(r) + \mathbf{m}_i \Delta_i H_1^3(r) + \Delta_i \mathbf{m}_{i+1} H_2^3(r) + \mathbf{x}_{i+1} H_3^3(r), \qquad (11.18)$$

where the H_j^3 are cubic Hermite polynomials from (7.14) and $r = (u - \tau_i)/\Delta_i$ is the local parameter of the interval (τ_i, τ_{i+1}). In (11.18), the $\mathbf{x}_i$ are the known data points, whereas the $\mathbf{m}_i$ are as yet unknown tangent vectors. The interpolant is supposed to be G^2; it is therefore characterized by (11.6), more specifically,

$$\ddot{\mathbf{x}}_+(\tau_i) - \ddot{\mathbf{x}}_-(\tau_i) = \nu_i \mathbf{m}_i \qquad (11.19)$$

for some constants ν_i, where $\mathbf{m}_i = \dot{\mathbf{x}}(\tau_i)$. The ν_i are constants that can be used to manipulate the shape of the interpolant; they will be discussed soon. We insert (11.18) into (11.19) and obtain the linear system

$$\begin{aligned}3\left(\frac{\Delta_i \Delta\mathbf{x}_{i-1}}{\Delta_{i-1}} + \frac{\Delta_{i-1}\Delta\mathbf{x}_i}{\Delta_i}\right) &= \Delta_i \mathbf{m}_{i-1} + (2\Delta_{i-1} + 2\Delta_i + \frac{1}{2}\Delta_{i-1}\Delta_i\nu_i)\mathbf{m}_i \\ &+ \Delta_{i-1}\mathbf{m}_{i+1}; i = 1, \dots, L-1 \qquad (11.20)\end{aligned}$$

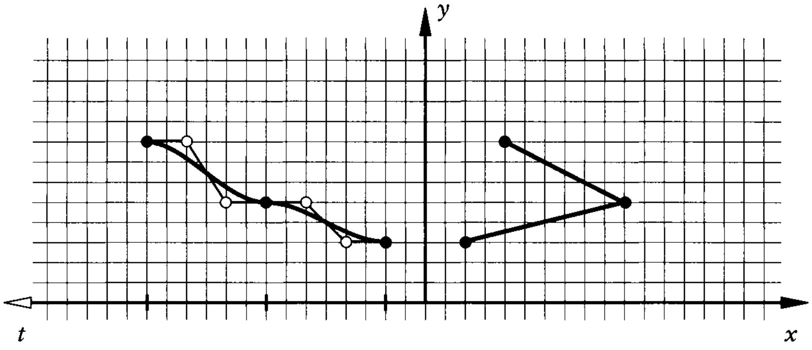


Figure 11.8 Infinite tension: an interpolating ν-spline curve with the cross plot of the y-component.

Together with two end conditions, (11.20) 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 τ_i; it is not invariant under affine parameter transformations.

If the ν_i are chosen to be nonnegative, the linear system (11.20) 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. Although the resulting curve will look piecewise linear, it is actually C^1, as shown in the cross plot (see Section 6.6) in Figure 11.8. Only the y-component is shown; it has zero slopes at the τ_i.

11.6 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 rth 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 10.6). In particular, the new parameter might be arc length.

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:

$$\frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_- = \frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_+; \quad i = 0, \ldots, r. \tag{11.21}$$

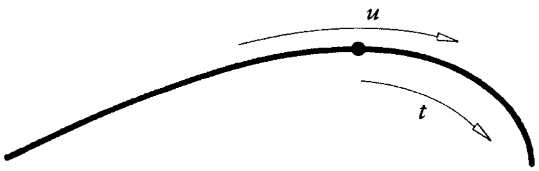


Figure 11.9 G^r continuity: a segment of a C^r curve may be reparametrized. The resulting curve is not C^r anymore, but still G^r.

Now let us reparametrize the right segment by introducing a new parameter $t = t(u)$; see Figure 11.9. 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 (11.21) so as to incorporate the new parametrization yields:

$$\frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}_- = \frac{\mathrm{d}^i}{\mathrm{d}u^i}\mathbf{x}(t)_+; \quad i = 0, \ldots, r. \tag{11.22}$$

The terms on the right-hand side of this equation may be expanded using the chain rule. For $i = 1$, we obtain

$$\mathbf{x}'_- = \dot{\mathbf{x}}_+ \frac{\mathrm{d}t}{\mathrm{d}u}, \tag{11.23}$$

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 (11.23):

$$\mathbf{x}_-'' = \ddot{\mathbf{x}}_+ \left(\frac{\mathrm{d}t}{\mathrm{d}u}\right)^2 + \dot{\mathbf{x}}_+ \frac{\mathrm{d}^2 t}{\mathrm{d}u^2}. \tag{11.24}$$

For the case $i = 3$:

$$\mathbf{x}_-''' = \dddot{\mathbf{x}}_+ \left(\frac{\mathrm{d}t}{\mathrm{d}u}\right)^3 + 3\ddot{\mathbf{x}}_+ \frac{\mathrm{d}t}{\mathrm{d}u}\frac{\mathrm{d}^2 t}{\mathrm{d}u^2} + \dot{\mathbf{x}}_+ \frac{\mathrm{d}^3 t}{\mathrm{d}u^3}. \tag{11.25}$$

Let us define $\alpha_i = \mathrm{d}^i t/\mathrm{d}u^i$. Then these equations may be written in matrix form:

$$\begin{bmatrix} \mathbf{x}'_- \\ \mathbf{x}_-'' \\ \mathbf{x}_-''' \end{bmatrix} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & \alpha_1^2 & 0 \\ \alpha_3 & 3\alpha_1\alpha_2 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_+ \\ \ddot{\mathbf{x}}_+ \\ \dddot{\mathbf{x}}_+ \end{bmatrix}. \tag{11.26}$$

The lower triangular matrix in (11.26) is called a *connection matrix*; it connects the derivatives of one segment to that of the other. For rth order geometric continuity, the connection matrix is a lower triangular $r \times r$ matrix; for more details, see Gregory [291] or Goodman [270]. See also the related discussion in Section 10.6. The connection matrix is a powerful theoretical tool, and has been used to derive variation diminishing properties of geometrically continuous curves (Dyn and Micchelli [181]), to show the projective invariance of torsion continuity (Boehm [76]), and for other theoretical pursuits (Goldman and Micchelli [267]).

The definition of geometric continuity has been used by Manning [414], Barsky [40], Barsky and DeRose [43], Degen [152], Pottmann [488], [489], and Farin [192]. In terms of classical differential geometry, the concept of G^2 is called order two of contact; see do Carmo [170]. It was used in a constructive context by G. Geise [256] 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 [192]). This more general class of curves, called *Frenet frame continuous*, has been studied by Boehm [74]; see also Section 10.6 and Hagen [298], [299]. 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, we have to resort to higher-dimensional spaces; this has been carried out by Dyn and Micchelli [181], Goodman [270], Goldman and Micchelli [267], and Pottmann [487]; see also the survey by Gregory [291]. An even more general concept than that of Frenet frame continuity has been discussed recently by H. Pottmann [488].

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

$$\frac{\text{volume}[\mathbf{b}_{n-3}, \ldots, \mathbf{b}_n]}{\|\Delta \mathbf{b}_{n-1}\|^6} = \frac{\text{volume}[\mathbf{c}_0, \ldots, \mathbf{c}_3]}{\|\Delta \mathbf{c}_0\|^6}. \tag{11.27}$$

See Boehm [73], Farin [192], or Hagen [298].

A nice geometric interpretation of the fact that torsion continuity is more general than G^3 continuity is due to W. Boehm [73]. 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!

11.7 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!
*/
```

11.8 Problems

1 Figure 11.1 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 11.5 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.

This Page Intentionally Left Blank

Conic Sections 12

Conic sections (or, simply, 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; Liming [390] and [391] are two books with detailed descriptions of those methods. A thorough development of conics can also be found in [85] and [202].

The first person to consider conics in a CAD environment was S. Coons [124]. Later, Forrest [240] further investigated conics and also rational cubics. We shall treat conics in the rational Bézier form; a good reference for this approach is Lee [377]. 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.

12.1 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 [461] or to [85] and [202]. All we need here is the concept of a projective map.

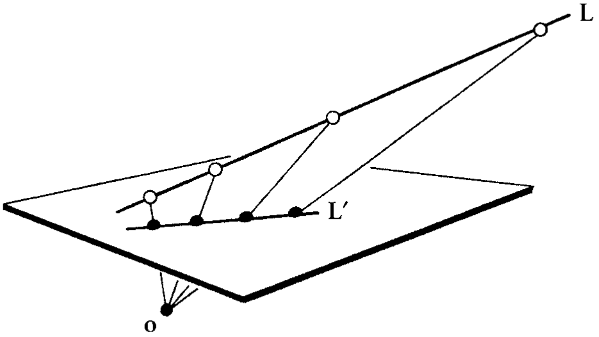


Figure 12.1 Projections: a straight line L is mapped onto another straight line L′ by a projection. Note how ratios of corresponding triples of points are distorted.

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 the intersection $\hat{\mathbf{p}}$ between the straight line through $\mathbf{o}$ and $\mathbf{p}$ with $\mathbf{P}$. For a projection to be well defined, it is necessary that $\mathbf{o}$ is not in $\mathbf{P}$. Any object in $\mathbb{E}^3$ can be projected into $\mathbf{P}$ in this manner.

In particular, we can project a straight line, L, say, onto P, as shown in Figure 12.1. 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 (3.6)]:

$$\mathrm{cr}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = \frac{\mathrm{ratio}(\mathbf{a},\mathbf{b},\mathbf{d})}{\mathrm{ratio}(\mathbf{a},\mathbf{c},\mathbf{d})}. \tag{12.1}$$

This particular definition is only one of several equivalent ones; any permutation of the four points gives rise to a valid definition. Our convention (12.1) 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 [429].

Let us now prove this invariance claim. We have to show, with the notation from Figure 12.2, that

$$\mathrm{cr}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) = \mathrm{cr}(\hat{\mathbf{a}},\hat{\mathbf{b}},\hat{\mathbf{c}},\hat{\mathbf{d}}). \tag{12.2}$$

This fact is called the *cross ratio theorem.*

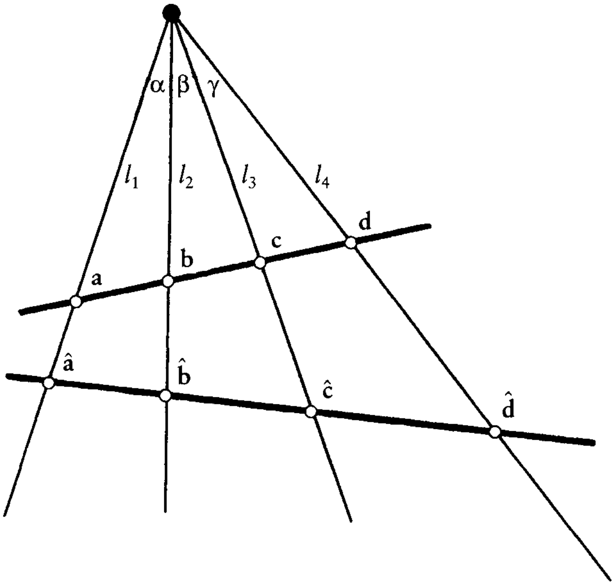


Figure 12.2 Cross ratios: the cross ratios of **a**, **b**, **c**, **d** and **â**, $\hat{\mathbf{b}}$, **ĉ**, $\hat{\mathbf{d}}$ only depend on the angles shown and are thus equal.

For a proof, consider Figure 12.2.

Denote the area of a triangle with vertices **p**, **q**, **r** by $\Delta(\mathbf{p}, \mathbf{q}, \mathbf{r})$. We note that, for instance,

$$\text{ratio}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \Delta(\mathbf{a}, \mathbf{b}, \mathbf{o}) / \Delta(\mathbf{b}, \mathbf{c}, \mathbf{o}).$$

This gives

$$\begin{aligned} \text{cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) &= \frac{\Delta(\mathbf{a}, \mathbf{b}, \mathbf{o}) / \Delta(\mathbf{b}, \mathbf{d}, \mathbf{o})}{\Delta(\mathbf{a}, \mathbf{c}, \mathbf{o}) / \Delta(\mathbf{c}, \mathbf{d}, \mathbf{o})} \\ &= \frac{l_1 l_2 \sin \alpha / l_2 l_4 \sin(\beta + \gamma)}{l_1 l_3 \sin(\alpha + \beta) / l_3 l_4 \sin \gamma} \\ &= \frac{\sin \alpha / \sin(\beta + \gamma)}{\sin(\alpha + \beta) / \sin \gamma}. \end{aligned}$$

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 12.1, we can interpret both L and L′ as copies of the real line. Then the projection of L onto 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 12.2. The claim is that **a**, **b**, **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 **c** on L, its image $\hat{\mathbf{c}}$ on 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 $\hat{t}$:

$$\hat{t} = \frac{(t-a)\hat{\rho}\hat{c} + (c-t)\hat{a}\rho}{\rho(c-t) + \hat{\rho}(t-a)}. \tag{12.3}$$

A convenient choice for the image and preimage points is $a = \hat{a} = 0, c = \hat{c} = 1$. Equation (12.3) then takes on the simpler form

$$\hat{t} = \frac{t\hat{\rho}}{\rho(1-t) + \hat{\rho}t}. \tag{12.4}$$

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 $\rho = \hat{\rho}$.

12.2 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 12.3 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 e, 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,

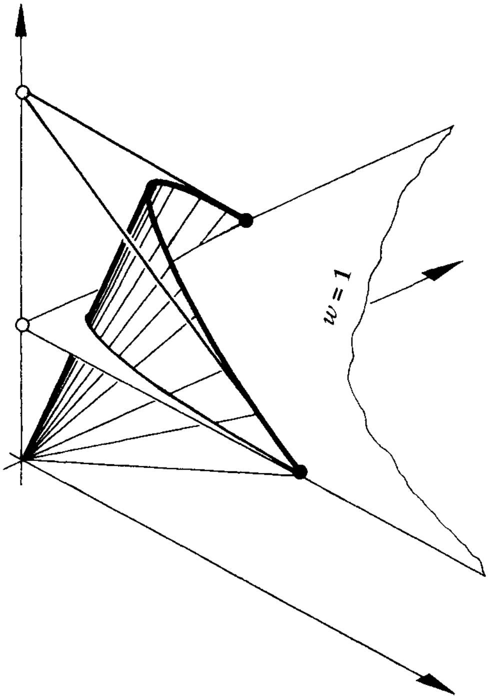
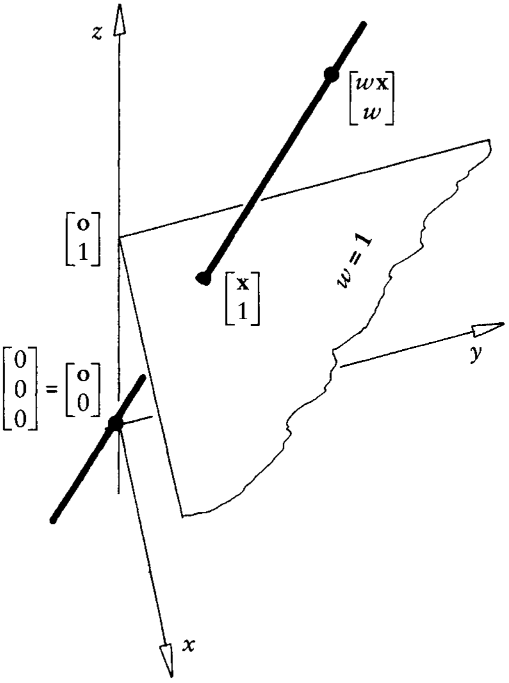


Figure 12.3 Conic sections: a parabolic arc in 3D space is projected into the plane $z = 1$; the result, in this example, is part of a hyperbola.



z
$\begin{bmatrix} w\mathbf{x} \\ w \end{bmatrix}$
$\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$
$w = 1$
$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$
y
$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}$
x

Figure 12.4 Projections: the special projection that is used to write objects in the plane $z = 1$ as projections of objects in $\mathbb{E}^3$.

we 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 12.4.

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

$$\mathbf{c}(t) = \frac{w_0\mathbf{b}_0B_0^2(t) + w_1\mathbf{b}_1B_1^2(t) + w_2\mathbf{b}_2B_2^2(t)}{w_0B_0^2(t) + w_1B_1^2(t) + w_2B_2^2(t)}. \tag{12.5}$$

Let us prove (12.5). 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

[1] The set of all points $[\,wx \quad wy \quad w\,]^{\mathrm{T}}$ is called the *homogeneous form* or *homogeneous coordinates* of $[\,x \quad y\,]^{\mathrm{T}}$.

may be expressed in Bernstein form:

$$w(t) = w_0 B_0^2(t) + w_1 B_1^2(t) + w_2 B_2^2(t).$$

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

$$\sum_{i=0}^{2} \mathbf{p}_i B_i^2(t) = \mathbf{c}(t) \sum_{i=0}^{2} w_i B_i^2(t), \tag{12.6}$$

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 (12.5).

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 (12.5) is called the *rational quadratic form* of a conic section. If all weights are equal, we recover nonrational quadratics, that is, parabolas. The influence of the weights on the shape of the conic is illustrated in Figure 12.5. 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 $w_0 \neq 0$, we may therefore always achieve $w_0 = 1$ by a simple scaling of all w_i.

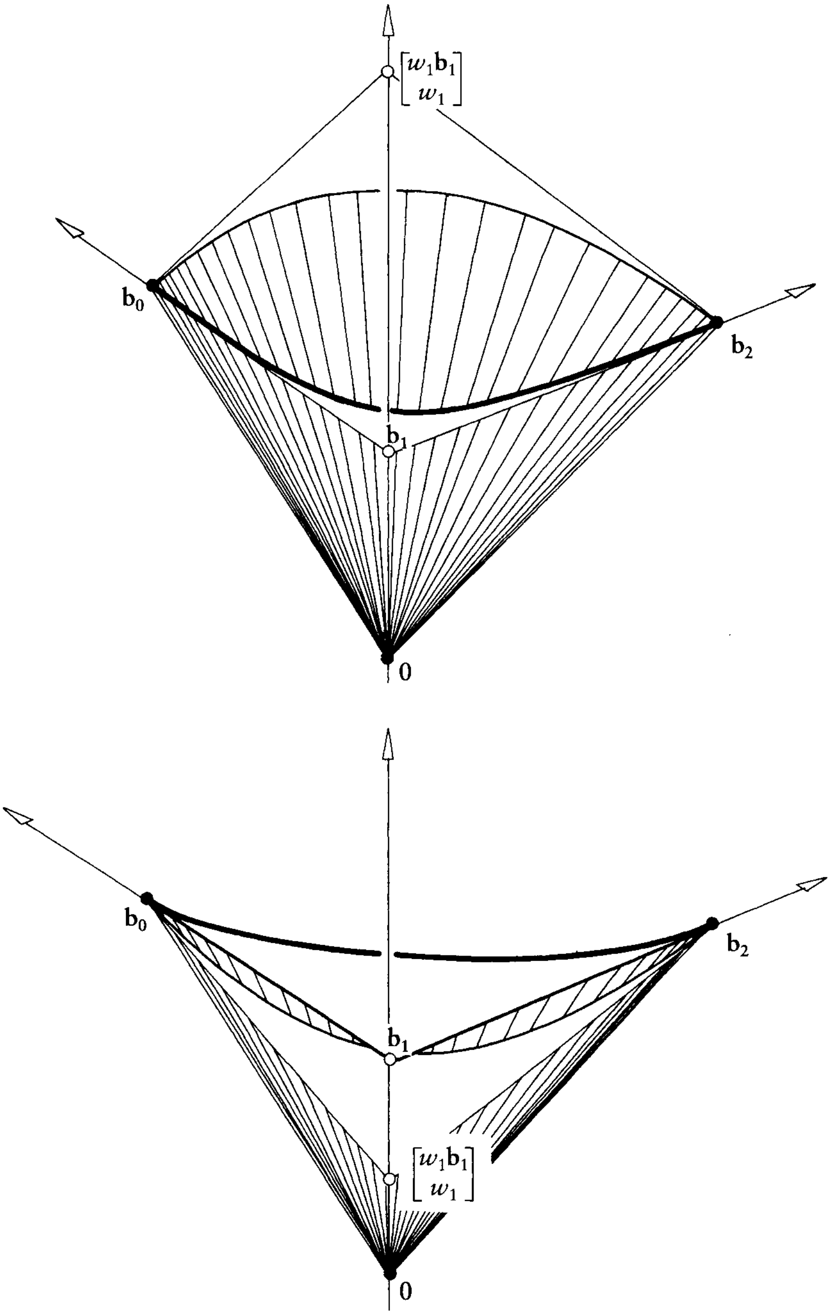

$\begin{bmatrix} w_1\mathbf{b}_1 \\ w_1 \end{bmatrix}$
$\mathbf{b}_0$
$\mathbf{b}_2$
$\mathbf{b}_1$
0
$\mathbf{b}_0$
$\mathbf{b}_2$
$\mathbf{b}_1$
$\begin{bmatrix} w_1\mathbf{b}_1 \\ w_1 \end{bmatrix}$
0

Figure 12.5 Conic sections: in the two examples shown, $w_0 = w_2 = 1$. As w_1 becomes larger, that is, as $[w_1\mathbf{b}_1, w_1]$ moves "up" on the z-axis, the conic is "pulled" toward $\mathbf{b}_1$.

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 (12.4)]. We may insert this into (12.5) and obtain:

$$\mathbf{c}(\hat{t}) = \frac{\hat{\rho}^2 w_0 \mathbf{b}_0 B_0^2(\hat{t}) + \hat{\rho} w_1 \mathbf{b}_1 B_1^2(\hat{t}) + w_2 \mathbf{b}_2 B_2^2(\hat{t})}{\hat{\rho}^2 w_0 B_0^2(\hat{t}) + \hat{\rho} w_1 B_1^2(\hat{t}) + w_2 B_2^2(\hat{t})}. \qquad (12.7)$$

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 [240]). 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 choice of $\hat{\rho}$, provided, of course, that $w_2/w_0 > 0$.

If in standard form, that is, $w_0 = w_2 = 1$, 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

$$\text{ratio}(\mathbf{m}, \mathbf{s}, \mathbf{b}_1) = w_1. \qquad (12.8)$$

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 12.6. The proof of this four tangent theorem is simple: one shows that it is true for parabolas (see Problems). 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 4.1.

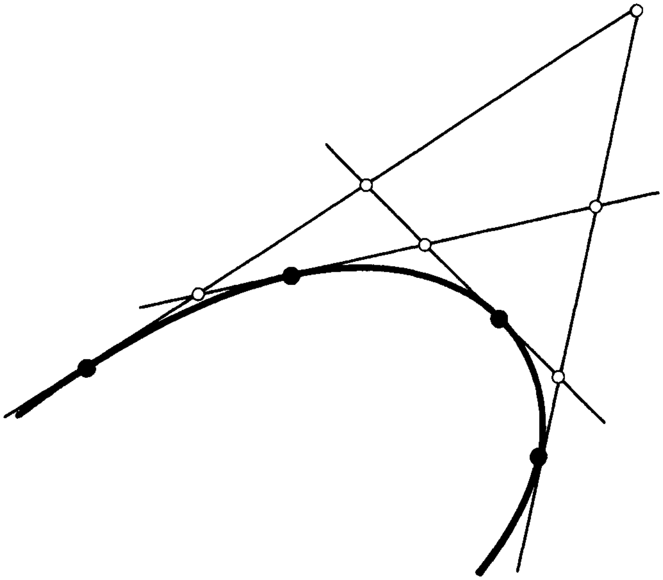


Figure 12.6 The four tangent theorem: four points are marked on each of the four tangents to the shown conic. The four cross ratios generated by them are all equal.

12.3 A de Casteljau Algorithm

We may evaluate (12.5) 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 $\mathbb{E}^2$:

$$\mathbf{b}_i^r(t) = (1-t)\frac{w_i^{r-1}}{w_i^r}\mathbf{b}_i^{r-1} + t\frac{w_{i+1}^{r-1}}{w_i^r}\mathbf{b}_{i+1}^{r-1}, \qquad (12.9)$$

where

$$w_i^r(t) = (1-t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t). \qquad (12.10)$$

This algorithm has a strong connection to the four tangent theorem: if we introduce *weight points*

$$\mathbf{q}_i^r(t) = \frac{w_i^r \mathbf{b}_i^r + w_{i+1}^r \mathbf{b}_{i+1}^r}{w_i^r + w_{i+1}^r}, \qquad (12.11)$$

then

$$\mathrm{cr}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_i^{r+1}, \mathbf{b}_{i+1}^r) = \frac{1-t}{t} \tag{12.12}$$

assumes the same value for all r, i. Though 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 $t \in [0, 1]$.

12.4 Derivatives

To find the derivative of a conic section, that is, 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 (12.6) 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 $\dot{\mathbf{c}}(t)$:

$$\dot{\mathbf{c}}(t) = \frac{1}{w(t)}[\dot{\mathbf{p}}(t) - \dot{w}(t)\mathbf{c}(t)]. \tag{12.13}$$

We may evaluate (12.13) 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

$$\dot{\mathbf{c}}(0) = \frac{2w_1}{w_0}\Delta\mathbf{b}_0. \tag{12.14}$$

Similarly, we obtain

$$\dot{\mathbf{c}}(1) = \frac{2w_1}{w_2}\Delta\mathbf{b}_1. \tag{12.15}$$

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

$$\frac{w_1}{\Delta_0}\Delta\mathbf{b}_1 = \frac{w_3}{\Delta_1}\Delta\mathbf{b}_2, \tag{12.16}$$

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.

12.5 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 $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$:

$$\mathbf{c}(t) = \tau_0\mathbf{b}_0 + \tau_1\mathbf{b}_1 + \tau_2\mathbf{b}_2; \tag{12.17}$$

see Section 3.5. Since $\mathbf{c}(t)$ may also be written as a rational Bézier curve (12.5), and since both representations are unique, we may compare the coefficients of the $\mathbf{b}_i$:

$$\tau_0 = [w_0(1-t)^2]/D, \tag{12.18}$$

$$\tau_1 = [2w_1t(1-t)]/D, \tag{12.19}$$

$$\tau_2 = [w_2t^2]/D, \tag{12.20}$$

where $D = \sum w_iB_i^2$. We may solve (12.18) and (12.20) for $(1-t)$ and t, respectively. Inserting both expressions into (12.19) yields

$$\tau_1^2 = 4\frac{\tau_0\tau_2w_1^2}{w_0w_2}.$$

This may be written more symmetrically as

$$\frac{\tau_1^2}{\tau_0\tau_2} = \frac{4w_1^2}{w_0w_2}. \tag{12.21}$$

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 (12.5). 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 (12.21). If (12.21) is satisfied, $\mathbf{x}$ lies on the conic (but see Problems).

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 point 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 $\nabla^{\perp} f(\mathbf{b}_0) = [-f_y(\mathbf{b}_0), f_x(\mathbf{b}_0)]^{\mathrm{T}}$. Thus our tangents are given by

$$\mathbf{t}_0(t) = \mathbf{b}_0 + t\nabla^{\perp} f(\mathbf{b}_0) \quad \text{and}$$
$$\mathbf{t}_2(s) = \mathbf{b}_2 + s\nabla^{\perp} f(\mathbf{b}_2).$$

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 (12.8), 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 $\mathbf{b}_0$

[2] The quadratic equation will in general have two solutions. We take the one inside the triangle $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$.

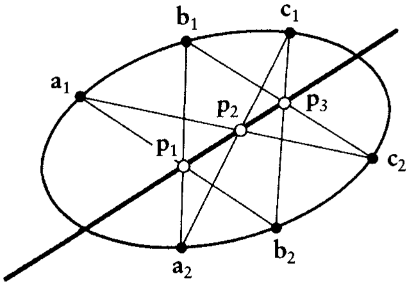


Figure 12.7 Pascal's theorem: the intersection points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ of the indicated pairs of straight lines are collinear.

and $\mathbf{b}_2$ subtends an angle larger than, say, 120 degrees, it should be subdivided. For more details, see [619].

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 12.7. If we connect points as shown, 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 12.7 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 $\mathbf{p}_1$.

12.6 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 the book by R. Liming [391]. 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 (12.21). 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 (12.21) for the unknown weight w_1:

$$w_1 = \frac{\tau_1}{2\sqrt{\tau_0 \tau_2}}. \tag{12.22}$$

If $\mathbf{p}$ is inside the triangle formed by $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$, then (12.22) always has a solution. Otherwise, problems might occur (see Problems). If we do not insist on the conic in standard form, the given point may be given the parameter value $t = 1/2$, 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 12.8.

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 (12.11), 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 (12.12) now yields

$$\frac{r_0}{w_1} = r_1 w_1, \tag{12.23}$$

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 Problems). Again, if we do not insist on standard form, we may associate the parameter value $t = 1/2$ with the given tangent—it is then called a *shoulder tangent.*

Figure 12.8 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 $\mathbf{p}$.

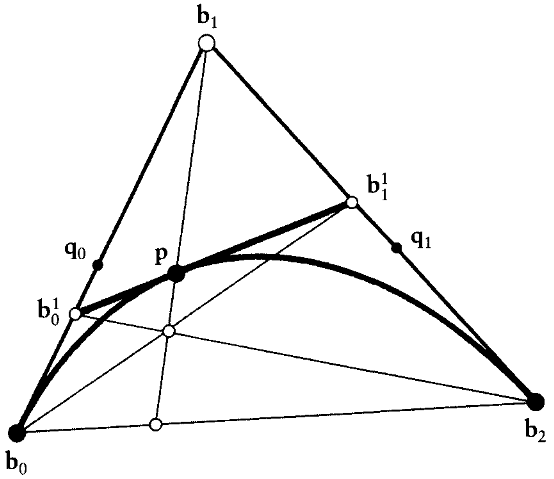


Figure 12.8 Conic constructions: b_0, b_1, b_2, and the tangent through b_0^1 and b_1^1 are given.

12.7 Classification

In a projective environment, all conics are equivalent: projective maps map conics to conics. In affine geometry, conics fall into three classes, namely, 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 [377]), 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 (12.21) is not affected by this; hence we still have the same conic, but with a different representation. If $c(t)$ is a point on the original conic and $\hat{c}(t)$ is a point on the complementary segment, one easily verifies that b_1, $c(t)$, and $\hat{c}(t)$ are collinear, as shown in Figure 12.9. If we assume that $w_1 > 0$, then the behavior of $\hat{c}(t)$ determines what type the conic is: if $\hat{c}(t)$ 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 $\hat{c}(t)$, are determined by the real roots of the denominator $\hat{w}(t)$ of $\hat{c}(t)$. There are at most two real roots, and they are given by

$$t_{1,2} = \frac{1 + w_1 \pm \sqrt{w_1^2 - 1}}{2 + 2w_1}.$$

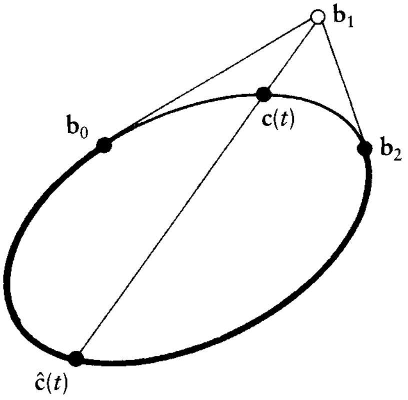


Figure 12.9 The complementary segment: the original conic segment and the complementary segment, both evaluated for all parameter values $t \in [0, 1]$, comprise the whole conic section.

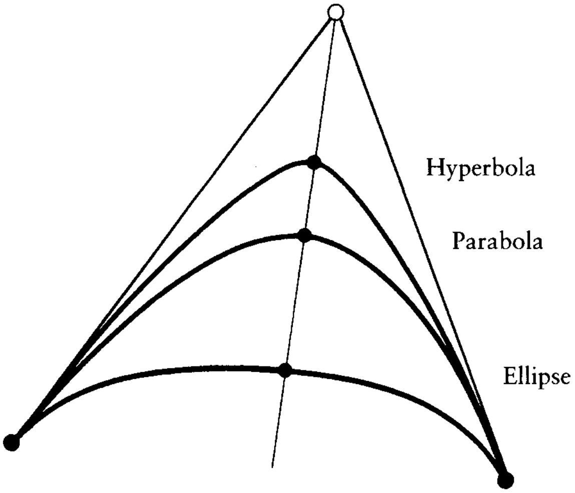


Figure 12.10 Conic classification: the three types of conics are obtained by varying the center weight w_1, assuming $w_0 = w_2 = 1$.

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 12.10 (see also Figure 12.5).

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

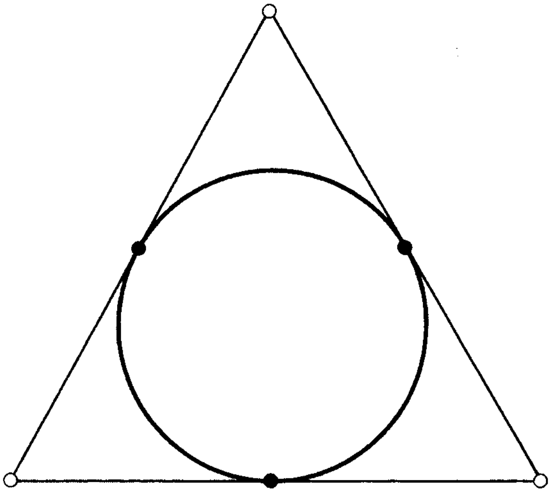


Figure 12.11 Circles: a whole circle may be written as three rational Bézier quadratics.

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 use the solution to the second problem in Section 12.6 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 12.11. The angles α equal 60 degrees, and so the weights of the inner Bézier points are 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 (12.16). 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 10. If uniform traversal of the circle is necessary for some application, we have no choice but to resort to the classical sine and cosine representation. It can be shown (Farouki and Sakkalis [226]) that no rational curve other than the straight line is parametrized with respect to arc

[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!

length: when evaluated at equal increments of its parameter t, the curve will not be traced out at uniform speed.

12.8 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 12.12.

The equation of our conic is given by

$$\mathbf{c}(t) = \frac{(1-t)^2\begin{bmatrix}-1\\0\end{bmatrix} + \cos\alpha \cdot 2ct(1-t)\begin{bmatrix}0\\\tan\alpha\end{bmatrix} + t^2\begin{bmatrix}1\\0\end{bmatrix}}{(1-t)^2 + 2ct(1-t)\cos\alpha + t^2}.$$

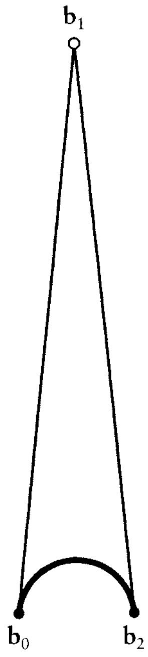


Figure 12.12 Conic arcs: a 168-degree arc of a circle is shown. Note that α is close to 90 degrees.

What happens as α tends to $\pi/2$? For the limiting conic, we obtain the equation

$$\mathbf{c}(t) = \frac{(1-t)^2\begin{bmatrix}-1\\0\end{bmatrix} + 2t(1-t)\begin{bmatrix}0\\c\end{bmatrix} + t^2\begin{bmatrix}1\\0\end{bmatrix}}{(1-t)^2+t^2}. \tag{12.24}$$

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 (12.24) 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 (12.24) contains both control points and control vectors.[4] An important property of Bézier curves is thus lost, namely, the convex hull property: it is defined only for point sets, not for a potpourri of points and vectors.

The use of control vectors allows a very compact form of writing a semicircle. 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 13.6) how to avoid the use of the control vector form.

12.9 Implementation

The following routine solves the first problem in Section 12.6:

```
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.
*/
```

[4] In projective geometry, vectors are sometimes called points at infinity. This has given rise to the name *infinite control points* by Versprille [601]; see also L. Piegl [477]. We prefer the term *control vector* since this allows us to distinguish between $[0, c]^{\mathrm{T}}$ and $[0, -c]^{\mathrm{T}}$.

12.10 **Problems**

1 Equation (12.22) 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 (12.23).

3 Prove that the four tangent theorem holds for parabolas.

*4 Establish the connection between (12.12) and the four tangent theorem.

*5 Our discussion of the implicit form (12.21) was somewhat academic: in a "real-life" situation, (12.21) will never be satisfied *exactly*. Discuss the tolerance problem that arises here; that is, how closely does (12.21) 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 initial conic? (See also [403].)

P2 Write a routine to approximate a given Bézier curve by a sequence of elliptic arcs within a given tolerance.

This Page Intentionally Left Blank

Rational Bézier and B-Spline Curves 13

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 to Coons [124], [126], and Forrest [240]. By now, there are books on NURBS: Fiorot and Jeannin [233], Farin [202], Piegl and Tiller [482].

13.1 Rational Bézier Curves

In Chapter 12, 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 nth 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 $\begin{bmatrix} x & y & z & w \end{bmatrix}^{\mathrm{T}}$. Proceeding in exactly the same way as we did for conics, we can show that an nth degree rational Bézier curve is given by

$$\mathbf{x}(t) = \frac{w_0\mathbf{b}_0 B_0^n(t) + \cdots + w_n\mathbf{b}_n B_n^n(t)}{w_0 B_0^n(t) + \cdots + w_n B_n^n(t)}; \quad \mathbf{x}(t), \mathbf{b}_i \in \mathbb{E}^3. \tag{13.1}$$

The w_i are again called *weights*; the $\mathbf{b}_i$ form the control polygon. It is the projection of the 4D control polygon $\begin{bmatrix} w_i\mathbf{b}_i & w_i \end{bmatrix}^{\mathrm{T}}$ 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 (13.1) 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 13.1. Note that the effect of changing a weight is different from that of moving a control vertex. 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 have this property only for a subset of all projective maps, that is, 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:

$$\mathbf{b}_i = (1 - \alpha_i)\mathbf{b}_0 + \alpha_i \mathbf{b}_n; \quad i = 0, \ldots, n$$

[2] This is also true if the weights are not unity, but are equal to each other—a common factor does not matter.

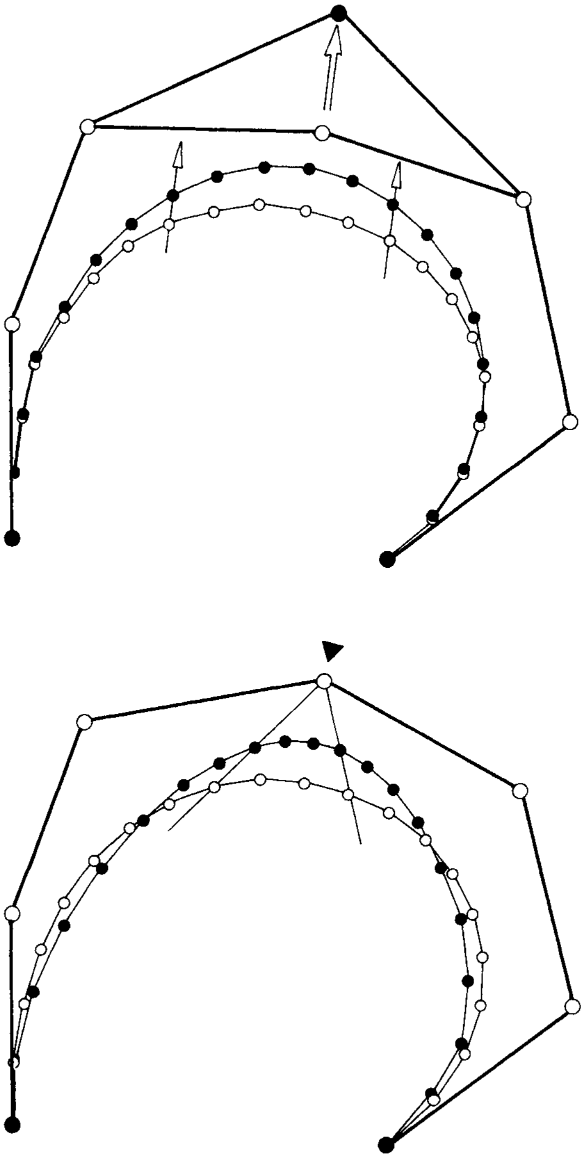


Figure 13.1 Influence of the weights: top, changing one control point; bottom, changing one weight.

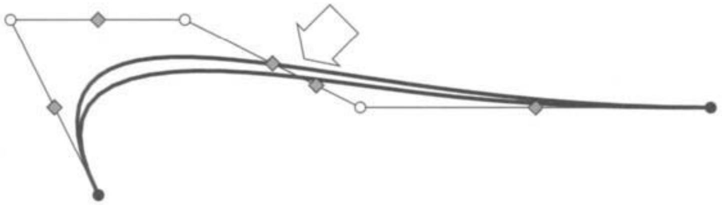


Figure 13.2 Weight points: changing one weight point influences the shape of the curve. Weight points are marked by diamonds.

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 [211] and [234].

Rational Bézier curves may be modified in another way. Let us define *weight points* $\mathbf{q}_i$ by setting

$$\mathbf{q}_i = \frac{w_i\mathbf{b}_i + w_{i+1}\mathbf{b}_{i+1}}{w_i + w_{i+1}}. \tag{13.2}$$

These points are defined via the weights; they may be used as shape parameters for the curve. If we change the location of one of the $\mathbf{q}_i$, we may recompute a new set of weights by setting $w_0 = 1$ and using (13.2) as a recursion for the w_i. Figure 13.2 illustrates.

13.2 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: though 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]

[3] These points are obtained by applying the de Casteljau algorithm to the control points $w_i\mathbf{b}_i$ of the numerator of (13.1). They have no true geometric interpretation because their location is not invariant under translations of the original control polygon.

An expensive yet more geometric technique is to project every intermediate de Casteljau point $\begin{bmatrix} w_i\mathbf{b}_i & w_i \end{bmatrix}^{\mathrm{T}}$; $\mathbf{b}_i \in \mathbb{E}^3$ into the hyperplane $w = 1$. This yields the rational de Casteljau algorithm (see Farin [193]):

$$\mathbf{b}_i^r(t) = (1-t)\frac{w_i^{r-1}}{w_i^r}\mathbf{b}_i^{r-1} + t\frac{w_{i+1}^{r-1}}{w_i^r}\mathbf{b}_{i+1}^{r-1}, \tag{13.3}$$

with

$$w_i^r(t) = (1-t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t). \tag{13.4}$$

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

$$\operatorname{ratio}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_{i+1}^r) = \frac{w_{i+1}^r}{w_i^r}.$$

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:

$$\operatorname{cr}(\mathbf{b}_i^r, \mathbf{q}_i^r, \mathbf{b}_i^{r+1}, \mathbf{b}_{i+1}^r) = \frac{1-t}{t} \quad \text{for all } r, i.$$

For $r = 0$, the weight points

$$\mathbf{q}_i = \mathbf{q}_i^0 = \frac{w_i\mathbf{b}_i + w_{i+1}\mathbf{b}_{i+1}}{w_i + w_{i+1}}$$

are directly related to the weights w_i: given the weights, we can find the $\mathbf{q}_i$, and given the $\mathbf{q}_i$, we can find the weights w_i.[4] Thus the $\mathbf{q}_i$ may be used as *shape*

[4] To be precise, we can only find them modulo an—immaterial—common factor.

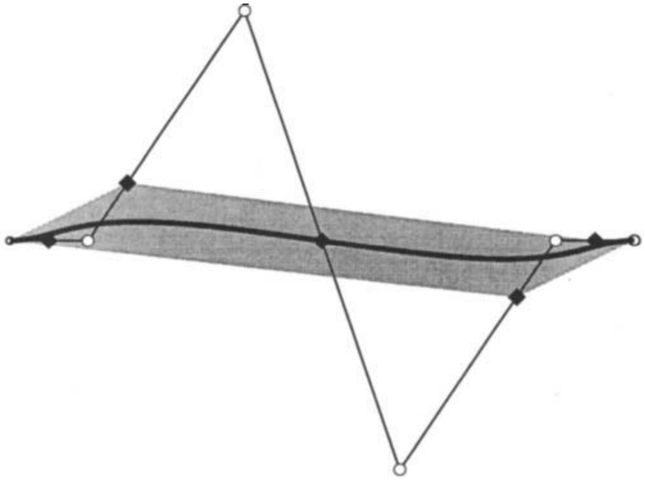


Figure 13.3 Convex hulls: if the weight points are used, tighter bounds on the curve are possible.

parameters: moving a $\mathbf{q}_i$ along the polygon leg $\mathbf{b}_i, \mathbf{b}_{i+1}$ influences the shape of the curve. It may be preferable to let a designer use these geometric handles rather than requiring him or her to input numbers for the weights.[5]

As in the nonrational case, the de Casteljau algorithm may be used to *subdivide* a curve. The de Casteljau algorithm subdivides the 4D preimage of our 3D rational Bézier curve $\mathbf{x}(t)$; see Section 5.4. The intermediate 4D points $[w_i^r \mathbf{b}_i^r \; w_i^r]^{\mathrm{T}}$; $\mathbf{b}_i^r \in \mathbb{E}^3$, may be projected into the hyperplane $w = 1$ to provide us with the control polygons for the "left" and "right" curve segment. The control vertices and weights corresponding to the interval $[0, t]$ are given by

$$\mathbf{b}_i^{\text{left}} = \mathbf{b}_0^i(t), \qquad w_i^{\text{left}} = w_0^i, \tag{13.5}$$

where the $\mathbf{b}_0^i(t)$ and the w_0^i are computed from (13.3). The control points and weights corresponding to the interval $[t, 1]$ are given by

$$\mathbf{b}_i^{\text{right}} = \mathbf{b}_i^{n-i}(t), \qquad w_i^{\text{right}} = w_i^{n-i}. \tag{13.6}$$

The weight points may be used to sharpen the convex hull property of rational Bézier curves. We know that every curve is inside the convex hull of its control polygon. But it is also contained within the convex hull of $\mathbf{b}_0, \mathbf{q}_0, \ldots, \mathbf{q}_{n-1}, \mathbf{b}_n$; see [201]. Figure 13.3 illustrates.

[5] This situation is similar to the way curves are generated using the direct G^2 spline algorithm from Chapter 11 compared to the generation of γ splines.

13.3 Derivatives

For the first derivative of a rational Bézier curve, we obtain

$$\dot{\mathbf{x}}(t) = \frac{1}{w(t)}[\dot{\mathbf{p}}(t) - \dot{w}(t)\mathbf{x}(t)], \tag{13.7}$$

where we have set

$$\mathbf{p}(t) = w(t)\mathbf{x}(t); \quad \mathbf{p}(t), \mathbf{x}(t) \in \mathbb{E}^3 \tag{13.8}$$

in complete analogy to the development in Section 12.4. For higher derivatives, we differentiate (13.8) r times:

$$\mathbf{p}^{(r)}(t) = \sum_{j=0}^{r} \binom{r}{j} w^{(j)}(t)\mathbf{x}^{(r-j)}(t).$$

We can solve for $\mathbf{x}^{(r)}(t)$:

$$\mathbf{x}^{(r)}(t) = \frac{1}{w(t)}\left[\mathbf{p}^{(r)} - \sum_{j=1}^{r} \binom{r}{j} w^{(j)}(t)\mathbf{x}^{(r-j)}(t)\right]. \tag{13.9}$$

This is a recursive formula for the rth derivative of a rational Bézier curve. It only involves taking derivatives of polynomial curves.

The first derivative may also be obtained as a byproduct of the de Casteljau algorithm, as described by Floater [235]:

$$\dot{\mathbf{x}}(t) = n\frac{w_0^{n-1}w_1^{n-1}}{[w_0^n]^2}[\mathbf{b}_1^{n-1} - \mathbf{b}_0^{n-1}]. \tag{13.10}$$

At the endpoint $t = 0$, we find

$$\dot{\mathbf{x}}(0) = \frac{nw_1}{w_0}\Delta\mathbf{b}_0.$$

Let us now consider two rational Bézier curves, one defined over the interval $[u_0, u_1]$ with control polygon $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and weights $w_0, \ldots, w_n$ and the other defined over the interval $[u_1, u_2]$ with control polygon $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$ and weights $w_n, \ldots, w_{2n}$. Both segments form a C^1 curve if

$$\frac{w_{n-1}}{\Delta_0}\Delta\mathbf{b}_{n-1} = \frac{w_{n+1}}{\Delta_1}\Delta\mathbf{b}_n, \tag{13.11}$$

where the appearance of the interval lengths Δ_i is due to the application of the chain rule. This is necessary since we now consider a composite curve with a global parameter u. Note that the weight w_n has no influence on differentiability at all!

Of course, two rational Bézier curves form a C^r curve if all of their components are C^r in homogeneous form:

$$\frac{\Delta^r[w_{n-r}\mathbf{b}_{n-r}]}{(\Delta_0)^r} = \frac{\Delta^r[w_n\mathbf{b}_n]}{(\Delta_1)^r}.$$

But keep in mind that there are composite C^r curves that do not satisfy this condition!

Although the computation of higher-order derivatives is quite involved in the case of rational Bézier curves, we note that the computation of curvature or torsion may be simplified by the application of (10.9) or (10.10) and (10.11).

13.4 Osculatory Interpolation

With rational cubics, it is easy to solve an interesting kind of interpolation problem: given a Bézier polygon $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ and a curvature value at each endpoint, find a set of weights w_0, w_1, w_2, w_3 such that the corresponding rational cubic assumes the given curvatures at $\mathbf{b}_0$ and $\mathbf{b}_3$. The following method is very similar to one developed by T. Goodman in 1988; see [271]. We assume without loss of generality that $w_0 = w_3 = 1$.[6] The given curvatures κ_0 and κ_3 are then related to the unknown weights by (10.10):

$$\kappa_0 = \frac{4}{3}\frac{w_2}{w_1^2}c_0, \quad \kappa_3 = \frac{4}{3}\frac{w_1}{w_2^2}c_1, \tag{13.12}$$

where

$$c_0 = \frac{\text{area}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}{\text{dist}^3[\mathbf{b}_0, \mathbf{b}_1]}, \qquad c_1 = \frac{\text{area}[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]}{\text{dist}^3[\mathbf{b}_2, \mathbf{b}_3]}.$$

Equations (13.12) decouple nicely so that we can determine our unknowns w_1 and w_2:

$$w_1 = \frac{4}{3}\left[\frac{c_0^2}{\kappa_0^2}\frac{c_1}{\kappa_3}\right]^{\frac{1}{3}}, \quad w_2 = \frac{4}{3}\left[\frac{c_0}{\kappa_0}\frac{c_1^2}{\kappa_3^2}\right]^{\frac{1}{3}}. \tag{13.13}$$

[6] Goodman [271] assumes that $w_1 = w_2 = 1$.

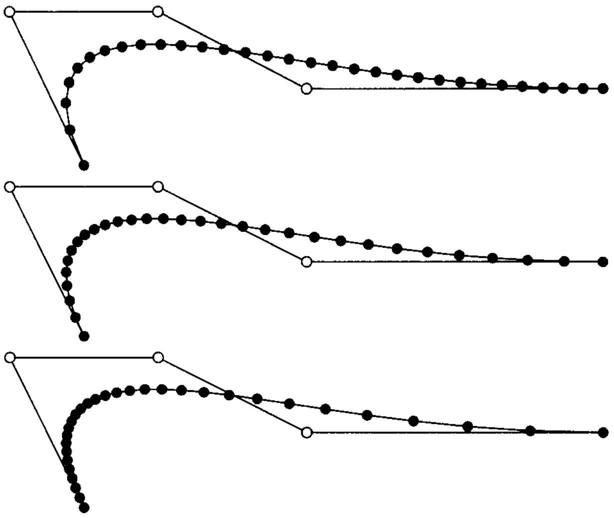


Figure 13.4 Reparametrizations: three rational Bézier curves with identical control polygons, evaluated at 31 equally spaced parameter values. Reparametrization constants are top, $c = 2$; middle, $c = 1$; bottom, $c = 1/2$.

For planar control polygons, the quantities c_0 or c_1 may be negative—this happens when a control polygon is S-shaped. This is meaningful since curvature may be defined as *signed curvature* for 2D curves, as defined in (23.1). Of course, we should then also prescribe the corresponding κ_0 and κ_3 as being negative so that we end up with positive weights.

A similar interpolation problem was addressed by Klass [361] and de Boor, Höllig, and Sabin for the nonrational case: they prescribe two points and corresponding tangent directions and curvatures [144]. The solution (when it exists) can only be obtained using an iterative method.

13.5 Reparametrization and Degree Elevation

Arguing exactly as in the conic case (see the end of Section 12.2), we may *reparametrize* a rational Bézier curve by changing the weights according to

$$\hat{w}_i = c^i w_i; \quad i = 0, \ldots, n,$$

where c is any nonzero constant. Figure 13.4 shows how the reparametrization affects the parameter spacing on the curve; note that the curve shape remains the same.

The new weights correspond to new weight points $\hat{\mathbf{q}}_i$. One can show (see Farin and Worsey [216]) that the new and old weight points are strongly related:

the cross ratios of any four points $[\mathbf{b}_i, \mathbf{q}_i, \hat{\mathbf{q}}_i, \mathbf{b}_{i+1}]$ are the same for all polygon legs.

We may always transform a rational Bézier curve to *standard form* by using the rational linear parameter transformation resulting from the choice

$$c = \sqrt[n]{\frac{w_0}{w_n}}.$$

This results in $\hat{w}_n = w_0$; after dividing all weights through by w_0, we have the standard form $\hat{w}_0 = \hat{w}_n = 1$. Of course, we have to require that the root exists. A different derivation of this result is in Patterson [457].

How can rational Bézier curves in nonstandard form arise? A common case occurs in connection with rational Bézier surfaces, as discussed in Section 16.6: the end weights of an isoparametric curve will in general not be unity. Such curves are often "extracted" from a surface and then treated as entities in their own right.

We may perform *degree elevation* (in analogy to Section 6.1) by degree elevating the 4D polygon with control vertices $[\, w_i\mathbf{b}_i \quad w_i\,]^{\mathrm{T}}$ and projecting the resulting control vertices into the hyperplane $w = 1$. Let us denote the control vertices of the degree elevated curve by $\mathbf{b}_i^{(1)}$; they are given by

$$\mathbf{b}_i^{(1)} = \frac{w_{i-1}\alpha_i\mathbf{b}_{i-1} + w_i(1-\alpha_i)\mathbf{b}_i}{w_{i-1}\alpha_i + w_i(1-\alpha_i)}; \quad i = 0, \ldots, n+1 \tag{13.14}$$

and $\alpha_i = i/(n+1)$. The weights $w_i^{(1)}$ of the new control vertices are given by

$$w_i^{(1)} = w_{i-1}\alpha_i + w_i(1-\alpha_i); \quad i = 0, \ldots, n+1.$$

Degree elevation is illustrated in Example 13.1.

The connection of reparametrization and degree elevation may lead to surprising situations. Consider the following procedure: take any rational Bézier curve in standard form and degree elevate it. Next, take the original curve, reparametrize it, then degree elevate it and bring it to standard form. We end up with two different polygons (and two different sets of standardized weights) that both describe the same rational curve. This situation is very different from the nonrational case! It is illustrated in Figure 13.5.

For the sake of completeness, we should mention that ways other than just by rational linear reparametrizations exist to reparametrize rational curves. For

Example 13.1 **Writing a semicircle as a rational cubic.**

Let a rational quadratic semicircle in control vector form be given by two control points $\mathbf{b}_0, \mathbf{b}_2$ with weights of unity and one control vector $\mathbf{v}_1$, without a weight. Its equation is given by (12.24). After degree elevation, we obtain a rational cubic with four control points:

$$[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3] = \left[\begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right]$$

and weights

$$[w_0, w_1, w_2, w_3] = \left[1, \frac{1}{3}, \frac{1}{3}, 1\right].$$

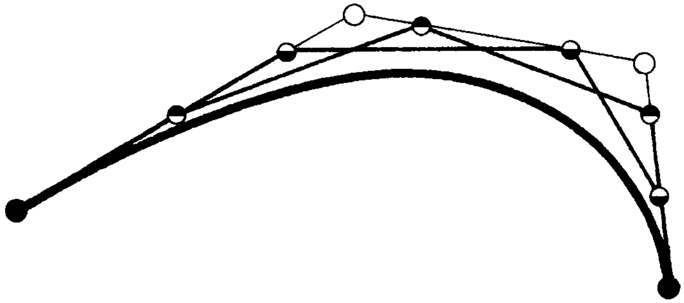


Figure 13.5 Ambiguous curve representations: the two heavy polygons represent the same rational quartic. Also indicated is the rational cubic representation that they were both obtained from.

example, the reparametrization $t \leftarrow t(2-t)$ does not change the curve, but it raises the degree from n to $2n$. As long as the reparametrization is of the form $t \leftarrow r(t)$, where $r(t)$ is a rational polynomial, we do not leave the class of rational curves. But if $r(t)$ is not a monotonic function, then the reparametrized curve will be multiply traced, or after T. Sederberg [550], “improperly parametrized.” An example of an improperly parametrized curve is given in Example 13.2, since every point of the curve corresponds to two parameter values.

Example 13.2 **Writing a full circle as a rational quintic.**

It is possible to write a *full* circle as one rational Bézier curve of degree five (see Chou [114]). Its Bézier points are given by

$$\left[\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\4\end{bmatrix}\begin{bmatrix}-3\\2\end{bmatrix}\begin{bmatrix}-3\\-2\end{bmatrix}\begin{bmatrix}1\\-4\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\right],$$

and its weights are

$$\left[1, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 1\right].$$

As the parameter t traces out all values between $-\infty$ and $+\infty$, the circle is traced out twice.

13.6 Control Vectors

In Section 12.8, we encountered control vectors (also known as infinite control points) as the limiting case of parallel tangents to a conic. The resulting curve representation contained both points and vectors. We can devise a similar form for rational Bézier curves, first suggested by K. Versprille [601]; see also [477]. They will be of the form

$$\mathbf{b}(t) = \frac{\sum_{\text{points}} w_i \mathbf{b}_i B_i^n(t) + \sum_{\text{vectors}} \mathbf{v}_i B_i^n(t)}{\sum_{\text{points}} w_i B_i^n(t)}. \tag{13.15}$$

The control vectors do not have weights in this form; we may multiply each $\mathbf{v}_i$ by a factor, however, and the curve will change accordingly. Note that at least one of the point weights w_i must be nonzero for (13.15) to be meaningful.

As in the conic case, we have lost the convex hull property, and evaluation of (13.15) will require special case treatment. However, we can eliminate the control vectors completely—we just have to degree elevate the curve (possibly more than once). Example 13.1 shows how to do this.

13.7 Rational Cubic B-Spline Curves

A 3D rational B-spline curve is the projection through the origin of a 4D nonrational B-spline curve into the hyperplane $w = 1$. The control polygon of the

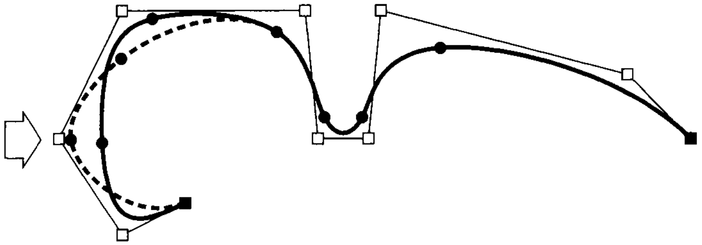


Figure 13.6 Rational B-splines: the weight of the indicated control point is changed. Dashed curve, weight = 3.0; black curve, weight = 0.33.

rational B-spline curve is given by vertices $\mathbf{d}_0, \ldots, \mathbf{d}_L$; each vertex $\mathbf{d}_i \in \mathbb{E}^3$ has a corresponding weight w_i. A point $\mathbf{x}(u)$ on a rational B-spline curve is thus given by

$$\mathbf{x}(u) = \frac{\sum_{i=0}^{L} w_i \mathbf{d}_i N_i^n(u)}{\sum_{i=0}^{L} w_i N_i^n(u)}. \tag{13.16}$$

It may be evaluated by applying the de Boor algorithm (see Section 8.2), to the homogeneous coordinates, in the same way as we evaluated rational Bézier curves. We have affine and projective invariance, and also the local control property. For nonnegative weights, we have the variation diminishing property as well as the convex hull property.

Designing with rational B-spline curves is not very different from designing with their nonrational counterparts. We now have the added freedom of being able to change weights. A change of only one weight affects a rational B-spline curve only locally, as shown in Figure 13.6.

Let us close this section with a somewhat negative result: *there is no symmetric periodic representation of a circle as a C^2 rational cubic B-spline curve.* If such a representation existed, it would be of the form

$$\mathbf{x}(u) = \sum w_i \mathbf{d}_i N_i^3(u) / \sum w_i N_i^3(u),$$

where all w_i are equal by symmetry. Then the w_i cancel, leaving us with an integral B-spline curve, which is not capable of representing a circle. Note, however, that we can represent any *open* circular arc by C^2 rational cubics.

13.8 Interpolation with Rational Cubics

The interpolation problem in the context of rational B-splines is the following:

> ***Given:*** 3D data points $\mathbf{x}_0, \ldots, \mathbf{x}_K$, parameter values $u_0, \ldots, u_K$, and weights $w_0, \ldots, w_K$.
>
> ***Find:*** A C^2 rational cubic B-spline curve with control vertices $\mathbf{d}_0, \ldots, \mathbf{d}_{K+2}$ and weights $v_0, \ldots, v_{K+2}$ that interpolates to the given data and weights.

For the solution of this problem, we follow the philosophy outlined at the end of the last section: solve a 4D interpolation problem to the data points $[\, w_i\mathbf{x}_i \quad w_i \,]^{\mathrm{T}}$ and parameter values u_i. All we have to do is to solve the linear system (9.10), where input and output is now 4D instead of the usual 3D. We will obtain a 4D control polygon $\left[\, \mathbf{e}_i \quad v_i \,\right]^{\mathrm{T}}$, from which we now obtain the desired $\mathbf{d}_i$ as $\mathbf{d}_i = \mathbf{e}_i/v_i$. The v_i are the weights of the control vertices $\mathbf{d}_i$.

We have not yet addressed the problem of how to choose the weights w_i for the data points $\mathbf{x}_i$. No known algorithms exist for this problem. It seems reasonable to assign high weights in regions where the interpolant is expected to curve sharply. Yet there is a limit to the assignment of weights: if all of them are very high, this will not have a significant effect on the curve since a common factor in all weights will simply cancel. Moreover, care must be taken to prevent the denominator of the interpolant from being zero. This is not a trivial task—for instance, we might assign a very large weight to one data point while keeping all the others at unity. The resulting weight function $w(t)$ will not be positive everywhere, giving rise to singularities at its zeros.

Integral cubic spline interpolation has *cubic precision*: if the data points and the parameter values come from one global cubic, the interpolant reproduces that cubic. In the context of rational spline interpolation, an analogous question is that of *conic precision*: if the data points and the parameter values come from one global conic, can we reproduce it? We must also require that the data points have weights assigned to them. With them, we may view the rational spline interpolation problem as an integral spline interpolation problem in $\mathbb{E}^4$. There, cubic splines have quadratic precision; that is, we may recapture any parabola. The projection of the parabola yields a conic section; thus if our data—points, parameter values, and weights—were taken from a conic, rational cubic spline interpolation will reproduce the conic.

We should note, however, that this argument is limited to open curves; for closed curves, we have already seen that we cannot represent a circle as a C^2 symmetric periodic B-spline curve.

More approaches to rational spline interpolation have recently appeared; we list Schneider [541] and Ma and Kruth [408].

13.9 Rational B-Splines of Arbitrary Degree

The process of generalizing the concept of general B-spline curves to the rational case is now straightforward. A 3D rational B-spline curve is the projection through the origin of a 4D nonrational B-spline curve into the hyperplane $w = 1$. It is thus given by

$$\mathrm{s}(u) = \frac{\sum_{j=0}^{L} w_i \mathbf{d}_i N_i^n(u)}{\sum_{j=0}^{L} w_i N_i^n(u)}. \tag{13.17}$$

We have chosen the notation from Chapter 8. Thus (13.17) is the generalization of (8.15) to the rational parametric case.

A rational B-spline curve is given by its knot sequence, its 3D control polygon, and its weight sequence. The control vertices $\mathbf{d}_i$ are the projections of the 4D control vertices $[\, w_i\mathbf{d}_i \quad w_i \,]^{\mathrm{T}}$.

To evaluate a rational B-spline curve at a parameter value u, we may apply the de Boor algorithm to both numerator and denominator of (13.17) and finally divide through. This corresponds to the evaluation of the 4D nonrational curve with control vertices $[\, w_i\mathbf{d}_i \quad w_i \,]^{\mathrm{T}}$ and to projecting the result into $\mathbb{E}^3$. Just as in the case of Bézier curves, this may lead to instabilities, and so we give a rational version of the de Boor algorithm that is more stable but also computationally more involved.

de Boor algorithm, rational: Let $u \in [u_I, u_{I+1}) \subset [u_{n-1}, u_L]$. Define

$$\mathbf{d}_i^k(u) = \left[(1-\alpha_i^k) w_{i-1}^{k-1} \mathbf{d}_{i-1}^{k-1}(u) + \alpha_i^k w_i^{k-1} \mathbf{d}_i^{k-1}(u)\right] / w_i^k \tag{13.18}$$

for $k = 1, \ldots, n-r$, and $i = I - n + k + 1, \ldots, I + 1$, where

$$\alpha_i^k = \frac{u - u_{i-1}}{u_{i+n-k} - u_{i-1}}$$

and

$$w_i^k = (1-\alpha_i^k) w_{i-1}^{k-1} + \alpha_i^k w_i^{k-1}.$$

Then

$$\mathrm{s}(u) = \mathbf{d}_{I+1}^{n-r}(u) \tag{13.19}$$

is the point on the B-spline curve at parameter value u. Here, r denotes the multiplicity of u in case it was already one of the knots. If it was not, set $r = 0$. As usual, we set $\mathbf{d}_i^0 = \mathbf{d}_i$ and $w_i^0 = w_i$.

Refer to Section 8.3 for the notation.

Knot insertion is, as in the nonrational case, performed by executing just one step of the de Boor algorithm, that is, by fixing $k = 1$ in the preceding algorithm. The original polygon vertices $\mathbf{d}_{I-n+2}, \ldots, \mathbf{d}_I$ are replaced by the $\mathbf{d}^{(1)}_{I-n+2}, \ldots, \mathbf{d}^{(1)}_{I+1}$; their weights are the numbers $w^{(1)}_{I-n+2}, \ldots, w^{(1)}_{I+1}$.

A rational B-spline curve, being a piecewise rational polynomial, has a piecewise rational Bézier representation. We can find the Bézier points and their weights for each segment by inserting every knot until it has multiplicity n, that is, by applying the de Boor algorithm to each knot. The routine bsptobez_blossom uses blossoms to perform this task.

It is also possible to *reparametrize* a rational B-spline curve, just as we could do for Bézier curves. For a description, see Lee and Lucian [381].

The *derivative* of a rational B-spline curve is conveniently found using a result by Floater:

$$\dot{\mathbf{s}}(u) = \frac{n}{u_{I+1} - u_I} \frac{w_I^{n-1} w_{I+1}^{n-1}}{[w_{I+1}^n]^2} [\mathbf{d}_{I+1}^{n-1} - \mathbf{d}_I^{n-1}], \tag{13.20}$$

which is quite analogous to (13.10).

13.10 Implementation

The following computes a point on a rational Bézier curve:

```
float ratbez(degree,coeff,weight,t)
/*
    uses rational de casteljau to compute
    point on ratbez curve for param. value t.
Input: degree:   degree of curve
       coeff:    control point coordinates
       weight:   weights
       t:        evaluation parameter
*/
```

Reparametrizing a rational Bézier curve:

```
void reparam(wold,degree,s,wnew)
/*     reparametrizes ratbez curve: only the weights, stored in wold,
       are changed. New weights are in wnew. Parametrization is
       determined by shoulder point s. For s=0.5, nothing changes.
       Also, s should be in (0,1).
*/
```

The routine to subdivide a rational Bézier curve at a parameter value t was already given in Section 5.8.

A program that generates the piecewise rational Bézier form from a rational cubic B-spline curve is:

```
void ratbspline_to_bezier(bspl_x,bspl_y,bspl_w,knot,l,bez_x,bez_y,bez_w)
/* converts rational  cubic B-spline polygon into piecewise
rational Bezier polygon
Input: bspl_x, bspl_y: planar B-spline control polygon
       bspl_w:         B-spline weights
       knot:           knot sequence
       l:              no. of intervals

Output: bez_x, bez_y:  planar piecewise Bezier polygon
        bez_w:         Bezier weights (not in piecewise standard form!)
*/
```

13.11 Problems

1 Suppose you are given two coplanar rational quadratic segments that form a C^1 curve, but not a G^2 curve. Can you adjust the weights (not the control polygons!) such that the resulting new segments form a G^2 curve? Hint: use (10.9).

***2** A rational Bézier curve may be *closed*, as in the example of a degree elevated ellipse. Show that a nonplanar 3D rational cubic cannot be closed.

***3** In Section 13.4, we said that signed curvature makes sense only in $\mathbf{e}$. Why not in $\mathbb{E}^3$?

***4** In Section 13.5, we remarked that the cross ratios of any four points $(\mathbf{b}_i, \mathbf{q}_i, \hat{\mathbf{q}}_i, \mathbf{b}_{i+1})$ are the same for all polygon legs. How is this cross ratio related to the reparametrization constant c?

P1 Define and program a rational Aitken algorithm, that is, one where the data points are assigned weights. Try to adjust those weights in an attempt to reduce the oscillatory behavior of the interpolant.

P2 Use `deboor_blossom` to write a degree elevation program for rational B-splines. Apply it repeatedly and study the behavior of the weights.

Tensor Product Patches

14

The first person to consider this class of surfaces for design purposes was probably de Casteljau, who investigated them between 1959 and 1963. The popularity of this type of surface is, however, due to the work of Bézier only slightly later, as documented in Chapter 1. Initially, Bézier patches were only used to approximate a given surface. It took some time for people to realize that any B-spline surface can also be written in piecewise Bézier form.

We will use the example of Bézier patches to demonstrate the tensor product approach to surface patches. Once that principle is developed, it will be trivial to generalize other curve schemes to tensor product surfaces.

14.1 Bilinear Interpolation

In Section 3.1, we studied linear interpolation in $\mathbb{E}^3$ and derived properties of this elementary method that we then used for the development of Bézier curves. In an analogous fashion, we can base the theory of *tensor product Bézier surfaces* on the concept of *bilinear interpolation.* Whereas linear interpolation fits the "simplest" *curve* between two points, bilinear interpolation fits the "simplest" *surface* between four points.

To be more precise: let $\mathbf{b}_{0,0}, \mathbf{b}_{0,1}, \mathbf{b}_{1,0}, \mathbf{b}_{1,1}$ be four distinct points in $\mathbb{E}^3$. The set of all points $\mathbf{x} \in \mathbb{E}^3$ of the form

$$\mathbf{x}(u, v) = \sum_{i=0}^{1} \sum_{j=0}^{1} \mathbf{b}_{i,j} B_i^1(u) B_j^1(v) \tag{14.1}$$

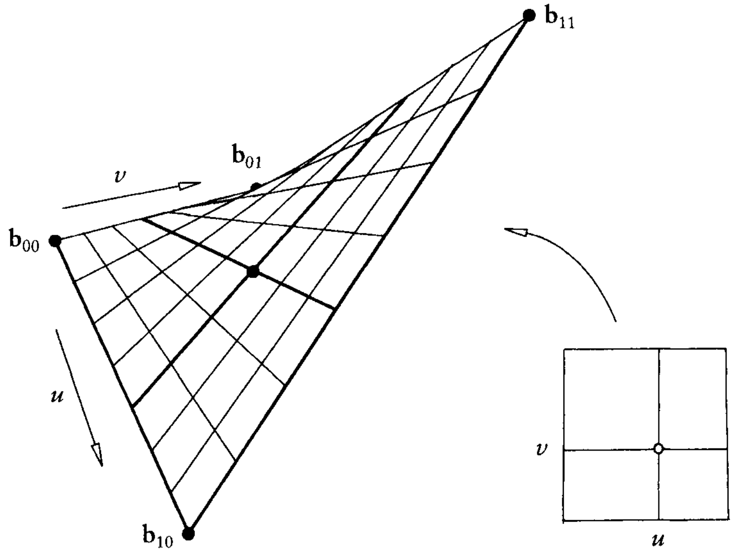


Figure 14.1 Bilinear interpolation: a hyperbolic paraboloid is defined by four points $\mathbf{b}_{i,j}$.

is called a *hyperbolic paraboloid* through the four $\mathbf{b}_{i,j}$. In matrix form:

$$\mathbf{x}(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{11} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}. \tag{14.2}$$

Since (14.1) is linear in both u and v and it interpolates to the input points, the surface $\mathbf{x}$ is called the *bilinear interpolant*. An example is shown in Figure 14.1.

The bilinear interpolant can be viewed as a map of the unit square $0 \le u, v \le 1$ in the u, v-plane. We say that the unit square is the *domain* of the interpolant, while the surface $\mathbf{x}$ is its *range*. A line parallel to one of the axes in the domain corresponds to a curve in the range; it is called an *isoparametric curve*. Every isoparametric curve of the hyperbolic paraboloid (14.1) is a straight line; thus, hyperbolic paraboloids are *ruled surfaces*; see also Sections 15.1 and 19.10. In particular, the isoparametric line $u = 0$ is mapped onto the straight line through $\mathbf{b}_{0,0}$ and $\mathbf{b}_{0,1}$; analogous statements hold for the other three boundary curves.

Instead of evaluating the bilinear interpolant directly, we can apply a two-stage process that we will employ later in the context of tensor product interpolation. We can compute two intermediate points

$$\mathbf{b}_{0,0}^{0,1} = (1-v)\mathbf{b}_{0,0} + v\mathbf{b}_{0,1}, \tag{14.3}$$

$$\mathbf{b}_{1,0}^{0,1} = (1-v)\mathbf{b}_{1,0} + v\mathbf{b}_{1,1}, \tag{14.4}$$

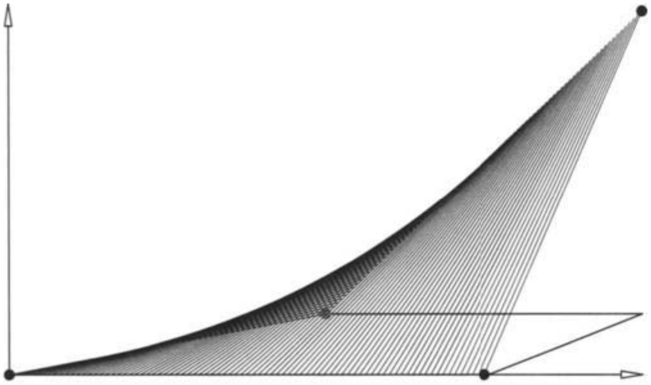


Figure 14.2 Bilinear interpolation: the surface $z = xy$ over the unit square.

and obtain the final result as

$$\mathbf{x}(u,v) = \mathbf{b}_{0,0}^{1,1}(u,v) = (1-u)\mathbf{b}_{0,0}^{0,1} + u\mathbf{b}_{1,0}^{0,1}.$$

This amounts to computing the coefficients of the isoparametric line $v = const$ first and then evaluating this isoparametric line at u. The reader should verify that the other possibility, computing a $u = const$ isoparametric line first and then evaluating it at v, gives the same result.

Since linear interpolation is an affine map, and since we apply linear interpolation (or affine maps) in both the u- and v-direction, we sometimes sees the term *biaffine map* for bilinear interpolation; see Ramshaw [498].

The term *hyperbolic paraboloid* comes from analytic geometry. We shall justify this name by considering the (nonparametric) surface $z = xy$. It can be interpreted as the bilinear interpolant to the four points

$$\begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$$

and is shown in Figure 14.2. If we intersect the surface with a plane parallel to the x, y-plane, the resulting curve is a *hyperbola*; if we intersect it with a plane containing the z-axis, the resulting curve is a *parabola*.

14.2 The Direct de Casteljau Algorithm

Bézier curves may be obtained by repeated application of linear interpolation. We shall now obtain surfaces from repeated application of *bilinear interpolation*.

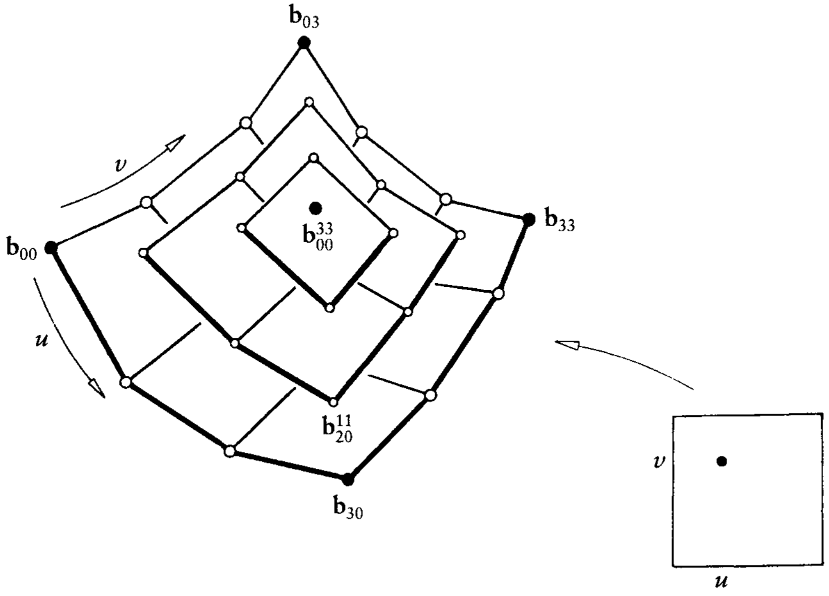


Figure 14.3 The direct de Casteljau algorithm for surfaces: the point on the surface is found from repeated bilinear interpolation.

Suppose we are given a rectangular array of points $\mathbf{b}_{i,j}$; $0 \le i, j \le n$ and parameter values (u, v). The following algorithm generates a point on a surface determined by the array of the $\mathbf{b}_{i,j}$:

Given $\{\mathbf{b}_{i,j}\}_{i,j=0}^{n}$ and $(u, v) \in \mathbb{R}^2$, set

$$\mathbf{b}_{i,j}^{r,r} = [\,1-u \quad u\,] \begin{bmatrix} \mathbf{b}_{i,j}^{r-1,r-1} & \mathbf{b}_{i,j+1}^{r-1,r-1} \\ \mathbf{b}_{i+1,j}^{r-1,r-1} & \mathbf{b}_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}, \quad \begin{matrix} r = 1, \ldots, n \\ i, j = 0, \ldots, n-r \end{matrix} \tag{14.5}$$

and $\mathbf{b}_{i,j}^{0,0} = \mathbf{b}_{i,j}$. Then $\mathbf{b}_{0,0}^{n,n}(u, v)$ is the point with parameter values (u, v) on the *Bézier surface* $\mathbf{b}^{n,n}$. (The reason for the somewhat clumsy identical superscripts will be explained in the next section.) The net of the $\mathbf{b}_{i,j}$ is called the *Bézier net* or *control net* of the surface $\mathbf{b}^{n,n}$. The $\mathbf{b}_{i,j}$ are called control points or Bézier points, just as in the curve case. Figure 14.3 shows an example for $n = 3$; Example 14.1 shows how to compute the quadratic case. An example of a bicubic ($n = 3$) Bézier patch is shown in Figure 14.4.

We have defined a surface scheme through a constructive algorithm just as we have done in the curve case. We could now continue to derive analytic properties

Example 14.1 **Computing a point on a Bézier surface using the direct de Casteljau algorithm.**

Let a Bézier control net be given by

$$\begin{matrix} \begin{bmatrix}0\\0\\0\end{bmatrix} & \begin{bmatrix}2\\0\\0\end{bmatrix} & \begin{bmatrix}4\\0\\0\end{bmatrix} \\ \begin{bmatrix}0\\2\\0\end{bmatrix} & \begin{bmatrix}2\\2\\0\end{bmatrix} & \begin{bmatrix}4\\2\\2\end{bmatrix} \\ \begin{bmatrix}0\\4\\0\end{bmatrix} & \begin{bmatrix}2\\4\\4\end{bmatrix} & \begin{bmatrix}4\\4\\4\end{bmatrix} \end{matrix}.$$

After one step of the direct de Casteljau algorithm for $(u, v) = (0.5, 0.5)$, we obtain

$$\begin{matrix} \begin{bmatrix}1\\1\\0\end{bmatrix} & \begin{bmatrix}3\\1\\0.5\end{bmatrix} \\ \begin{bmatrix}1\\3\\1\end{bmatrix} & \begin{bmatrix}3\\3\\2.5\end{bmatrix} \end{matrix}.$$

The point on the surface is

$$\begin{bmatrix}2\\2\\1\end{bmatrix}.$$

of these surfaces, again as in the curve case. This is possible without much effort; however, we use a different approach in Section 14.3.

In the next section, we shall be able to handle surfaces that are of different degrees in u and v. Such surfaces have control nets $\{\mathbf{b}_{i,j}\}$; $i = 0, \ldots, m, j = 0, \ldots, n$. The direct de Casteljau algorithm for such surfaces exists, but it needs a case distinction: consulting Figure 14.5, we see that the direct de Casteljau algorithm cannot be performed until the point of the surface is reached. Instead, after $k = \min(m, n)$, the intermediate $\mathbf{b}_{i,j}^{k,k}$ form a curve control polygon. We now must proceed with the univariate de Casteljau algorithm to obtain a point on the surface. This case distinction is awkward and will not be encountered by the tensor product approach in the next section.

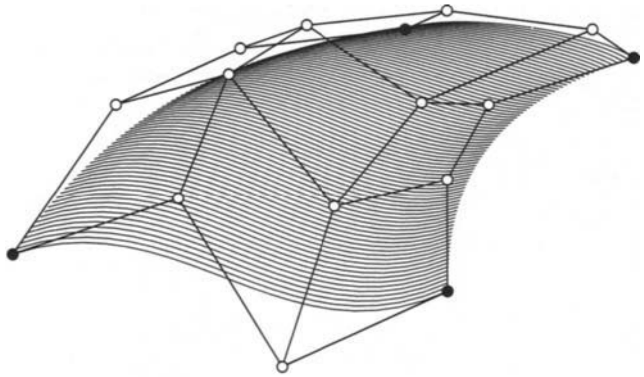


Figure 14.4 Bézier surfaces: a bicubic patch with its defining control net.

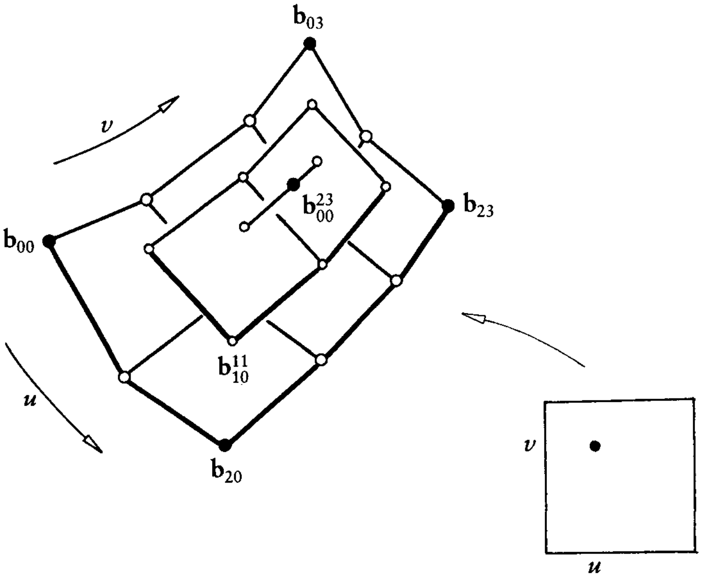


Figure 14.5 The direct de Casteljau algorithm: a surface with $(m, n) = (2, 3)$ proceeds in a univariate manner after no more direct de Casteljau steps can be performed.

14.3 The Tensor Product Approach

We have seen in the first chapter by P. Bézier how stylists in the design shop physically created surfaces: templates were used to scrape material off a rough clay model (see Figure 1.12 in Chapter 1). Different templates are used as more and more of the surface is carved out of the clay. Analyzing this process from a

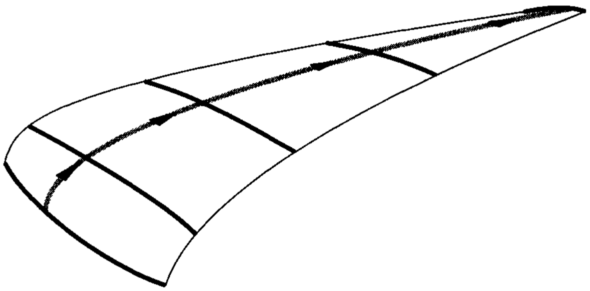


Figure 14.6 Tensor product surfaces: a surface can be thought of as being swept out by a moving and deforming curve.

theoretical viewpoint, we arrive at the following intuitive definition of a surface: *a surface is the locus of a curve that is moving through space and thereby changing its shape*. See Figure 14.6 for an illustration.

We will now formalize this intuitive concept in order to arrive at a mathematical description of a surface. First, we assume that the moving curve is a Bézier curve of constant degree m. (This assumption is made so that the following formulas will work out; it is actually a serious restriction on the class of surfaces that we can represent using the tensor product approach.) At any time, the moving curve is then determined by a set of control points. Each original control point moves through space on a curve. Our next assumption is that this curve is also a Bézier curve, and that the curves on which the control points move are all of the same degree. An example is given in Figure 14.7.

This can be formalized as follows: let the initial curve be a Bézier curve of degree m:

$$\mathbf{b}^m(u) = \sum_{i=0}^{m} \mathbf{b}_i B_i^m(u).$$

Let each $\mathbf{b}_i$ traverse a Bézier curve of degree n:

$$\mathbf{b}_i = \mathbf{b}_i(v) = \sum_{j=0}^{n} \mathbf{b}_{i,j} B_j^n(v).$$

We can now combine these two equations and obtain the point $\mathbf{b}^{m,n}(u, v)$ on the surface $\mathbf{b}^{m,n}$ as

$$\mathbf{b}^{m,n}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{b}_{i,j} B_i^m(u) B_j^n(v). \tag{14.6}$$

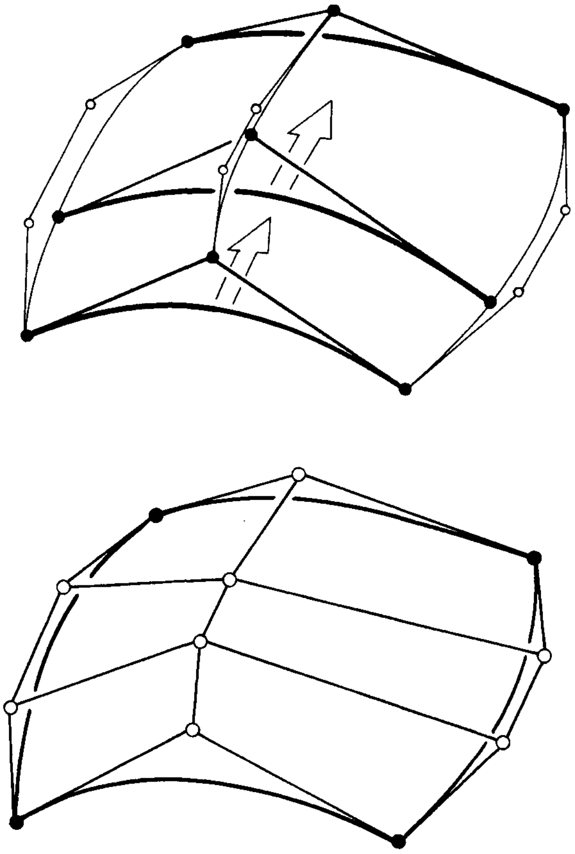


Figure 14.7 Tensor product Bézier surfaces: top, a surface is obtained by moving the control points of a curve (quadratic) along other Bézier curves (cubic); bottom; the final Bézier net.

With this notation, the original curve $\mathbf{b}^m(u)$ now has Bézier points $\mathbf{b}_{i,0}; i = 0, \ldots, m$.

It is not difficult to prove that the definition of a Bézier surface (14.6) and the definition using the direct de Casteljau algorithm are equivalent (see Problems). Example 14.2 supports this view.

We have described the Bézier surface (14.6) as being obtained by moving the isoparametric curve corresponding to $v = 0$. It is an easy exercise to check that the three remaining boundary curves could also have been used as the starting curve.

An arbitrary isoparametric curve $\hat{v} = const$ of a Bézier surface $\mathbf{b}^{m,n}$ is a Bézier curve of degree m in u, and its $m + 1$ Bézier points are obtained by evaluating all columns of the control net at $v = const$. As a formula:

Example 14.2 **Computing a point on a Bézier surface using the tensor product method.**

We can also compute the point on the surface of Example 14.1 by the tensor product method. We then evaluate each row of Bézier points for $u = 1/2$, and obtain the intermediate values

$$\begin{matrix}\begin{bmatrix}2\\0\\0\end{bmatrix}\\\begin{bmatrix}2\\2\\0.5\end{bmatrix}\\\begin{bmatrix}2\\4\\3\end{bmatrix}\end{matrix}.$$

This quadratic control polygon defines the isoparametric curve $\mathbf{b}(\frac{1}{2}, v)$; we evaluate it for $v = 1/2$ to obtain the same point as in Example 14.1.

$$\mathbf{b}_{i,0}^{0,n}(\hat{v}) = \sum_{j=0}^{n} \mathbf{b}_{ij} B_j^n(\hat{v}); \qquad i = 0, \ldots m.$$

This process of obtaining the Bézier points of an isoparametric line is a second possible interpretation of Figure 14.7. The coefficients of the isoparametric line can be obtained by applying $m + 1$ de Casteljau algorithms. A point on the surface is then obtained by performing one more de Casteljau algorithm.

Isoparametric curves $u = const$ are treated analogously. Note, however, that other straight lines in the domain are mapped to higher-degree curves on the patch: they are generally of degree $n + m$. Two special examples of such curves are the two diagonals of the domain rectangle. See also Section 14.8.

14.4 Properties

Most properties of Bézier patches follow in a straightforward way from those of Bézier curves—refer to Sections 4.3 and 5.2. We give a brief listing:

Affine invariance: The direct de Casteljau algorithm consists of repeated bilinear and possibly subsequent repeated linear interpolation. All these operations are affinely invariant; hence, so is their composition. We can also argue that in order for (14.6) to be a barycentric combination (and therefore affinely invariant), we must have

$$\sum_{j=0}^{n}\sum_{i=0}^{m} B_i^m(u)B_j^n(v) \equiv 1. \tag{14.7}$$

This identity is easily verified algebraically. A warning: there is no projective invariance of Bézier surfaces! In particular, we cannot apply a perspective projection to the control net and then plot the surface that is determined by the resulting image. Such operations will be possible by means of rational Bézier surfaces.

Convex hull property: For $0 \le u, v \le 1$, the terms $B_i^m(u)B_j^n(v)$ are nonnegative. Then, taking (14.7) into account, (14.6) is a convex combination.

Boundary curves: The boundary curves of the patch $\mathbf{b}^{m,n}$ are polynomial curves. Their Bézier polygons are given by the boundary polygons of the control net. In particular, the four corners of the control net all lie on the patch.

Variation diminishing property: This property is *not* inherited from the univariate case. In fact, it is not at all clear what the definition of variation diminution should be in the bivariate case. Counting intersections with straight lines, as we did for curves, would not make Bézier patches variation diminishing; it is easy to visualize a patch that is intersected by a straight line while its control net is not. (Here, we would view the control net as a collection of bilinear patches.) Other attempts at a suitable definition of a bivariate variation diminishing property have been similarly unsuccessful.

14.5 Degree Elevation

Suppose we want to rewrite a Bézier surface of degree (m, n) as one of degree $(m+1, n)$. This amounts to finding coefficients $\mathbf{b}_{i,j}^{(1,0)}$ such that

$$\mathbf{b}^{m,n}(u, v) = \sum_{j=0}^{n}\left[\sum_{i=0}^{m+1} \mathbf{b}_{i,j}^{(1,0)} B_i^{m+1}(u)\right] B_j^n(v).$$

The $n+1$ terms in square brackets represent $n+1$ univariate degree elevation problems as discussed in Section 6.1. They are solved by a direct application of (6.1):

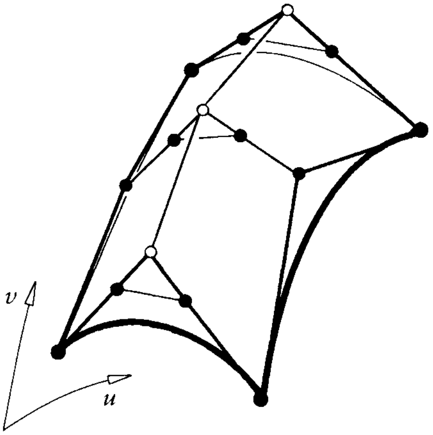


Figure 14.8 Degree elevation: the surface problem can be reduced to a series of univariate problems.

$$\mathbf{b}_{i,j}^{(1,0)} = \frac{i}{m+1}\mathbf{b}_{i-1,j} + \left(1 - \frac{i}{m+1}\right)\mathbf{b}_{i,j}; \begin{cases} i = 0, \ldots, m+1 \\ j = 0, \ldots, n. \end{cases} \tag{14.8}$$

A tensor product surface is thus degree elevated in the u-direction by treating all rows of the control net as Bézier polygons of mth degree curves and degree elevating each of them. This is illustrated in Figure 14.8.

Degree elevation in the v-direction works the same way, of course. If we want to degree elevate in both the u- and the v-direction, we can perform the procedure first in the u-direction, then in the v-direction, or we can proceed the other way around. Both approaches yield the same surface of degree $(m+1, n+1)$. Its coefficients $\mathbf{b}_{i,j}^{(1,1)}$ may be found in a one-step method:

$$\mathbf{b}_{i,j}^{(1,1)} = \begin{bmatrix} \frac{i}{m+1} & 1 - \frac{i}{m+1} \end{bmatrix} \begin{bmatrix} \mathbf{b}_{i-1,j-1} & \mathbf{b}_{i-1,j} \\ \mathbf{b}_{i,j-1} & \mathbf{b}_{i,j} \end{bmatrix} \begin{bmatrix} \frac{j}{n+1} \\ 1 - \frac{j}{n+1} \end{bmatrix} \qquad \begin{aligned} i &= 0, \ldots, m+1, \\ j &= 0, \ldots, n+1. \end{aligned} \tag{14.9}$$

The net of the $\mathbf{b}_{i,j}^{(1,1)}$ is obtained by *piecewise bilinear interpolation* from the original control net.

14.6 Derivatives

In the curve case, taking derivatives was accomplished by differencing the control points. The same will be true here. The derivatives that we will consider are

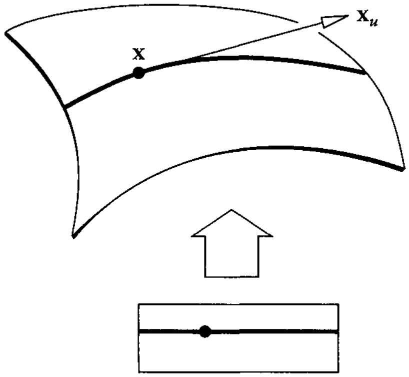


Figure 14.9 Derivatives: a partial derivative is the tangent vector of an isoparametric curve.

partial derivatives $\partial/\partial u$ or $\partial/\partial v$. A partial derivative is the tangent vector of an isoparametric curve; see Figure 14.9. It can be found by a straightforward calculation:

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)=\sum_{j=0}^{n}\left[\frac{\partial}{\partial u}\sum_{i=0}^{m}\mathbf{b}_{i,j}B_i^m(u)\right]B_j^n(v).$$

The bracketed terms depend only on u, and we can apply the formula for the derivative of a Bézier curve (5.19):

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)=m\sum_{j=0}^{n}\sum_{i=0}^{m-1}\Delta^{1,0}\mathbf{b}_{i,j}B_i^{m-1}(u)B_j^n(v).$$

Here we have generalized the standard difference operator in the obvious way: the superscript $(1,0)$ means that differencing is performed only on the first subscript: $\Delta^{1,0}\mathbf{b}_{i,j}=\mathbf{b}_{i+1,j}-\mathbf{b}_{i,j}$. If we take v-partials, we employ a difference operator that acts only on the second subscripts: $\Delta^{0,1}\mathbf{b}_{i,j}=\mathbf{b}_{i,j+1}-\mathbf{b}_{i,j}$. We then obtain

$$\frac{\partial}{\partial v}\mathbf{b}^{m,n}(u,v)=n\sum_{i=0}^{m}\sum_{j=0}^{n-1}\Delta^{0,1}\mathbf{b}_{i,j}B_j^{n-1}(v)B_i^m(u).$$

Again, a surface problem can be broken down into several univariate problems: to compute a u-partial, for instance, interpret all columns of the control net as Bézier curves of degree m and compute their derivatives (evaluated at the desired value of u). Then interpret these derivatives as coefficients of another Bézier curve of degree n and compute its value at the desired value of v.

We can write down formulas for higher-order partials:

$$\frac{\partial^r}{\partial u^r}\mathbf{b}^{m,n}(u,v) = \frac{m!}{(m-r)!}\sum_{j=0}^{n}\sum_{i=0}^{m-r}\Delta^{r,0}\mathbf{b}_{i,j}B_i^{m-r}(u)B_j^n(v) \tag{14.10}$$

and

$$\frac{\partial^s}{\partial v^s}\mathbf{b}^{m,n}(u,v) = \frac{n!}{(n-s)!}\sum_{i=0}^{m}\sum_{j=0}^{n-s}\Delta^{0,s}\mathbf{b}_{i,j}B_j^{n-s}(v)B_i^m(u). \tag{14.11}$$

Here, the difference operators are defined by

$$\Delta^{r,0}\mathbf{b}_{i,j} = \Delta^{r-1,0}\mathbf{b}_{i+1,j} - \Delta^{r-1,0}\mathbf{b}_{i,j}$$

and

$$\Delta^{0,s}\mathbf{b}_{i,j} = \Delta^{0,s-1}\mathbf{b}_{i,j+1} - \Delta^{0,s-1}\mathbf{b}_{i,j}.$$

It is not hard now to write down the most general case, namely, *mixed partials* of arbitrary order:

$$\frac{\partial^{r+s}}{\partial u^r \partial v^s}\mathbf{b}^{m,n}(u,v)$$
$$= \frac{m!n!}{(m-r)!(n-s)!}\sum_{i=0}^{m-r}\sum_{j=0}^{n-s}\Delta^{r,s}\mathbf{b}_{i,j}B_i^{m-r}(u)B_j^{n-s}(v). \tag{14.12}$$

Before we proceed to consider some special cases, recall that the coefficients $\Delta^{r,s}\mathbf{b}_{i,j}$ are vectors and therefore do not "live" in $\mathbb{E}^3$. See Section 5.3 for more details.

For $r = s = 1$, we obtain a mixed partial which is known as the "twist." It is discussed in more detail in Section 14.10.

A partial derivative of a point-valued surface is itself a vector-valued surface. We can evaluate it along isoparametric lines, of which the four boundary curves are the ones of most interest. Such a derivative, for example, $\partial/\partial u\,|_{u=0}$, is called a *cross boundary derivative*. We can thus restrict (14.10) to $u = 0$ and get, with a slight abuse of notation,

$$\frac{\partial^r}{\partial u^r}\mathbf{b}^{m,n}(0,v) = \frac{m!}{(m-r)!}\sum_{j=0}^{n}\Delta^{r,0}\mathbf{b}_{0,j}B_j^n(v). \tag{14.13}$$

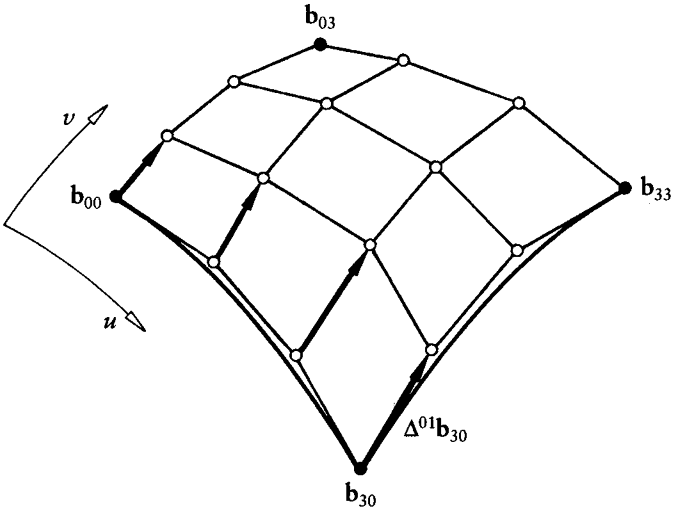


Figure 14.10 Cross boundary derivatives: along the edge $v = 0$, the cross boundary derivative depends on only two rows of control points.

Similar formulas hold for the other three edges. We thus have determined that rth order cross boundary derivatives, evaluated along that boundary, depend only on the $r + 1$ rows (or columns) of Bézier points next to that boundary. This will be important when we formulate conditions for C^r continuity between adjacent patches. The case $r = 1$ is illustrated in Figure 14.10.

14.7 Blossoms

Blossoms helped us gain insight into many properties of polynomial curves; the tensor product analogy is just as helpful and is developed easily. We define a tensor product blossom as

$$\mathbf{b}[u_1, \ldots, u_m | v_1, \ldots, v_n],$$

meaning the following: compute the (curve) blossom values $\mathbf{b}_i[u_1, \ldots, u_m]$ of all rows of control points, using the same values for each row. Then use those values as input to the (curve) blossom $\mathbf{b}[v_1, \ldots, v_n]$.[1]

[1] Of course, we could have started with the columns first.

Tensor product blossoms inherit their properties from their curve building blocks. Thus, the blossom $\mathbf{b}[u^{<m>}|v^{<n>}]$ is the point on the surface, the order of evaluations does not matter, and we have multiaffinity in both u and v.

Two examples of blossoms: the osculating bilinear surface $\mathbf{t}(s,t)$ at a point $\mathbf{x}(u,v)$ may be written as

$$\mathbf{t}(s,t) = \mathbf{b}[u^{<m-1>}, s|v^{<n-1>}, t]. \tag{14.14}$$

This surface is linear in both s and t and agrees with $\mathbf{x}(u,v)$ in both partials and twist (modulo some constant factors). The surface $\mathbf{o}(s,t)$ given by

$$\mathbf{o}(s,t) = \mathbf{b}[u, s^{<m-1>}|v, t^{<n-1>}]$$

is the *osculant* or *first polar* of the given surface at $\mathbf{x}(u,v)$. It is the analog of the univariate polar from Section 5.6.

Just as in the curve case, we may use blossoms for subdivision or domain transformation. If the new patch is to be defined over the domain rectangle $[a,b] \times [c,d]$, then its Bézier points $\mathbf{c}_{i,j}$ are given by

$$\mathbf{c}_{i,j} = \mathbf{b}[a^{<m-i>}, b^{<i>}|c^{<n-j>}, d^{<j>}]. \tag{14.15}$$

For the special case $[a,b] = [c,d] = [0,1]$, we recover the original Bézier points. Though (14.15) may look complicated, it really is not: all we have to do is to write a tensor product blossom routine—a matter of about ten lines of code!

Blossoms may also be used to find derivatives, in complete analogy to Section 5.3. Mixed partials take the form

$$\frac{\partial^{r+s}\mathbf{x}(u,v)}{\partial u^r \partial v^s} = \frac{m!}{(m-r)!}\frac{n!}{(n-s)!}\mathbf{b}[\vec{1}^{<r>}, u^{<m-r>}|\vec{1}^{<s>}, v^{<n-s>}]. \tag{14.16}$$

Evaluations with respect to the vector $\vec{1}$ in the u-domain are equivalent to taking differences in the i-direction, those with respect to the v-vector $\vec{1}$ correspond to differences in the j-direction. Again, it does not matter in which order we perform the evaluations.

We may use the blossom formulation of derivatives to approach a practical problem. It is often the case[2] that not only a point on a surface is needed, but also its u- and v-partials. Standard tensor product evaluation will give us only either a u-partial *or* a v-partial as a byproduct. However, (14.14) may always be used to compute both partials. Algorithmically, here is what to do: for a given

[2] For applications such as rendering or numerical methods.

(u, v) (no blossom notation here), perform evaluation with respect to u for all rows of control points, but stop all evaluations at level $m-1$. This gives us two points per row. Then perform evaluation with respect to v for the resulting two columns of points, now stopping at level $n-1$. We have generated four control points, corresponding to the bilinear osculant $\mathbf{t}$ of (14.14). They may now be used for evaluation of position and partials. For example, we find the u-partial as:

$$\frac{\partial \mathbf{x}(u,v)}{\partial u} = m\big([(1-v)\mathbf{t}_{10} + v\mathbf{t}_{11}] - [(1-v)\mathbf{t}_{00} + v\mathbf{t}_{01}]\big)$$

with $\mathbf{t}_{ij}$ the control points of $\mathbf{t}(u, v)$. This approach was first discussed by Mann and DeRose [413]. See also Sederberg [551].

14.8 Curves on a Surface

Let $\mathbf{p} = (\mathbf{p}_u, \mathbf{p}_v)$ and $\mathbf{q} = (\mathbf{q}_u, \mathbf{q}_v)$ be u, v-coordinates of two points in the domain of a tensor product patch of degree (n, n).[3] Let

$$\mathbf{u}(t) = (1-t)\mathbf{p} + t\mathbf{q}$$

be the parametric form of a straight line through $\mathbf{p}$ and $\mathbf{q}$. This line is mapped to a curve on the surface. What are its Bézier points $\mathbf{c}_k$?

Let $\mathbf{b}[u_1, \ldots, u_n \mid v_1, \ldots, v_n]$ be the blossom of the surface. Then a point on the curve is given by

$$\mathbf{b}[((1-t)\mathbf{p}_u + t\mathbf{q}_u)^{<n>} \mid ((1-t)\mathbf{p}_v + t\mathbf{q}_v)^{<n>}].$$

Applying the Leibniz formula (3.23) to the u-part of the blossom, we can write it as

$$\sum_{i+j=n} \binom{n}{i,j} (1-t)^i t^j \mathbf{b}[\mathbf{p}_u^{<i>}, \mathbf{q}_u^{<j>} \mid ((1-t)\mathbf{p}_v + t\mathbf{q}_v)^{<n>}].$$

Applying this technique to the v-part also, we get

$$\sum_{i+j=n} \sum_{r+s=n} \binom{n}{i,j} \binom{n}{r,s} (1-t)^i t^j (1-t)^r t^s \mathbf{b}[\mathbf{p}_u^{<i>}, \mathbf{q}_u^{<j>} \mid \mathbf{p}_v^{<r>}, \mathbf{q}_v^{<s>}]$$

[3] This is not a restriction: we may always degree elevate to achieve equal degrees in u and in v.

Equivalently,

$$\sum_{i=0}^{n}\sum_{j=0}^{n}\binom{n}{i}\binom{n}{j}(1-t)^{2n-i-j}t^{i+j}\mathbf{b}[\mathbf{p}_u^{<i>},\mathbf{q}_u^{<n-i>} \mid \mathbf{p}_v^{<j>},\mathbf{q}_v^{<n-j>}]$$

or

$$\sum_{k=0}^{2n}\sum_{i+j=k}\frac{\binom{n}{i}\binom{n}{j}}{\binom{2n}{k}}B_k^{2n}\mathbf{b}[\mathbf{p}_u^{<i>},\mathbf{q}_u^{<n-i>} \mid \mathbf{p}_v^{<j>},\mathbf{q}_v^{<n-j>}].$$

Thus

$$\mathbf{c}_k=\frac{\binom{n}{i}\binom{n}{j}}{\binom{2n}{k}}\mathbf{b}[\mathbf{p}_u^{<i>},\mathbf{q}_u^{<n-i>} \mid \mathbf{p}_v^{<j>},\mathbf{q}_v^{<n-j>}].$$

This development is due to T. DeRose [159].

14.9 Normal Vectors

The *normal vector* **n** of a surface is a normalized vector that is normal to the surface at a given point. It can be computed from the cross product of any two vectors that are tangent to the surface at that point. Since the partials $\partial/\partial u$ and $\partial/\partial v$ are two such vectors, we may set

$$\mathbf{n}(u,v)=\frac{\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)\wedge\frac{\partial}{\partial v}\mathbf{b}^{m,n}(u,v)}{\|\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)\wedge\frac{\partial}{\partial v}\mathbf{b}^{m,n}(u,v)\|},\tag{14.17}$$

where $\wedge$ denotes the cross product.

At the four corners of the patch, the involved partials are simply differences of boundary points, for example,

$$\mathbf{n}(0,0)=\frac{\Delta^{1,0}\mathbf{b}_{0,0}\wedge\Delta^{0,1}\mathbf{b}_{0,0}}{\|\Delta^{1,0}\mathbf{b}_{0,0}\wedge\Delta^{0,1}\mathbf{b}_{0,0}\|}.\tag{14.18}$$

The normal at one of the corners (we take $\mathbf{b}_{0,0}$ as an example) is undefined if $\Delta^{1,0}\mathbf{b}_{0,0}$ and $\Delta^{0,1}\mathbf{b}_{0,0}$ are linearly dependent: if that were the case, (14.18) would degenerate into an expression of the form 0/0. The corresponding patch corner is then called *degenerate*. Two cases of special interest are illustrated in Figures 14.11, 14.12, and 14.13.

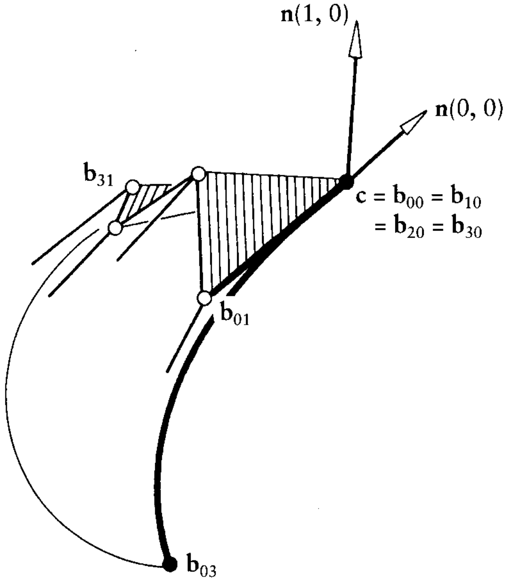


Figure 14.11 Degenerate patches: a "triangular" patch is created by collapsing a whole boundary curve into a point. The normal at that point may be undefined. Normals are shown for $u = 0$ and for $u = 1$.

In the first of these, a whole boundary curve is collapsed into a single point. As an example, we could set $\mathbf{b}_{00} = \mathbf{b}_{10} = \cdots = \mathbf{b}_{m0} = \mathbf{c}$. Then the boundary $\mathbf{b}(u, 0)$ would degenerate into a single point. In such cases, the normal vector at $v = 0$ may or may not be defined. To examine this in more detail, consider the tangents of the isoparametric lines $u = \hat{u}$, evaluated at $v = 0$. These tangents must be perpendicular to the normal vector, if it exists. So a condition for the existence of the normal vector at $\mathbf{c}$ is that all v-partials, evaluated at $v = 0$, are coplanar. But that is equivalent to $\mathbf{b}_{01}, \mathbf{b}_{11}, \ldots, \mathbf{b}_{m1}$ and $\mathbf{c}$ being coplanar.

A second possibility in creating degenerate patches is to allow two corner partials to be collinear, for example, $\partial/\partial u$ and $\partial/\partial v$ at $(0, 0)$, as shown in Figure 14.13. In that case, $\mathbf{b}_{10}, \mathbf{b}_{01}$, and $\mathbf{b}_{00}$ are collinear. Then the normal at $\mathbf{b}_{00}$ is defined, provided that $\mathbf{b}_{11}$ is not collinear with $\mathbf{b}_{10}, \mathbf{b}_{01}$, and $\mathbf{b}_{00}$. Recall that $\mathbf{b}_{00}, \mathbf{b}_{10}, \mathbf{b}_{01}$, and $\mathbf{b}_{11}$ form the osculating paraboloid at $(u, v) = (0, 0)$. Then it follows that the tangent plane at $\mathbf{b}_{00}$ is the plane through the four coplanar points $\mathbf{b}_{00}, \mathbf{b}_{10}, \mathbf{b}_{01}$, and $\mathbf{b}_{11}$. The normal at $\mathbf{b}_{00}$ is perpendicular to it.

A warning: when we say "the normal is defined," it should be understood that this is a purely mathematical statement. In any of the preceding degeneracies, a program using (14.17) will crash. A case distinction is necessary, and then

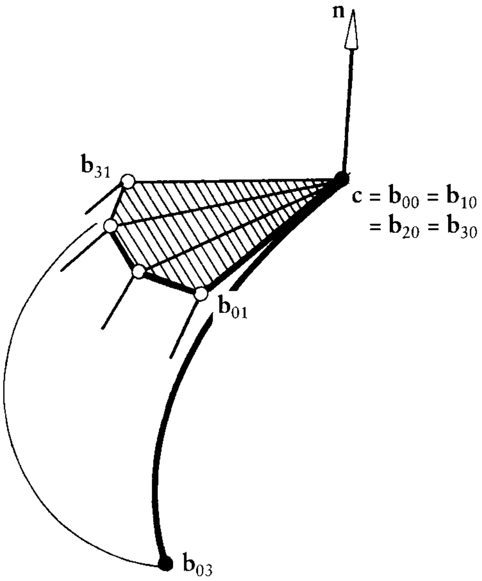

n
$\mathbf{b}_{31}$
$\mathbf{c} = \mathbf{b}_{00} = \mathbf{b}_{10}$
$= \mathbf{b}_{20} = \mathbf{b}_{30}$
$\mathbf{b}_{01}$
$\mathbf{b}_{03}$

Figure 14.12 Degenerate patches: if all $\mathbf{b}_{i1}$ and $\mathbf{c}$ are coplanar, then the normal vector at $\mathbf{c}$ is perpendicular to that plane.

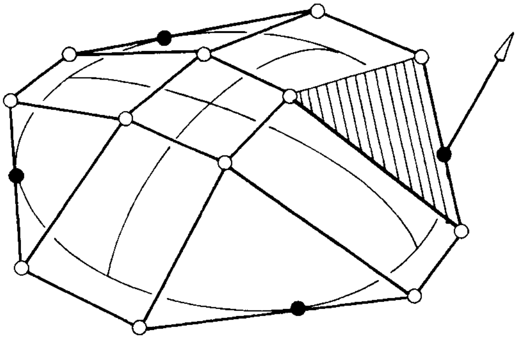


Figure 14.13 Degenerate patches: the normals at all four corners of this patch are determined by the triangles that are formed by the corner subquadrilaterals (one corner highlighted).

the program can branch into the special cases that we just described. More complex situations are encountered when we also want to compute curvatures of a degenerate patch. A solution is offered in [616]. An a priori check for degenerate normals is described in [357].

14.10 Twists

The *twist* of a surface[4] is its mixed partial $\partial^2/\partial u \partial v$. According to (14.12), the twist surface of $\mathbf{b}^{m,n}$ is a Bézier surface of degree $(m-1, n-1)$, and its (vector) coefficients have the form $mn\Delta^{1,1}\mathbf{b}_{i,j}$. These coefficients have a nice geometric interpretation. The bilinear case is shown in Figure 14.14. In general, the point $\mathbf{p}_{i,j}$ is the fourth point on the parallelogram defined by $\mathbf{b}_{i,j}, \mathbf{b}_{i+1,j}, \mathbf{b}_{i,j+1}$. It is defined by

$$\mathbf{p}_{i,j} - \mathbf{b}_{i+1,j} = \mathbf{b}_{i,j+1} - \mathbf{b}_{i,j}. \tag{14.19}$$

Since

$$\Delta^{1,1}\mathbf{b}_{i,j} = (\mathbf{b}_{i+1,j+1} - \mathbf{b}_{i+1,j}) - (\mathbf{b}_{i,j+1} - \mathbf{b}_{i,j}), \tag{14.20}$$

it follows that

$$\Delta^{1,1}\mathbf{b}_{i,j} = \mathbf{b}_{i+1,j+1} - \mathbf{p}_{i,j}. \tag{14.21}$$

Thus the terms $\Delta^{1,1}\mathbf{b}_{i,j}$ measure the deviation of each subquadrilateral of the Bézier net from a parallelogram.

The twists at the four patch corners determine the deviation of the respective corner subquadrilaterals of the control net from parallelograms. For example,

$$\frac{\partial^2}{\partial u \partial v}\mathbf{b}^{m,n}(0,0) = mn\Delta^{1,1}\mathbf{b}_{00}. \tag{14.22}$$

This twist vector is a measure for the deviation of $\mathbf{b}_{11}$ from the tangent plane at $\mathbf{b}_{00}$.

An interesting class of surfaces is obtained if all subquadrilaterals $\mathbf{b}_{i,j}$, $\mathbf{b}_{i,j+1}$, $\mathbf{b}_{i+1,j}$, $\mathbf{b}_{i+1,j+1}$ are parallelograms; in that case the twist vanishes everywhere. Such surfaces are called *translational surfaces* and will be discussed in Section 15.3; an example is shown in Figure 15.6. They have an interesting shape property: if all control points of a translational surface lie on the boundary of their convex hull, then the surface is *convex*; see Schelske [540]. A surface is convex if it does not contain a pair of points such that their connection by a straight line intersects the surface.

[4] In this chapter, we are dealing only with polynomial surfaces. For these, the twist is uniquely defined. For other surfaces, it may depend on the order in which derivatives are taken; see Section 22.6.

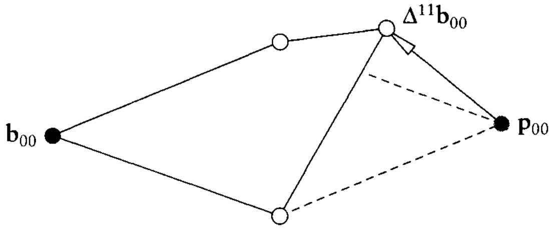


Figure 14.14 Twists: the twist is proportional to the deviation of a control net quadrilateral from a parallelogram.

14.11 The Matrix Form of a Bézier Patch

In Section 5.7, we formulated a matrix expression for Bézier curves. This approach carries over well to tensor product patches. We can write:

$$\mathbf{b}^{m,n}(u,v) =$$
$$[\,B_0^m(u) \quad \dots \quad B_m^m(u)\,] \begin{bmatrix} \mathbf{b}_{00} & \dots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \dots & \mathbf{b}_{mn} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix}. \tag{14.23}$$

The matrix $\{\mathbf{b}_{ij}\}$, defining the control net, is sometimes called the *geometry matrix* of the patch. If we perform a basis transformation and write the Bernstein polynomials in monomial form, we obtain

$$\mathbf{b}^{m,n}(u,v) = [\,u^0 \quad \dots \quad u^m\,]M^{\mathrm{T}} \begin{bmatrix} \mathbf{b}_{00} & \dots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \dots & \mathbf{b}_{mn} \end{bmatrix} N \begin{bmatrix} v^0 \\ \vdots \\ v^n \end{bmatrix}. \tag{14.24}$$

The square matrices M and N are given by

$$m_{ij} = (-1)^{j-i}\binom{m}{j}\binom{j}{i} \tag{14.25}$$

and

$$n_{ij} = (-1)^{j-i}\binom{n}{j}\binom{j}{i}. \tag{14.26}$$

In the bicubic case, $m = n = 3$, we have

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

For reasons of numerical stability, the use of the monomial form (14.24) is not advisable (see the discussion in Section 24.3). It is included here since it is still in widespread use.

14.12 Nonparametric Patches

This section is the bivariate analog of Section 6.5. Having outlined the main ideas there, we can be brief here. A nonparametric surface is of the form $z = f(x, y)$. It has the parametric representation

$$\mathbf{x}(u, v) = \begin{bmatrix} u \\ v \\ f(u, v) \end{bmatrix},$$

and we restrict both u and v to between zero and one. We are interested in functions f that are in Bernstein form:

$$f(x, y) = \sum_{i}^{m} \sum_{j}^{n} b_{ij} B_i^m(x) B_j^n(y).$$

Using the identity (5.14) for both variables u and v, we see that the Bézier points of $\mathbf{x}$ are given by

$$\mathbf{b}_{ij} = \begin{bmatrix} i/m \\ j/n \\ b_{ij} \end{bmatrix}.$$

The points $(i/m, j/n)$ in the x, y-plane are called *Bézier abscissas* of the function f; the b_{ij} are called its *Bézier ordinates*. A nonparametric Bézier function is not constrained to be defined over the unit square; if a point $\mathbf{p}$ and two vectors $\mathbf{v}$ and $\mathbf{w}$ define a parallelogram in the x, y-plane, then the Bézier abscissas $\mathbf{a}_{ij} \in \mathbb{E}$ of a nonparametric Bézier function over this domain are given by $\mathbf{a}_{ij} = \mathbf{p} + i\mathbf{v} + j\mathbf{w}$. Figure 14.15 gives an example.

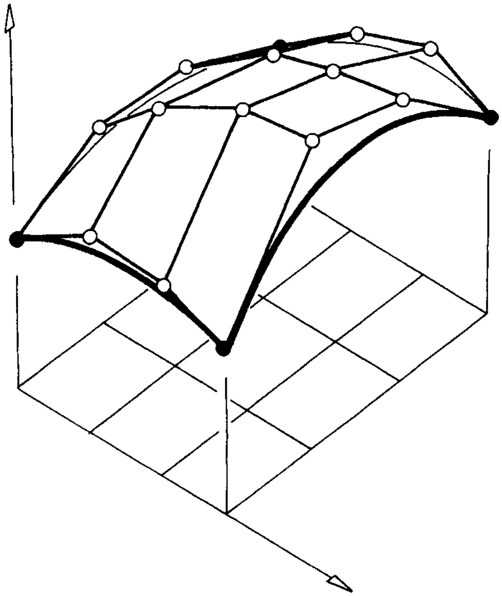


Figure 14.15 Nonparametric patches: the Bézier points are located over a regular partition of the domain rectangle.

Integrals also carry over from the univariate case. With a proof analogous to the one in Section 6.7, we can show that

$$\int_0^1 \int_0^1 \sum_i^m \sum_j^n b_{ij} B_i^m(x) B_j^n(y) = \frac{\sum_i^m \sum_j^n b_{ij}}{(m+1)(n+1)}. \qquad (14.27)$$

14.13 Problems

1 Draw the hyperbolic paraboloid from Figure 14.2 over the square $(-1,-1)$, $(1,-1)$, $(1,1)$, $(-1,1)$. Try to do it manually, that is, without graphics support.

2 Show that the direct de Casteljau algorithm generates surfaces of the form (14.6). Hint: use blossoms.

3 If a Bézier surface is given by its control net, we can use the de Casteljau algorithm to compute $\mathbf{b}^{m,n}(u,v)$ in three ways: by the direct form from Section 14.2, or by the two possible tensor product approaches, computing the coefficients of a u (or v) isoparametric line, and then evaluating that curve at v (or u). Though theoretically equivalent, the computation counts for these methods differ. Work out the details.

***4** Show that Bézier surfaces have bilinear precision: if $\mathbf{b}_{i,j} = \mathbf{x}(\frac{i}{m}, \frac{j}{n})$ and $\mathbf{x}$ is bilinear, then $\mathbf{b}^{m,n}(u, v) = \mathbf{x}(u, v)$ for all u, v and for arbitrary m, n.

***5** Generalize (5.31) to the tensor product case.

P1 Generalize the routine `degree_elevate` to the tensor product case.

P2 Generalize the routine `aitken` to the tensor product case, that is, program tensor product Lagrange interpolation.

P3 The data file `car.dat` contains data points (slightly modified) of four boundary curves of one of the surfaces shown in Color Plate III. Try to fit a Bézier patch (your pick of the degrees!) so that you get close to the corresponding surface in the color plates.

Constructing Polynomial Patches 15

We have discussed the underlying principles of polynomial patches; now it is time to study how they can be used. First applications of tensor product patches go back to General Motors, and Boeing in the United States, and to Rénault and Citröen in France.

15.1 Ruled Surfaces

One of the most elementary surface-building schemes is this: let two Bézier curves $\mathbf{b}(u)$ and $\mathbf{c}(u)$ be given by their control polygons $\mathbf{b}_0, \ldots, \mathbf{b}_n$ and $\mathbf{c}_0, \ldots, \mathbf{c}_n$, and find a surface $\mathbf{r}(u, v)$ that connects them. The simplest such surface will use a linear type of connection; it is called a *ruled surface* or *lofted surface*[1] It is given by

$$\mathbf{r}(u, v) = (1 - v)\mathbf{b}(u) + v\mathbf{c}(u). \tag{15.1}$$

For $v = 0$, it interpolates to $\mathbf{b}(u)$; for $v = 1$, it interpolates to $\mathbf{c}(u)$.

The ruled surface $\mathbf{r}$ is linear in v and of degree n in u. In Bézier form, it becomes

$$\mathbf{r}(u, v) = \begin{bmatrix} 1 - v & v \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 & \ldots & \mathbf{b}_n \\ \mathbf{c}_0 & \ldots & \mathbf{c}_n \end{bmatrix} \begin{bmatrix} B_0^n(u) \\ \vdots \\ B_n^n(u) \end{bmatrix}.$$

[1] The word *lofted* has an interesting history. In the days of completely manual ship design, full-scale drawings were difficult to handle in the design office. These drawings were stored and dealt with in large attics, called *lofts*.

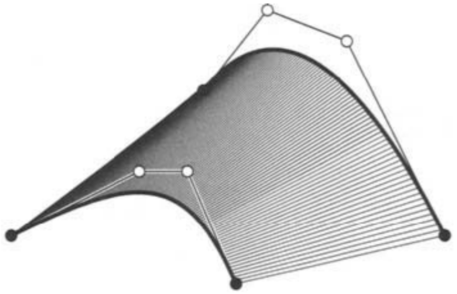


Figure 15.1 Ruled surfaces: corresponding points are connected linearly.

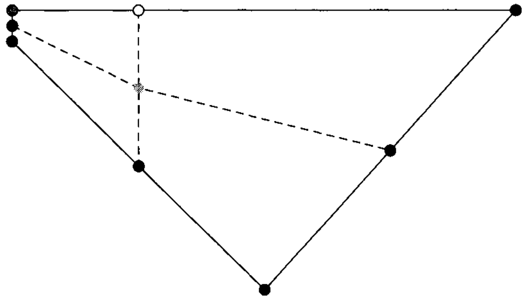


Figure 15.2 Linear interpolation: the average of two convex polygons may not be convex itself.

Figure 15.1 illustrates.

The straight lines on a ruled surface are called its *rulings*. If all rulings are parallel, our ruled surface is *cylindrical*; if all intersect in a point, we have a *conical* surface. Both cylindrical and conical surfaces have zero Gaussian curvature and are special cases of *developable surfaces*; see Chapter 19 for more details.

Linear interpolation between curves may not be as intuitive as might be expected. To see why, consult Figure 15.2. It shows that the average of two convex curves (polygons in the case of the figure) is not necessarily convex. Linear interpolation between curves is thus not shape preserving.

15.2 Coons Patches

Consider the following design situation: four boundary curves of a surface are given, all four in Bézier form, that is, by their control polygons. Let us assume that opposite boundary curves are of the same degrees. The problem: find the control

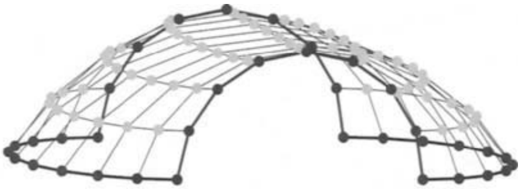


Figure 15.3 Coons patches: an example for $n = 11$ and $m = 9$.

net of a Bézier surface that fits between the boundary curves. This situation is illustrated in Figure 15.3.

A configuration of the given data for $m = n = 3$ looks like this:

$$\begin{matrix} \mathbf{b}_{00} & \mathbf{b}_{01} & \mathbf{b}_{02} & \mathbf{b}_{03} \\ \mathbf{b}_{10} & & & \mathbf{b}_{13} \\ \mathbf{b}_{20} & & & \mathbf{b}_{23} \\ \mathbf{b}_{30} & \mathbf{b}_{31} & \mathbf{b}_{32} & \mathbf{b}_{33} \end{matrix}$$

The construction for the Coons surface mesh relies on the ruled surface form Section 15.1 as well as the bilinear interpolant from Section 14.1. In order to find the control point $\mathbf{b}_{i,j}$, we first construct points on ruled surfaces:

$$\mathbf{b}^{u}_{i,j} = \left(1 - \frac{i}{m}\right)\mathbf{b}_{0,j} + \frac{i}{m}\mathbf{b}_{m,j}$$

and

$$\mathbf{b}^{v}_{i,j} = \left(1 - \frac{j}{n}\right)\mathbf{b}_{i,0} + \frac{j}{n}\mathbf{b}_{i,n}.$$

We also need the point on the bilinear interpolant to the four corner points:

$$\mathbf{b}^{u,v}_{i,j} = [\, 1 - \tfrac{j}{n} \quad \tfrac{j}{n} \,] \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,m} \\ \mathbf{b}_{n,0} & \mathbf{b}_{m,n} \end{bmatrix} \begin{bmatrix} 1 - \frac{i}{m} \\ \frac{i}{m} \end{bmatrix}.$$

The desired control point is found as a combination of these three points:

$$\mathbf{b}_{i,j} = \mathbf{b}^{u}_{i,j} + \mathbf{b}^{v}_{i,j} - \mathbf{b}^{u,v}_{i,j} \tag{15.2}$$

Figure 15.4 illustrates the construction method. Figure 15.5 gives an example of a completed control net.

Coons patches were originally defined in a more general setting, see Chapter 22. Since the Coons technique in this section only deals with piecewise linear

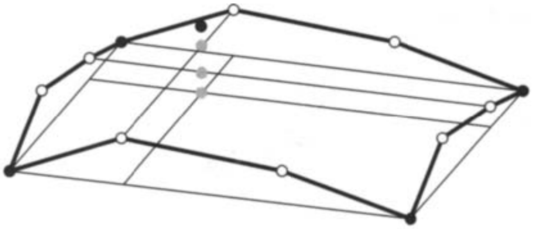


Figure 15.4 Coons patches: the construction. Gray points, from bottom: $\mathbf{b}_{1,2}^{u,v}$, $\mathbf{b}_{1.2}^{u}$, $\mathbf{b}_{1,2}^{v}$. Above them, solid black: $\mathbf{b}_{1,2}$.

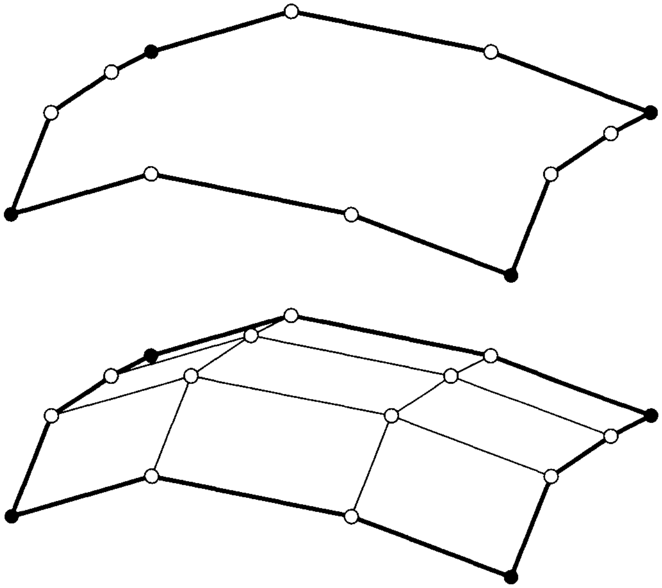


Figure 15.5 Coons patches: an example.

boundary polygons and control meshes, these are often referred to as *discrete Coons patches.*

15.3 Translational Surfaces

A translational surface has the simple structure of being generated by two curves: let $\mathbf{c}_1(u)$ and $\mathbf{c}_2(v)$ be two such curves, intersecting at a common point $\mathbf{a} = \mathbf{c}_1(0) = \mathbf{c}_2(0)$. A translational surface $\mathbf{t}(u, v)$ is then defined by

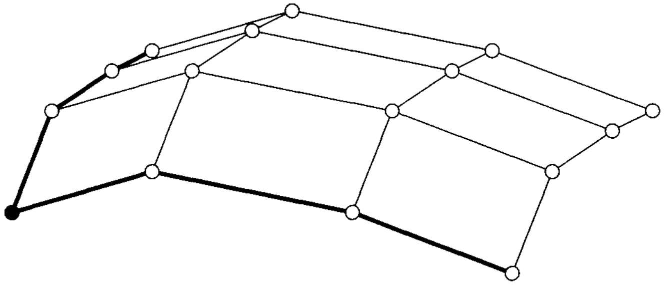


Figure 15.6 Translational surfaces: the Bézier net of a translational tensor product surface. The control polygons in each direction are translates of each other.

$$\mathbf{t}(u,v) = \mathbf{c}_1(u) + \mathbf{c}_2(v) - \mathbf{a}. \tag{15.3}$$

Why the name *translational*? It is justified by considering an arbitrary isoparametric line of the surface, say, $u = \hat{u}$. We obtain $\mathbf{t}(\hat{u},v) = \mathbf{c}_2(v) + [-\mathbf{a} + \mathbf{c}_1(\hat{u})]$, that is, all isoparametric lines are *translates* of one of the input curves; see also Figure 15.6.

An interesting property of translational surfaces is that their twist is identically zero everywhere:

$$\frac{\partial^2}{\partial u \partial v}\mathbf{t}(u,v) \equiv 0.$$

This property follows directly from the definition (15.3). Since both input curves may be arbitrarily shaped, the resulting surface may well have high curvatures. This dispels the myth that zero twists are identical to flat spots. In fact, twists are not related to the shape of a surface—rather, they are a result of a particular parametrization. See also Section 16.3 on twist generation.

A translational surface may be viewed as the solution to an interpolation problem: given two intersecting curves, find a surface that contains them as boundary curves. If four boundary control polygons are given, as in the problem definition for a Coons patch, we can form four translational surfaces, one for each corner. Let us denote by $\mathbf{t}_{i,j}$ the translational surface that interpolates to the boundary curves meeting at the corner (i,j); $i,j \in \{0,1\}$.

Now the Coons patch $\mathbf{x}(u,v)$ can be written as

$$\mathbf{x}(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{t}_{00}(u,v) & \mathbf{t}_{01}(u,v) \\ \mathbf{t}_{10}(u,v) & \mathbf{t}_{11}(u,v) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}. \tag{15.4}$$

This form of the Coons patch is called a *convex combination*. It blends together four surfaces, weighting each with a *weight function*. The weight functions sum to one (a necessity: nonbarycentric combinations are disallowed) and are nonnegative for $u, v \in \{0, 1\}$. Note that the weight functions are zero where the corresponding $\mathbf{t}_{i,j}$ is "wrong."

Translational and Coons surfaces are related in yet another way. If four boundary control polygons are related such that opposite polygons are translates of each other, then the resulting Coons control net will be a translational surface; an example is provided by Figure 15.6.

15.4 Tensor Product Interpolation

We could use curves for both free-form design and for interpolation; the same is true for tensor product patches. As a preparatory step, let us rewrite (14.6) in an equivalent matrix form:

$$\mathbf{x}(u,v) = [\,B_0^m(u) \quad \cdots \quad B_m^m(u)\,] \begin{bmatrix} \mathbf{b}_{00} & \cdots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \cdots & \mathbf{b}_{mn} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix}. \tag{15.5}$$

Suppose now that we are given an $(m+1) \times (n+1)$ array of data points $\mathbf{x}_{ij}$; $0 \le i \le m, 0 \le j \le n$. We want the surface (15.5) to interpolate to them, that is, (15.5) must be true for each pair (u_i, v_j). We thus obtain $(n+1) \times (m+1)$ equations, which we may write concisely as

$$\mathbf{X} = U\mathbf{B}V, \tag{15.6}$$

where

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{00} & \cdots & \mathbf{x}_{0n} \\ \vdots & & \vdots \\ \mathbf{x}_{m0} & \cdots & \mathbf{x}_{mn} \end{bmatrix},$$

$$U = \begin{bmatrix} B_0^m(u_0) & \cdots & B_m^m(u_0) \\ \vdots & & \vdots \\ B_0^m(u_m) & \cdots & B_m^m(u_m) \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}_{00} & \cdots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \cdots & \mathbf{b}_{mn} \end{bmatrix},$$

and

$$V = \begin{bmatrix} B_0^n(v_0) & \cdots & B_0^n(v_n) \\ \vdots & & \vdots \\ B_n^n(v_0) & \cdots & B_n^n(v_n) \end{bmatrix}.$$

Matrices U and V already appeared in Section 7.3; they are *Vandermonde matrices.*

In an interpolation context, the $\mathbf{x}_{ij}$ are known and the coefficients $\mathbf{b}_{ij}$ are unknown. They are found from (15.6) by setting

$$\mathrm{B} = U^{-1}\mathrm{X}V^{-1}. \tag{15.7}$$

The inverse matrices in (15.7) exist since the functions B_i^m and B_j^n are linearly independent.

Equation (15.7) shows how a solution to the interpolation problem *could* be found, but one should not try to invert the matrices U and V explicitly! To solve and understand better the tensor product interpolation problem, we rewrite (15.6) as

$$\mathrm{X} = \mathrm{D}V, \tag{15.8}$$

where we have set

$$\mathrm{D} = U\mathrm{B}. \tag{15.9}$$

Note that D consists of $(m+1)$ rows and $(n+1)$ columns. Equation (15.8) can be interpreted as a family of (m+1) univariate interpolation problems—one for each row of X and D, where D contains the unknowns. Having solved all $(m+1)$ problems (all having the same coefficient matrix V!), we can attack (15.9), since we have just computed D. Equation (15.9) may be interpreted as a family of $(n+1)$ univariate interpolation problems, all having the same coefficient matrix U.

We thus see how the tensor product form allows a significant "compactification" of the interpolation process. Without the tensor product structure, we would have to solve a linear system of order $(m+1)(n+1) \times (m+1)(n+1)$. That is an order of magnitude more complex than solving $m+1$ problems with the same $(n+1) \times (n+1)$ matrix and then solving $n+1$ problems with the same $(m+1) \times (m+1)$ matrix. If $m = n$, the naive approach would thus need $O(m^6)$ computations, whereas the tensor product approach just needs $O(m^3)$. This will be even more dramatic for interpolating spline surfaces.

There is a less algebraic way to describe the tensor product interpolation process as well. Considering (15.8), we see that it may be interpreted as a family of univariate interpolation problems with the same coefficient matrix V. That is

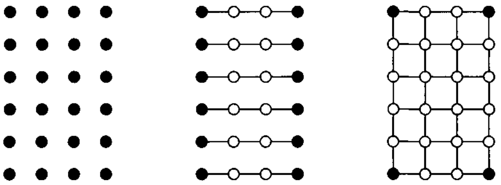


Figure 15.7 Tensor product interpolation: left, the data points; middle, interpolating all rows of data points; right, interpolating all columns from previous step.

to say, we have to solve a univariate interpolation problem for each row of data points, eventually resulting in the elements of D. Then we have to tackle (15.9), meaning we have to solve a family of univariate interpolation problems for each column of coefficients of D. All these problems have the same coefficient matrix U, finally resulting in the desired coefficient matrix B.

The tensor product structure of our problem thus allows for the following two-step solution:

> First, interpolate all rows of data points and write the resulting control points into an intermediate array.
>
> Second, interpolate all columns of that array; the resulting control points represent the solution to our problem.

This approach is illustrated schematically in Figure 15.7.

15.5 Bicubic Hermite Patches

Bézier patches are the tensor product generalization of Bézier curves; in a very similar way, we can also generalize Hermite curves (see Section 7.5) to patches. The input parameters to this patch representation are points, partials, and mixed partials. A bicubic patch in Hermite form is given by

$$\mathbf{x}(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}\mathbf{h}_{i,j}H_i^3(u)H_j^3(v);\quad 0\le u,v\le 1, \tag{15.10}$$

where the H_i^3 are the cubic Hermite functions from Section 7.5, and the $\mathbf{h}_{i,j}$ are given by

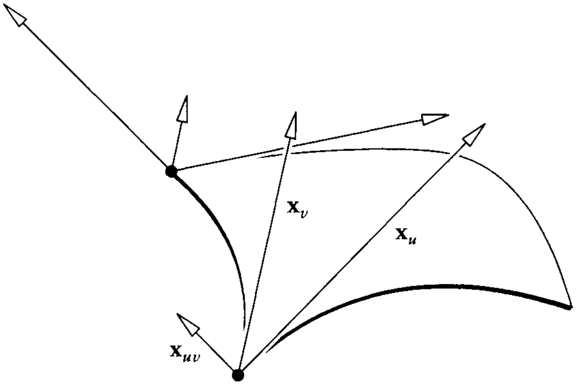


Figure 15.8 Bicubic Hermite patches: some of the data points and vectors.

$$[\mathbf{h}_{i,j}] = \begin{bmatrix} \mathbf{x}(0,0) & \mathbf{x}_v(0,0) & \mathbf{x}_v(0,1) & \mathbf{x}(0,1) \\ \mathbf{x}_u(0,0) & \mathbf{x}_{uv}(0,0) & \mathbf{x}_{uv}(0,1) & \mathbf{x}_u(0,1) \\ \mathbf{x}_u(1,0) & \mathbf{x}_{uv}(1,0) & \mathbf{x}_{uv}(1,1) & \mathbf{x}_u(1,1) \\ \mathbf{x}(1,0) & \mathbf{x}_v(1,0) & \mathbf{x}_v(1,1) & \mathbf{x}(1,1) \end{bmatrix}. \qquad (15.11)$$

The coefficients of this form are shown in Figure 15.8.

Note how the coefficients in the matrix are grouped into four 2 × 2 partitions, each holding the data pertaining to one corner.

As in the curve case, the Hermite form is very sensitive to the u- and v-parameter intervals. If these are not both [0, 1], as before, but rather $a \le u \le b$ and $c \le v \le d$, then our patch becomes

$$\mathbf{x}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{h}_{i,j} H_i^3(s) H_j^3(t); \quad 0 \le s,t \le 1. \qquad (15.12)$$

Here, s and t are local coordinates of the intervals $[a,b]$ and $[c,d]$. The coefficient matrix now changes. With $\Delta_u = b - a$ and $\Delta_v = d - c$, it is:

$$[\mathbf{h}_{i,j}] = \begin{bmatrix} \mathbf{x}(0,0) & \Delta_v\mathbf{x}_v(0,0) & \Delta_v\mathbf{x}_v(0,1) & \mathbf{x}(0,1) \\ \Delta_u\mathbf{x}_u(0,0) & \Delta_u\Delta_v\mathbf{x}_{uv}(0,0) & \Delta_u\Delta_v\mathbf{x}_{uv}(0,1) & \Delta_u\mathbf{x}_u(0,1) \\ \Delta_u\mathbf{x}_u(1,0) & \Delta_u\Delta_v\mathbf{x}_{uv}(1,0) & \Delta_u\Delta_v\mathbf{x}_{uv}(1,1) & \Delta_u\mathbf{x}_u(1,1) \\ \mathbf{x}(1,0) & \Delta_v\mathbf{x}_v(1,0) & \Delta_v\mathbf{x}_v(1,1) & \mathbf{x}(1,1) \end{bmatrix}. \quad (15.13)$$

15.6 Least Squares

In many cases, data points do not come on a rectangular grid that aligns with the patch boundaries. If the data points come from a laser digitizer, for example, we cannot expect them to have any recognizable structure whatsoever, except that they are numbered. We are thus dealing with a set of points $\mathbf{p}_k$, with k ranging from 0 to some number K—this number will typically be in the hundreds or thousands. We also assume that each data point $\mathbf{p}_k$ has a corresponding pair of parameters $\mathbf{u}_k = (u_k, v_k)$.

We wish to find a Bézier patch that fits the data as good as possible. Let it be given by a control net with coefficients $\mathbf{b}_{i,j}$, with $i = 0, \ldots, m$ and $j = 0, \ldots, n$. Before we formulate our problem, we need to introduce a new notation.

Instead of writing a Bézier patch as a matrix product (14.23), we may introduce a *linearized* notation. We will number all control points linearly, simply counting them as we traverse the control net row by row. For example, we may write a bilinear patch as

$$\mathbf{x}(u, v) =$$

$$[\, B_0^1(u)B_0^1(v) \quad B_0^1(u)B_1^1(v) \quad B_1^1(u)B_0^1(v) \quad B_1^1(u)B_1^1(v) \,] \begin{bmatrix} \mathbf{b}_{0,0} \\ \mathbf{b}_{0,1} \\ \mathbf{b}_{1,0} \\ \mathbf{b}_{1,1} \end{bmatrix}.$$

This equation has, for the general case, $(m + 1)(n + 1)$ terms. Using this linear ordering, our patch equation may be written as

$$\mathbf{x}(u, v) = [\, B_0^m(u)B_0^n(v), \ldots, B_m^m(u)B_n^n(v) \,] \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}. \qquad (15.14)$$

Returning to our problem, we would like to achieve that each data point lies on the approximating surface. For the kth data point $\mathbf{p}_k$, this becomes $\mathbf{p}_k = \mathbf{x}(\mathbf{u}_k)$ or

$$\mathbf{p}_k = [\, B_0^m(u_k)B_0^n(v_k), \ldots, B_m^m(u_k)B_n^n(v_k) \,] \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}. \qquad (15.15)$$

Combining all $K + 1$ of these equations, we obtain

$$\begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} B_0^m(u_0)B_0^n(v_0) & \dots & B_m^m(u_0)B_n^n(v_0) \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ B_0^m(u_K)B_0^n(v_K) & \dots & B_m^m(u_K)B_n^n(v_K) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}. \tag{15.16}$$

We abbreviate this to

$$\mathbf{P} = M\mathbf{B}. \tag{15.17}$$

These are $K+1$ equations in $(m+1)(n+1)$ unknowns. For the example of the bicubic case, we have 16 unknowns, but typically several hundred data points—thus the linear system (15.17) is *overdetermined.* It will in general not have an exact solution, but a good approximation is found by forming

$$M^{\mathrm{T}}\mathbf{P} = M^{\mathrm{T}}M\mathbf{B}. \tag{15.18}$$

Our linear system has a square coefficient matrix $M^{\mathrm{T}}M$, with $(m+1)(n+1)$ rows and columns.

In the bilinear case, we would thus have a linear system with four equations in four unknowns, as shown in Example 15.1. In the bicubic case, we would have

Example 15.1 **Bilinear least squares approximation.**

We give a very simple example for $m = n = 1$ and $K = 4$. Let the data points be

$$\mathbf{p}_0 = \begin{bmatrix} -2 \\ -2 \\ 1 \end{bmatrix}, \quad \mathbf{p}_1 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{p}_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{p}_4 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1 \end{bmatrix}.$$

Let their parameter values be

$$\begin{aligned} \mathbf{u}_0 &= (0, 0) \\ \mathbf{u}_1 &= (1, 1) \\ \mathbf{u}_2 &= (1, 0) \\ \mathbf{u}_3 &= (0, 1) \\ \mathbf{u}_4 &= (0.5, 0.5). \end{aligned}$$

Example 15.1 **Continued**

The overdetermined linear system (15.17) is given by

$$\begin{bmatrix} \begin{bmatrix} -2 \\ -2 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0.5 \\ 0.5 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} \\ \mathbf{b}_{0,1} \\ \mathbf{b}_{1,0} \\ \mathbf{b}_{1,1} \end{bmatrix}.$$

Multiplying both sides by the transpose of the coefficient matrix yields

$$\begin{bmatrix} \begin{bmatrix} -2 \\ -1.875 \\ 1.25 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1.125 \\ 0.25 \end{bmatrix} \\ \begin{bmatrix} 1.125 \\ 0.125 \\ 0.25 \end{bmatrix} \\ \begin{bmatrix} 2.125 \\ 2.125 \\ 1.25 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1+x & x & x & x \\ x & 1+x & x & x \\ x & x & 1+x & x \\ x & x & x & 1+x \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} \\ \mathbf{b}_{0,1} \\ \mathbf{b}_{1,0} \\ \mathbf{b}_{1,1} \end{bmatrix},$$

where $x = 1/16$.

Solving the linear system (actually one each for x, y, z) yields

$$\mathbf{b}_{0,0} = \begin{bmatrix} -2.062 \\ -1.95 \\ 1.1 \end{bmatrix}, \ \mathbf{b}_{0,1} = \begin{bmatrix} -0.063 \\ 1.050 \\ 0.1 \end{bmatrix}, \ \mathbf{b}_{1,0} = \begin{bmatrix} 1.062 \\ 0.050 \\ 0.1 \end{bmatrix}, \ \mathbf{b}_{1,1} = \begin{bmatrix} 2.062 \\ 2.050 \\ 1.1 \end{bmatrix}.$$

This surface does not go through any of the data points exactly, but it is reasonably close to them.

to solve a linear system with 16 equations in 16 unknowns. A note of caution: if the number of data points is very large (10^5 or more), then the normal equations become ill-conditioned and the least squares problem may not be solvable.

15.7 Finding Parameter Values

In a practical setting, we would not typically be given the parameter values $\mathbf{u}_k = (u_k, v_k)$. Finding good values for the $\mathbf{u}_k$ is not an easy problem.

But often, the data points can be projected into a plane; let us assume they can be projected into the x, y-plane for simplicity. Each $\mathbf{p}_k$ is projected by simply dropping its z-coordinate, leaving a pair (x_k, y_k). We scale the (x_k, y_k) so that they fit into the unit square. Then we can simply set $u_k = x_k$ and $v_k = y_k$.

Another solution may be obtained by using a *triangulation* of the data points. This scenario is realistic for data obtained using a laser digitizer. Assuming that the triangulation is isomorphic to the unit square, we can construct a triangulation in the unit square with the same connectivity as the given one in 3D.

The following method is due to Floater [236]. First, a convex polygon is built in the (u, v) unit square with as many vertices as the 3D mesh has boundary vertices. This polygon is somewhat arbitray; a circle or the boundary of the unit square is a good candidate for forming it. In this way, we assign 2D parameters to the 3D mesh boundary points.

Next, consider any interior point $\mathbf{u}_r$ of the 2D mesh with n_r neighbors. We call these neighbors $\mathbf{u}_{r,1}, \ldots, \mathbf{u}_{r,n_r}$. For a "nice" triangulation, the following condition should be satisfied: all interior $\mathbf{u}_r$ may be expressed as[2]

$$\mathbf{u}_r = \frac{1}{n_r} \sum_{i=0}^{n_r} \mathbf{u}_{r,n_i}. \qquad (15.19)$$

We now observe that there are as many equations (15.19) as there are interior points in the mesh. Some of these equations involve boundary points, others will not. Thus we have a linear system for the interior $\mathbf{u}_k$, with as many equations as there are interior points. It is always solvable.

[2] Triangulations with this property are the result of *Laplacian smoothing* of a mesh.

15.8 Shape Equations

The solution to a least squares problem may be close to the data points, yet the control net might "behave badly," similar to the curve case. A way to combat such behavior is to invoke *shape equations*. These are conditions that a "good" control net would satisfy. Out of many possibilities, we choose the following: a translational surface (see 15.3) is characterized by the fact that its twist vanishes everywhere. For the control net, this means

$$\Delta^{1,1}\mathbf{b}_{i,j} = 0; \quad i = 0, \ldots, m-1, \ j = 0, \ldots, n-1.$$

When we add these equations to our overdetermined linear system (15.17), we will be less faithful to the data points, but achieve a better shape of the control net. In practice, we would weight the shape equations, maybe by a factor of 0.1, just as we did for the curve case.

15.9 A Problem with Unstructured Data

The least squares approximation problem for Bézier patches leads to an interesting question: what happens if the number of data points equals the number of unknowns? In this case, we do not have to use the normal equations; the problem directly yields a linear system with a square coefficient matrix.

Although this interpolation scenario appears simpler than the least squares problem, it has a potential for a serious pitfall.

Our linear system is given by

$$\begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} B_0^m(u_0)B_0^n(v_0) & \ldots & B_m^m(u_0)B_n^n(v_0) \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ B_0^m(u_K)B_0^n(v_K) & \ldots & B_m^m(u_K)B_n^n(v_K) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}, \qquad (15.20)$$

but now we have $K = (m+1)(n+1)$.

The assignment of parameter values (u_i, v_i) to the data points $\mathbf{p}_i$ is (theoretically) arbitrary. We now perform a "thought experiment."[3] Take two data points $\mathbf{p}_i$ and $\mathbf{p}_j$. They have parameters (u_i, v_i) and (u_j, v_j). Now, leaving the data points untouched, start changing their parameter values. Do this as follows: find the circle having (u_i, v_i) and (u_j, v_j) as diameter and move both (u_i, v_i) and (u_j, v_j)

[3] Or "Gedankenexperiment" as coined by A. Einstein.

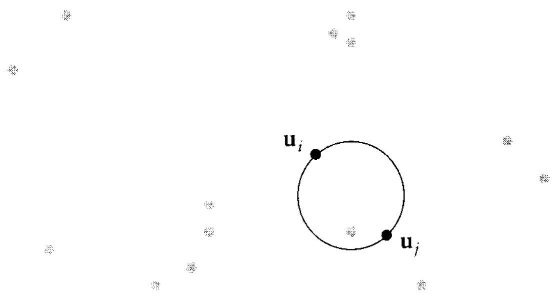


Figure 15.9 Unstructured data: two parameter locations $\mathbf{u}_i = (u_i, v_i)$ and $\mathbf{u}_j = (u_j, v_j)$ are interchanged.

along this circle in a counterclockwise fashion until they have interchanged their locations. Figure 15.9 illustrates.

The determinant of the linear system (15.20) has now changed its sign: two of its rows are interchanged. It follows that somewhere during the interchanging procedure, the determinant must have been zero, corresponding to an unsolvable interpolation problem.

For an arbitrary collection of data point parameters, we cannot know if we might be in or near an unsolvable situation. Hence this interpolation problem is ill-posed.

15.10 Implementation

The following is the header for a program to plot a tensor product Bézier surface, in fact, a rational one. If the polynomial case is desired, just set all weights to unity.

```
void plot_ratsurf(bx,by,bw,degree_u,degree_v,u_points,v_points,
                         scale_x,scale_y)
/*      plots v_points isoparametric
        curves of the  rat Bez surface, each with u_points
        points on it.

Input:  bx, by:          arrays with x- and y- coordinates of
                         control net.
        degree_u,degree_v: degrees in u- and v- direction
        u_points,v_points: plot resolution
        scale_x,scale_y:   scale factor for postscript.

Output: postscript file
*/
```

Next is a routine that fits a bilinearly blended Coons patch in between four boundary control polygons, as described in Section 15.2. The routine works on one coordinate only, and will have to be called separately for the x-, y-, and z-components of a control net.

```
void netcoons(net,rows,columns)
/* Uses bilinear Coons blending to complete a control
net of which only the four boundary polygons are used as input.
Works for one coordinate only.
Input:     net:            control net.
           rows, columns: dimensions of net.
Output:    net:            the completed net, with the old boundaries.
*/
```

15.11 Problems

1 Justify that in tensor product interpolation (Section 15.4), it does not matter if we start with the row interpolation process or with the column interpolation process. Give computation counts for both strategies. (In general, they are not equal!)

2 Generalize Lagrange interpolation to the tensor product case.

*** 3** Generalize quintic Hermite interpolation to the tensor product case.

*** 4** Suppose we want to find a parametrization $\{u_i\}$ for a tensor product interpolant. We may parametrize all rows of data points and then form the averages of the parametrizations thus obtained. Or we could average all rows of data points, for example, by setting $\mathbf{p}_i = \sum_j \frac{j}{n}\mathbf{x}_{i,j}$ and we could then parametrize the $\mathbf{p}_i$. Do we get the same result? Discuss both methods.

P1 Figure 7.3 shows how Lagrange interpolation behaves badly for curves. Write a program to exhibit this effect for tensor product patches.

Composite Surfaces

16

Tensor product Bézier patches were under development in the early 1960s; at about the same time, people started to think about piecewise surfaces. One of the first publications was de Boor's work on bicubic splines [136] in 1962. Almost simultaneously, and apparently unaware of de Boor's work, J. Ferguson [231] implemented piecewise bicubics at Boeing. His method was used extensively, although it had the serious flaw of using only zero corner twist vectors. An excellent account of the early industrial use of piecewise bicubics is the article by G. Peters [462].

16.1 Smoothness and Subdivision

Let $\mathbf{x}(u,v)$ and $\mathbf{y}(u,v)$ be two patches, defined over $[u_{I-1}, u_I] \times [v_J, v_{J+1}]$ and $[u_I, u_{I+1}] \times [v_J, v_{J+1}]$, respectively. They are r times continuously differentiable across their common boundary curve $\mathbf{x}(u_I, v) = \mathbf{y}(u_I, v)$ if all u-partials up to order r agree there:

$$\frac{\partial^r}{\partial u^r}\mathbf{x}(u,v)\bigg|_{u=u_I} = \frac{\partial^r}{\partial u^r}\mathbf{y}(u,v)\bigg|_{u=u_I}. \tag{16.1}$$

Now suppose both patches are given in Bézier form; let the control net of the "left" patch be $\{\mathbf{b}_{ij}\}; 0 \le i \le m, 0 \le j \le n$ and those of the "right" patch be $\{\mathbf{b}_{ij}\}; m \le i \le 2m, 0 \le j \le n$. We can then invoke (14.13) for the cross boundary derivative of a Bézier patch. That formula is in local coordinates. To make the transition to global coordinates (u, v), we must invoke the chain rule:

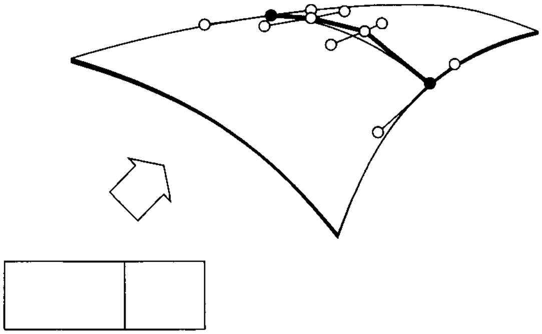


Figure 16.1 C^1 continuous Bézier patches: the shown control points must be collinear and all be in the same ratio.

$$\left(\frac{1}{\Delta_{I-1}}\right)^r \sum_{j=0}^{n} \Delta^{r,0}\mathbf{b}_{m-r,j}B_j^n(v) = \left(\frac{1}{\Delta_I}\right)^r \sum_{j=0}^{n} \Delta^{r,0}\mathbf{b}_{m,j}B_j^n(v), \tag{16.2}$$

where $\Delta_I = u_{I+1} - u_I$. Since the $B_j^n(v)$ are linearly independent, we may compare coefficients:

$$\left(\frac{1}{\Delta_{I-1}}\right)^r \Delta^{r,0}\mathbf{b}_{m-r,j} = \left(\frac{1}{\Delta_I}\right)^r \Delta^{r,0}\mathbf{b}_{m,j}; \quad j = 0, \ldots, n.$$

This is the C^r condition for Bézier curves, applied to all $n+1$ rows of the composite Bézier net. We thus have the C^r condition for composite Bézier surfaces: *two adjacent patches are C^r across their common boundary if and only if all rows of their control net vertices can be interpreted as polygons of C^r piecewise Bézier curves.* We have again succeeded in reducing a surface problem to several curve problems. The smoothness conditions apply analogously to the v-direction.

The basic theory for $r = 1$ is illustrated in Figure 16.1. An example of two C^1bicubic patches is shown in Figure 16.2.

The C^1 condition states that for every j, the polygon formed by $\mathbf{b}_{0,j}, \ldots, \mathbf{b}_{2m,j}$ is the control polygon of a C^1 piecewise Bézier curve. For this to be the case, the three points $\mathbf{b}_{m-1,j}, \mathbf{b}_{m,j}, \mathbf{b}_{m+1,j}$ must be collinear and in the same ratio for all j. Simple collinearity is *not* sufficient: composite surfaces that have $\mathbf{b}_{m-1,j}, \mathbf{b}_{m,j}, \mathbf{b}_{m+1,j}$ collinear for each j but not in the same ratio will in general not be C^1. Moreover, they will not even have a continuous tangent plane. The rigidity of the C^1 condition can be a serious obstacle in the design of surfaces that consist of a network of Bézier patches (or of piecewise polynomial patches in other representations).

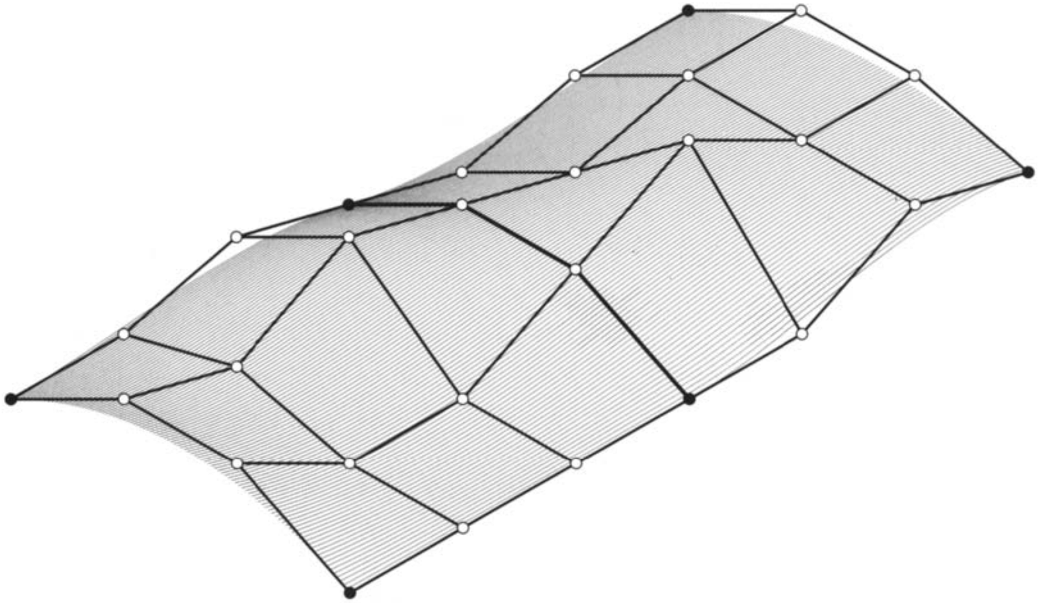


Figure 16.2 C^1 continuous Bézier patches: two C^1 bicubic patches.

We already saw how to use blossoms to subdidivide Bézier patches using (14.15). Here we treat an important special case more geometrically. Suppose the domain rectangle of a Bézier patch is subdivided into two subrectangles by a straight line $u = \hat{u}$. That line maps to an isoparametric curve on the patch, which is thus subdivided into two subpatches. We wish to find the control nets for each patch. These two patches, being part of one global surface, meet with C^n continuity. Therefore, all their rows of control points must be control polygons of C^n piecewise nth degree curves. Those curves are related to each other by the univariate subdivision process from Section 5.4.

We now have the following subdivision algorithm: interpret all rows of the control net as control polygons of Bézier curves. Subdivide each of these curves at $u = \hat{u}$. The resulting control points form the two desired control nets. For an example, see Figure 16.3.

Subdivision along an isoparametric line $v = \hat{v}$ is treated analogously. If we want to subdivide a patch into four subpatches that are generated by two isoparametric lines $u = \hat{u}$ and $v = \hat{v}$, we apply the subdivision procedure twice. It does not matter in which direction we subdivide first.

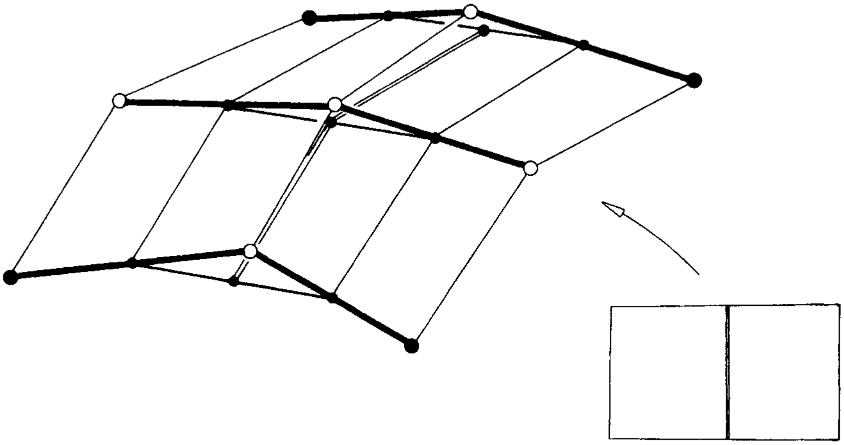


Figure 16.3 Subdivision of a Bézier patch: all rows are subdivided using the de Casteljau algorithm.

16.2 Tensor Product B-Spline Surfaces

B-spline surfaces (both rational and nonrational) play an important role in current surface design methods and will be discussed here in more detail. A parametric tensor product B-spline surface may be written as

$$\mathbf{x}(u,v)=\sum_i\sum_j \mathbf{d}_{ij}N_i^m(u)N_j^n(v), \tag{16.3}$$

where we assume that one knot sequence in the u-direction and one in the v-direction are given. A typical bicubic surface, corresponding to triple end knots and consisting of 3 × 4 bicubic patches, is shown in Figure 16.4.

For curves, triple end knots meant that the first and last two B-spline control points were also Bézier control points; the same is true here. The B-spline control points $\mathbf{d}_{ij}$ for which i or j equal 0 or 1, are also control vertices of the piecewise Bézier net of the surface. Thus they determine the boundary curves and the cross boundary derivatives.

Since a bicubic B-spline surface is a collection of bicubic patches, how can we find the Bézier net of each patch? The answer to this question may be useful for the conversion of a B-spline data format to the piecewise Bézier form. It is also relevant if we decide to evaluate a B-spline surface by first breaking it down into bicubics. The solution arises, as usual for tensor products, from the breakdown of this surface problem into a series of curve problems. If we rewrite (16.3) as

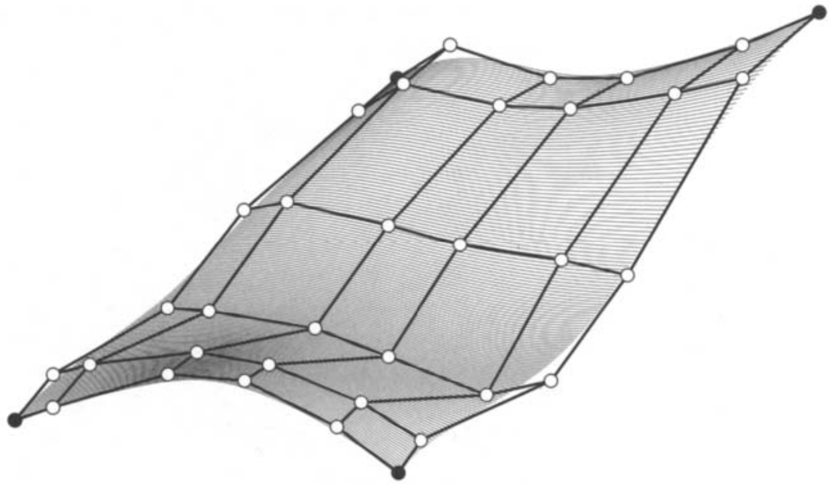


Figure 16.4 Bicubic B-spline surfaces: a surface consisting of 3 × 3 patches together with its control net.

$$\mathbf{x}(u,v) = \sum_i N_i^m(v) \left[\sum_j \mathbf{d}_{ij} N_j^n(u) \right],$$

we see that for each i the sum in square brackets describes a B-spline curve in the variable u. We may convert it to Bézier form by using the univariate methods described in Chapter 8. This corresponds to interpreting the B-spline control net row by row as univariate B-spline polygons and then converting them to piecewise Bézier form. The Bézier points thus obtained may be interpreted—column by column—as B-spline polygons, which we may again transform to Bézier form one by one. This final family of Bézier polygons constitutes the piecewise Bézier net of the surface, as illustrated in Figure 16.5.

Needless to say, we could have started the B-spline–Bézier conversion process column by column. From the Bézier form, we may now transform to any other piecewise polynomial forms, such as the piecewise monomial or the piecewise Hermite form.

B-spline curves may be open or closed; the same is true for surfaces. Yet B-spline surfaces may be closed in two different ways: we may form surfaces with the connectivity of a cylinder or with that of a torus. No tensor product surface, however, can have the connectivity of a "double torus" or more complicated surfaces. In fact, even a surface with the topology of a sphere is not representable as a tensor product surface, at least not as one without degeneracies.

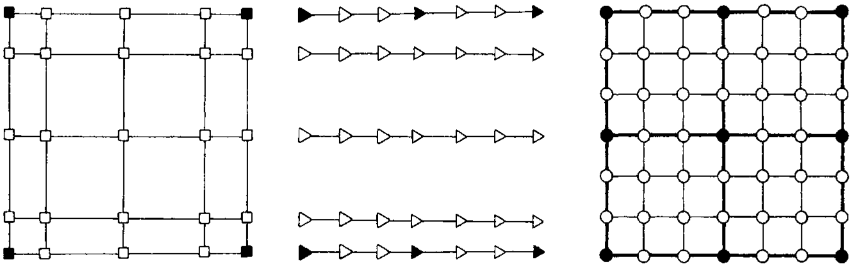


Figure 16.5 Bringing a bicubic B-spline surface into piecewise bicubic Bézier form: we first perform B-spline–Bézier curve conversion row by row, then column by column.

16.3 Twist Estimation

Suppose that we are given a rectangular network of points $\mathbf{x}_{IJ}$; $0 \le I \le M$, $0 \le J \le N$ and two sets of parameter values u_I and v_J. We want a C^1 piecewise cubic surface $\mathbf{x}(u, v)$ that interpolates to the data points:

$$\mathbf{x}(u_I, v_J) = \mathbf{x}_{IJ}.$$

For a solution, we use curve methods wherever possible. We will first fit piecewise cubics to all rows and columns of data points using methods that were developed in Chapter 9. We must keep in mind, however, that all curves in the u-direction have the same parametrization, given by the u_I; the v-curves are all defined over the v_J.

Creating a network of C^1 (or C^2) piecewise cubics through the data points is only the first step toward a surface, however. Our aim is a C^1 piecewise bicubic surface, and so far we have constructed only the boundary curves for each patch. This constitutes 12 data out of the 16 needed for each patch. Figure 16.6 illustrates the situation. In Bézier form, we are still missing four interior Bézier points per patch, namely, $\mathbf{b}_{11}, \mathbf{b}_{21}, \mathbf{b}_{12}, \mathbf{b}_{22}$; in terms of derivatives, we must still determine the *corner twists* of each patch; for a definition, see Section 14.10.

We now list a few methods to determine the missing twists.

Zero twists: Historically, this is the first twist estimation "method." It appears, hidden in a set of formulas in pseudocode, in the paper by Ferguson [231]. Ferguson did not comment on the effects that this choice of twist vectors might have.

"Nice" surfaces exist that have identically vanishing twists—these are translational surfaces (see Figure 15.6). If the boundary curves of a patch are pairwise

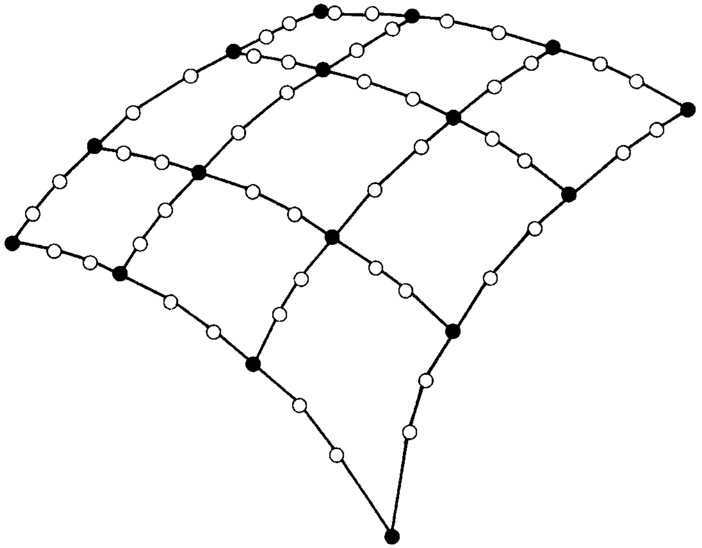


Figure 16.6 Piecewise bicubic interpolation: after a network of curves has been created, one still must determine four more coefficients per patch. A network of 3 × 3 patches is shown.

related by translations, then the assignment of zero twists is a good idea, but not otherwise. In these other cases, the boundary curves are *not* the generating curves of a translational surface. If zero twists are assigned, the generated patch *will* locally behave like a translational surface, giving rise to the infamous "flat spots" of zero twists. The effects of zero twists will be illustrated in Chapter 23.

If a network of patches has to be created, this choice of twists automatically guarantees C^1 continuity of the overall surface. Thus it is mathematically "safe," but does not guarantee "nice" shapes.

Adini's twists: This method has been introduced into the CAGD literature through the paper by Barnhill, Brown, and Klucewicz [28], based on a scheme ("Adini's rectangle") from the finite element literature. The basic idea is this: the four cubic boundary curves define a bilinearly blended Coons patch (see Chapter 22), which happens to be a bicubic patch itself. Take the corner twists of that patch to be the desired twist vectors.

If a network of patches has to be generated, the preceding Adini's twists would not guarantee a C^1 surface. A simple modification is necessary: let four patches meet at a point, as in Figure 16.7. The four outer boundary curves of the four patches again define a bilinearly blended Coons patch. This Coons patch (consisting of four bicubics) has a well-defined twist at the parameter value where the four bicubics meet. Take that twist to be the desired twist. It is given by

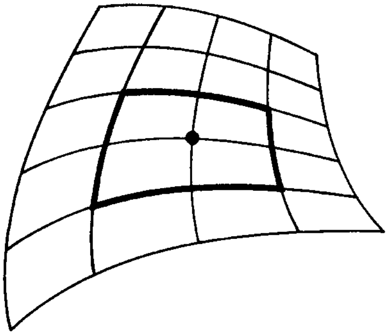


Figure 16.7 Adini's twist: the outer boundary curves of four adjacent patches define a Coons surface; its twist at the "middle" point is Adini's twist.

$$\begin{aligned}\mathbf{x}_{uv}(u_I, v_J) \\ &= \frac{\mathbf{x}_v(u_{I+1}, v_J) - \mathbf{x}_v(u_{I-1}, v_J)}{u_{I+1} - u_{I-1}} \\ &+ \frac{\mathbf{x}_u(u_I, v_{J+1}) - \mathbf{x}_u(u_I, v_{J-1})}{v_{J+1} - v_{J-1}} \\ &- \frac{\mathbf{x}(u_{I+1}, v_{J+1}) - \mathbf{x}(u_{I-1}, v_{J+1}) - \mathbf{x}(u_{I+1}, v_{J-1}) + \mathbf{x}(u_{I-1}, v_{J-1})}{(u_{I+1} - u_{I-1})(v_{J+1} - v_{J-1})}.\end{aligned}$$

It is easy to check that Adini's method, applied to patch boundaries of a translational surface, yields zero twists, which is desirable for that situation. Adini's twist is a reasonable choice because, considered as an interpolant, it reproduces all bivariate polynomials of the form $uv^j, u^iv; i, j \in \{0, 1, 2, 3\}$, which is a surprisingly large set.[1]

Figure 16.8 compares zero twists and Adini's twists if only one patch is used. The zero twists give rise to undesirable distortions.

Bessel twists: This method estimates the twist at $\mathbf{x}(u_I, v_J)$ to be the twist of the biquadratic interpolant to the nine points $\mathbf{x}(u_{I+r}, v_{J+s}); r, s \in \{-1, 0, 1\}$. Since a biquadratic patch has a bilinear twist, Bessel's twist is the bilinear interpolant to the twists of the four bilinear patches formed by the nine points. Those twists are given by

$$\mathbf{q}_{I,J} = \frac{\Delta^{1,1}\mathbf{x}(u_I, v_J)}{\Delta_I \Delta_J},$$

[1] This is why this twist is called, in the context of finite elements, a *serendipity element.*

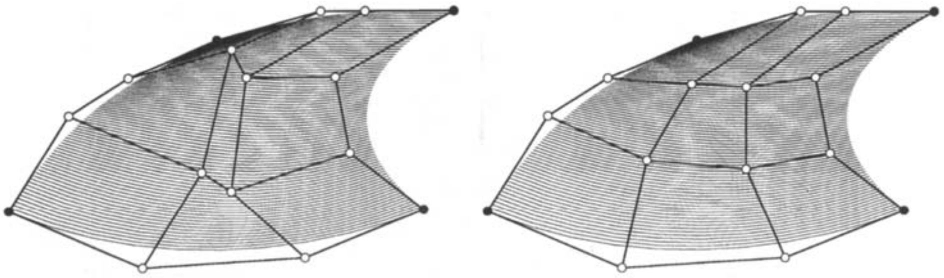


Figure 16.8 Twist estimation: the four interior Bézier points are computed to yield zero corner twists (left), and then according to Adini's method (right).

and Bessel's twist can now be written

$$\mathbf{x}_{uv}(u_I, v_J) = [\, 1-\alpha_I \quad \alpha_I \,] \begin{bmatrix} \mathbf{q}_{I-1,J-1} & \mathbf{q}_{I-1,J} \\ \mathbf{q}_{I,J-1} & \mathbf{q}_{I,J} \end{bmatrix} \begin{bmatrix} 1-\beta_J \\ \beta_J \end{bmatrix},$$

where

$$\alpha_I = \frac{\Delta_{I-1}}{u_{I+1} - u_{I-1}}, \qquad \beta_J = \frac{\Delta_{J-1}}{v_{J+1} - v_{J-1}}.$$

Other methods for twist estimation exist, including Brunet [94], Selesnick [570], Hagen and Schulze [303], and Farin and Hagen [206].

16.4 Bicubic Spline Interpolation

In Section 15.4, we saw how to fit an interpolating Bézier patch to a rectangular array of data points. In *real life*, this would not happen too often—rather we would use tensor product bicubic B-spline surfaces. The principles from Section 15.4 carry over for this case easily, and no new theory has to be developed.

Suppose we have $(K+1) \times (L+1)$ data points $\mathbf{x}_{IJ}$ and two knot sequences $u_0, \ldots, u_K$ and $v_0, \ldots, v_L$. Our development is illustrated in Figure 16.9. For each row of data points, we prescribe two end conditions (e.g., by specifying tangent vectors or Bézier points) and solve the univariate B-spline interpolation problem as described in Section 9.4. Since all these interpolation problems use the same tridiagonal coefficient matrix, an $L-U$ decomposition should be performed before the row-by-row loop is entered. We thus produce the elements of the matrix D, marked by triangles in Figure 16.9.

We now take every column of D and perform univariate B-spline interpolation on it, again by prescribing end conditions such as clamped end tangents or Bessel

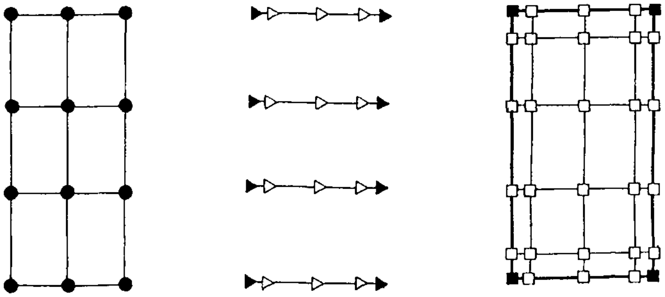


Figure 16.9 Tensor product bicubic spline interpolation: the solution is obtained in a two-step process.

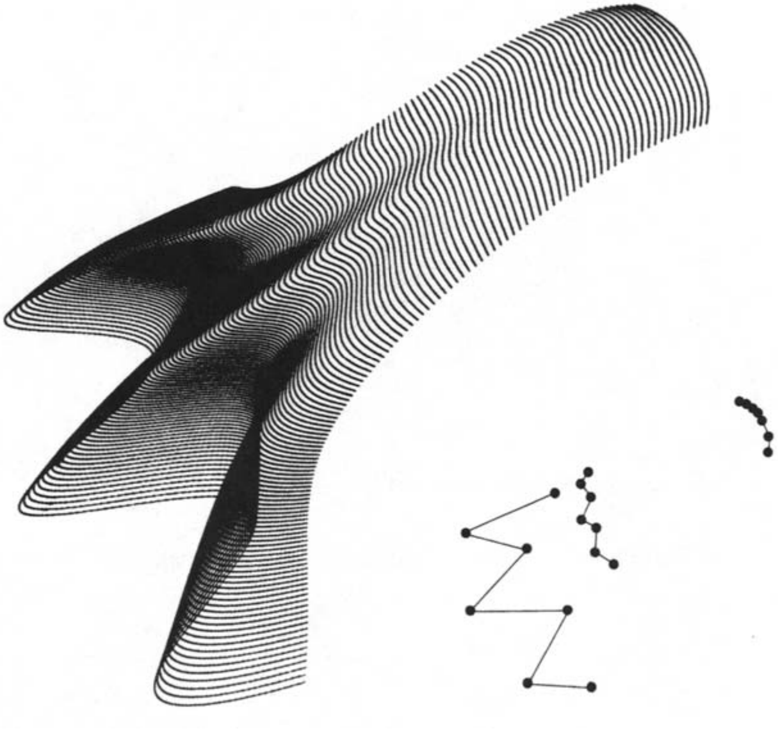


Figure 16.10 Tensor product bicubic spline interpolation: the given data, lower right (scaled down), and the solution, using Bessel end conditions and uniform parametrizations.

tangents. The resulting control points constitute the desired B-spline control net. An example is shown in Figure 16.10. In it, the data points are connected in the u-direction—this is just to highlight the structure of the data.

The final B-spline control net has two more rows and columns than **X**.[2] This is due to the end conditions; to resolve the apparent discrepancy, we may think of **X** as having two additional rows and columns that constitute the end condition data. This concept is implemented in the attached code.

Although mathematically equivalent, the two processes—first row by row, then column by column; or first column by column, then row by row—do not yield the same computation count if $K \neq L$.

16.5 Finding Knot Sequences

Although tensor product spline interpolation is very elegant, its use is limited to cases where the data points possess a rectangular structure. When the data points deviate from a *nice* grid, the problem of finding an appropriate parametrization is not easy; it may not have a solution at all. In the curve case (Section 9.6), we were able to devise several methods that assigned parameter values to the given data points. So why not take those methods and apply them to the tensor product case in much the same way in which we generalized curve methods to their tensor product counterparts? The problem is that we have to produce *one* set of parameter values for *all* isoparametric curves in the u-direction; the same holds for the v-direction.

We may endow each isoparametric curve (in the u-direction, say) with a parametrization from Section 9.6. To arrive at *one* parametrization for all of them, we may then carry out some averaging process. Such an approach will produce acceptable results only if all of our isoparametric curves have the same shape characteristics, that is, if they essentially yield the same parametrization. This is, however, not always the case, as Figures 16.11, 16.12, and 16.13 illustrate.[3]

Are there ways out of the dilemma? Not if we have unevenly distributed data and insist on bicubic spline interpolation. If we are willing to go to higher degrees and to replace C^1 or C^2 continuity by G^1 continuity (see Chapter 20), then several methods exist; see the literature cited in that chapter.

[2] This is inherited from the curve case: there one gets $L + 2$ control points for L data points.

[3] Another interesting phenomenon may be observed here: note how the first and the third of this set of surfaces have varying densities in their plots. The reason is that each cubic isoparametric curve was plotted in 90 increments on a pen plotter. With very unequal parameter spacing, this generates abruptly varying spacing on the curves.

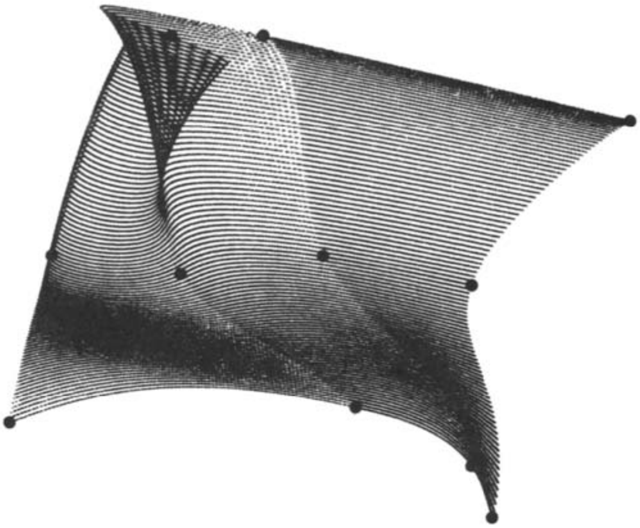


Figure 16.11 Finding knots for bicubic splines: all "horizontal" isoparametric curves have knots $u_i = [0, 4, 5.5, 6.0]$. Note that the bottom curve has a reasonable shape.

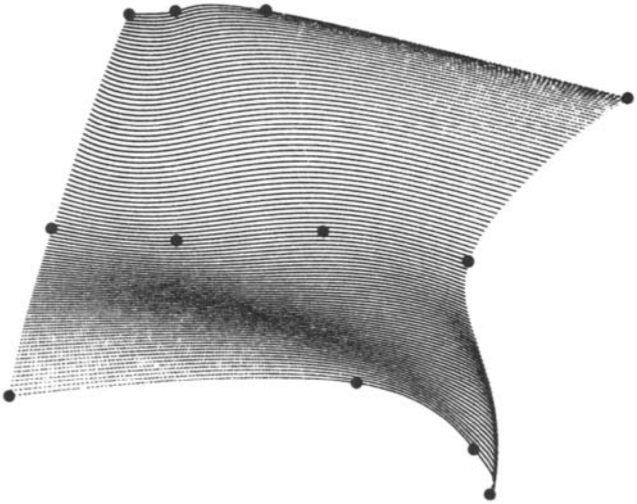


Figure 16.12 Finding knots for bicubic splines: all "horizontal" isoparametric curves have knots $u_i = [0, 1, 2, 3]$.

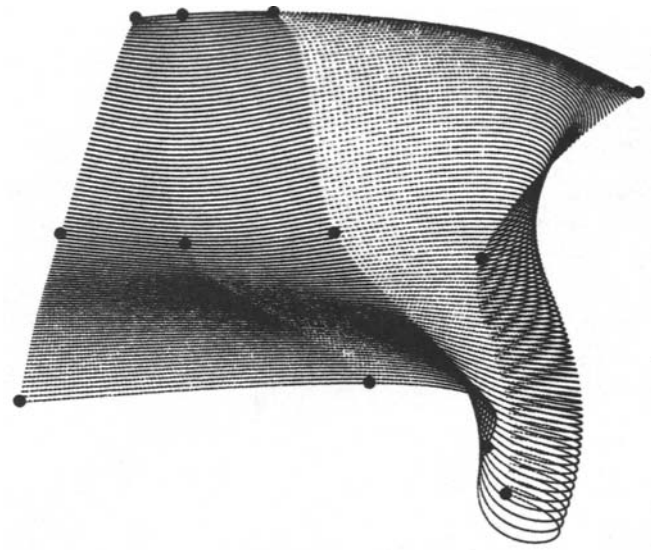


Figure 16.13 Finding knots for bicubic splines: all "horizontal" isoparametric curves have knots $u_i = [0, 0.5, 1.5, 6.5]$. Now the top curve has a good shape.

16.6 Rational Bézier and B-Spline Surfaces

We can generalize Bézier and B-spline surfaces to their rational counterparts in much the same way as we did for the curve cases. In other words, we define a rational Bézier or B-spline surface as the projection of a 4D tensor product Bézier or B-spline surface. Thus, the rational Bézier patch takes the form

$$\mathbf{x}(u, v) = \frac{\sum_i \sum_j w_{i,j} \mathbf{b}_{i,j} B_i^m(u) B_j^n(v)}{\sum_i \sum_j w_{i,j} B_i^m(u) B_j^n(v)}, \tag{16.4}$$

and a rational B-spline surface is written as

$$\mathbf{s}(u, v) = \frac{\sum_i \sum_j w_{i,j} \mathbf{d}_{i,j} N_i^m(u) N_j^n(v)}{\sum_i \sum_j w_{i,j} N_i^m(u) N_j^n(v)}. \tag{16.5}$$

Figure 16.14 shows an example of a rational B-spline surface. It was obtained from the same control net as the surface in Figure 16.4, but with weights as shown in the figure. Note how the "dip" became more pronounced, as well as the "vertical ridge."

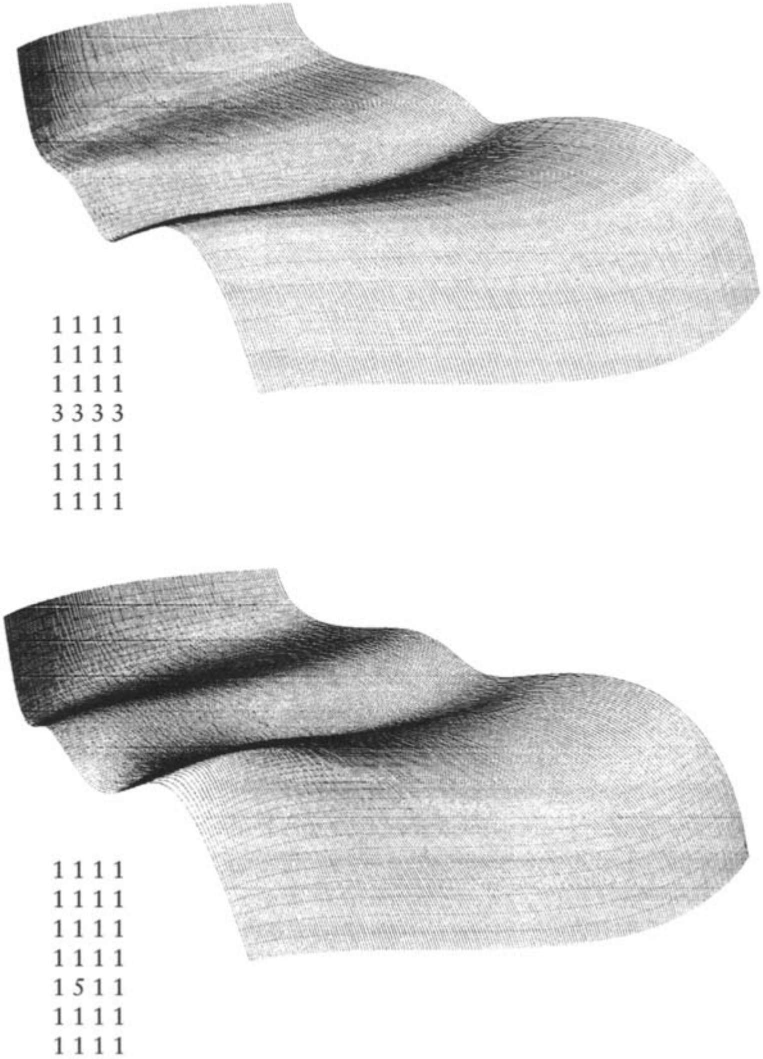


Figure 16.14 Rational B-spline surfaces: a surface together with the set of weights used to generate it.

Rational surfaces are obtained as the projections of tensor product patches, but they are not tensor product patches themselves. Recall that a tensor product surface is of the form $\mathbf{x}(u, v) = \sum_i \sum_j \mathbf{c}_{i,j} F_{i,j}(u, v)$, where the basis functions $F_{i,j}$ may be expressed as products $F_{i,j}(u, v) = A_i(u) B_j(v)$. The basis functions for (16.5) are of the form

$$F_{i,j}(u,v) = \frac{w_{i,j}N_i^m(u)N_j^n(v)}{\sum_i \sum_j w_{i,j}N_i^m(u)N_j^n(v)}.$$

Because of the structure of the denominator, this may in general not be factored into the required form $F_{i,j}(u,v) = A_i(u)B_j(v)$.

But even though rational surfaces do not possess a tensor product structure, we may use many tensor product algorithms for their manipulation. Consider, for example, the problem of finding the piecewise rational bicubic Bézier form of a rational bicubic B-spline surface. All we have to do is to convert each row of the B-spline control net into piecewise rational Bézier cubics (according to Section 13.7). Then we repeat this process for each column of the resulting net (and the resulting weights!), simply following the principle outlined in Figure 16.5.

As another example, consider the problem of extracting an isoparametric curve from a rational Bézier surface. Suppose the curve corresponds to $v = \hat{v}$. We simply interpret all columns of the control net as control polygons and evaluate each at $\hat{v}$, using the rational de Casteljau algorithm, for example. Keep in mind that we also have to compute a weight for each control polygon. We can now interpret all obtained points together with their weights as the Bézier control polygon of the desired isoparametric curve. In general, its end weights will not be unity, that is, the curve will not be in standard form (as described in Section 13.5). This situation may be remedied by the use of the reparametrization algorithm, which is also described in that section.

16.7 Surfaces of Revolution

Currently, rational B-spline surfaces are used for two reasons: they allow the exact representation of surfaces of revolution and of quadric surfaces. We will briefly describe surfaces of revolution in rational B-spline form here; quadric surfaces will be treated in Section 18.2.

A surface of revolution is given by

$$\mathbf{x}(u,v) = \begin{bmatrix} r(v)\cos u \\ r(v)\sin u \\ z(v) \end{bmatrix}.$$

For fixed v, an isoparametric line $v = const$ traces out a circle of radius $r(v)$, called a *meridian*. Since a circle may be exactly represented by rational quadratic arcs, we may find an exact rational representation of a surface of revolution provided we can represent $r(v), z(v)$ in rational form.

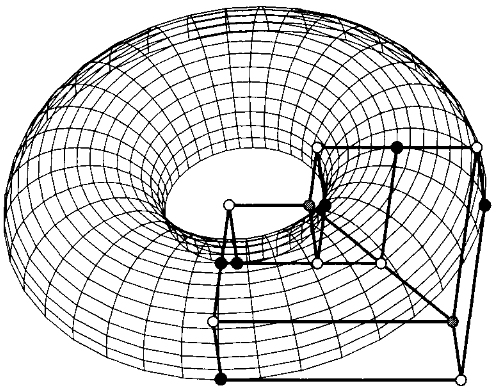


Figure 16.15 Surfaces of revolution: the surface is formed by eight rational biquadratic patches. The solid control points (shown for two patches only) have weight 1; the open points have weight 0.71, and the two gray points have weight 0.5.

The most convenient way to define a surface of revolution is to prescribe the (planar) generating curve, or generatrix, given by

$$\mathbf{g}(v) = [r(v), 0, z(v)]^{\mathrm{T}}$$

and by the axis of revolution, in the same plane as **g**. Suppose **g** is given by its control polygon, knot sequence, and weight sequence. We can construct a surface of revolution such that each meridian consists of three rational quadratic arcs, as shown in Figure 12.11. For each vertex of the generating polygon, construct an equilateral triangle (perpendicular to the axis of revolution) as in Figure 12.11. Assign the given weights of the generatrix to the three polygons corresponding to the triangle edge midpoints; assign half those weights to the three control polygons corresponding to the triangle vertices. In this way, we represent *exactly* “classical” surfaces such as cylinders, spheres, or tori.

Instead of breaking down each meridian into three arcs, we might have used four. The resulting four biquadratic control nets then form three concentric squares in the projection into the $z = 0$ plane. The control points at the squares’ midpoints are copies of the generatrix control points; their weights are those of the generatrix. The remaining weights, corresponding to the squares’ corners, are multiplied by $\cos(45°) = \sqrt{2}/2$. Figure 16.15 gives an example of a semicircle that sweeps out a surface of revolution.

Note that although the generatrix may be defined over a knot sequence $\{v_j\}$ with only simple knots, this is not possible for the knots of the meridian circles; we have to use double knots, thereby essentially reducing it to the piecewise Bézier form.

16.8 Volume Deformations

Sometimes local control of a surface, nice as it may be, is not what is needed. A typical design request is "stretch this surface in that direction," or "bend that surface like so." These are *global* shape deformations, and the usual tweaking of control polygon vertices is somewhat cumbersome for this task. P. Bézier devised a method to deform a Bézier patch in a manner that would satisfy this global deformation principle. We shall see that it is also applicable to B-spline surfaces. For literature, see Bézier [59], [62], [63]. A more graphics-oriented version of this principle was presented by Sederberg and Parry [558], see also [268].

To illustrate the principle, let us consider the 2D case first. Let $\mathbf{x}(t)$ be a planar curve (Bézier, B-spline, rational B-spline, and so forth), which is, without loss of generality, located within the (u, v) unit square. Next, let us cover the square with a regular grid of points $\mathbf{b}_{i,j} = [i/m, j/n]^{\mathrm{T}}; i = 0, \ldots, m; j = 0, \ldots, n$. We can now write every point (u, v) as

$$(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{b}_{i,j} B_i^m(u) B_j^n(v);$$

this follows from the linear precision property of Bernstein polynomials (5.14).

If we now distort the grid of $\mathbf{b}_{i,j}$ into a grid $\hat{\mathbf{b}}_{i,j}$, the point (u, v) will be mapped to a point $(\hat{u}, \hat{v})$:

$$(\hat{u}, \hat{v}) = \sum_{i=0}^{m} \sum_{j=0}^{n} \hat{\mathbf{b}}_{i,j} B_i^m(u) B_j^n(v). \tag{16.6}$$

In other words, we are dealing with a mapping of $\mathbb{E}^2$ to $\mathbb{E}^2$.

In particular, the control vertices of the curve $\mathbf{x}(t)$ will be mapped to new control vertices, which in turn determine a new curve $\mathbf{y}(t)$. Note that $\mathbf{y}$ is only an approximation to the image of $\mathbf{x}$ under (16.6).[4] This is highlighted by the fact that the image of $\mathbf{x}$'s control polygon under (16.6) would be a collection of curve arcs, not another piecewise linear polygon.

We now have an *indirect* method for curve design: changing the $\mathbf{b}_{i,j}$ will produce globally deformed curves. This technique may facilitate certain design tasks that are otherwise tedious to perform. Figure 16.16 gives an example of the use of this global design technique.

[4] An exact procedure is described by T. DeRose [159].

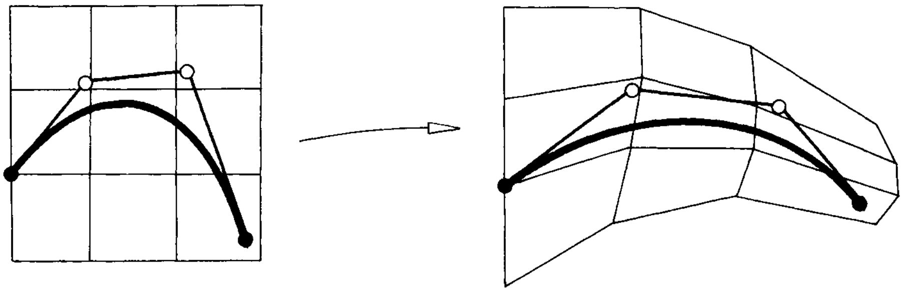


Figure 16.16 Global curve distortions: a Bézier polygon is distorted into another polygon, resulting in a deformation of the initial curve.

This technique may be generalized. For instance, we may replace the Bézier distortion (16.6) by an analogous tensor product B-spline distortion. This would reintroduce some form of local control into our design scheme.

The next level of generalization is to $\mathbb{E}^3$: we introduce a *trivariate* Bézier patch by

$$(\hat{u}, \hat{v}, \hat{w}) = \sum_{i=0}^{m} \sum_{j=0}^{n} \sum_{k=0}^{l} \hat{\mathbf{b}}_{i,j,k} B_i^m(u) B_j^n(v) B_k^l(w), \qquad (16.7)$$

which constitutes a deformation of 3D space $\mathbb{E}^3$. We may use (16.7) to deform the control net of a surface embedded in the unit cube. Again, the use of a Bézier patch for the distortion is immaterial; we might have used trivariate B-splines, and so on in order to introduce some degree of locality into the method.

An example is shown in Figure 16.17. Part of the mushroom-shaped surface is embedded in a trivariate Bézier volume that is cubic in the vertical direction and linear in the other two. The top layer of control points is moved upward, leading to a C^2 distortion of the initial object.

Why use deformation methods instead of just manipulating control vertices interactively? Volume deformation methods allow a designer to modify whole assemblies of surfaces at once, in a way that spreads out the changes in each part of the assembly in a very harmonic way. By tweaking control vertices one by one, a similarly balanced modification cannot be the result.

A practical example of volume deformations is in brain imaging. In comparative studies, many MRI brain scans have to be compared. Different people have differently shaped brains; in order to carry out a meaningful comparison, they have to be aligned (see Section 2.3); then they have to be deformed for a closer match—see [620] or [592].

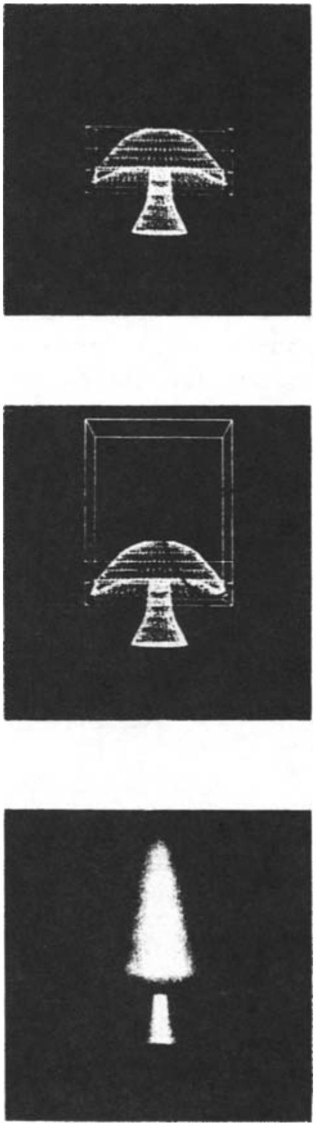


Figure 16.17 Global surface distortions: part of a surface is embedded in a Bézier volume (top). That volume is distorted (middle), leading to a distorted final object (bottom).

16.9 CONS and Trimmed Surfaces

If we create any parametric curve $(u(t), v(t))$ in the domain of a surface $\mathbf{x}(u, v)$, it will be mapped to a curve $\mathbf{x}(u(t), v(t))$ on the surface, or CONS. If the domain curve is itself a Bézier curve of degree p, then the CONS will be of degree $(m + n)p$, assuming m and n are the parametric degrees of $\mathbf{x}(u, v)$. Such curves were first considered by Bézier, see [57], [60], where they were called *transposants*.

In most practical applications, the curve in the domain is expressed as a piecewise linear curve, and the resulting CONS is approximated as being piecewise linear. If the piecewise linear CONS is dense enough, this should not cause problems. CONS can arise in many applications: if we intersect two surfaces, the resulting intersection curve is a CONS on either of the two surfaces. Or we could project a space curve onto a surface, again resulting in a CONS.

If the domain curve of a CONS is *closed*, then it divides the domain into two parts: those inside the curve and those outside. In the same way, the closed CONS divides the surface into two parts. If we want to know, for an arbitrary point (u, v) in the domain, if it lies inside the domain curve, take an arbitrary ray emanating from (u, v). Then count the number of its intersections with the domain curve. If it is even, (u, v) is outside, and inside otherwise; see Figure 16.18 for an illustration. For programming purposes, there are no "arbitrary" rays. Rays parallel to the u- or v-direction will typically suffice.

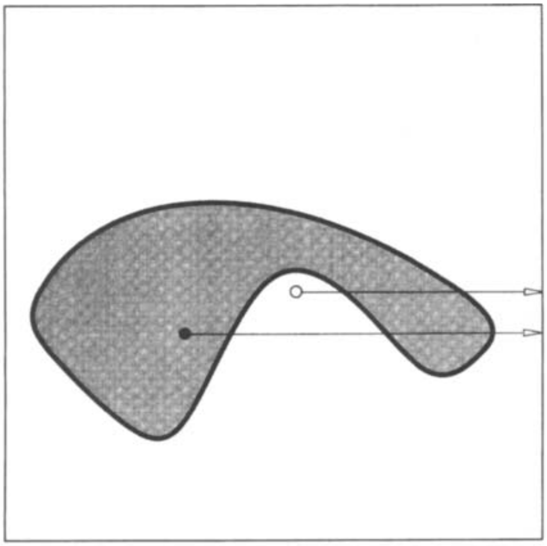


Figure 16.18 Inside/outside test: a ray from the solid point intersects the domain curve three times; it is inside. The open point is inside. The inside region is shown shaded.

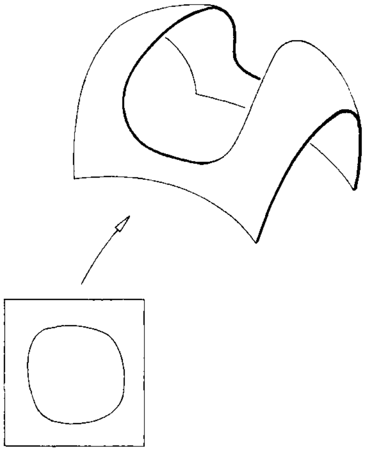


Figure 16.19 Trimmed surfaces: certain parts of a tensor product surface are marked as "invalid" by a pair of CONS.

CONS are mainly used for a modification of tensor product surfaces by a technique known as trimming. A trimmed surface has certain areas of it marked as invalid or invisible by a set of closed CONS. Figure 16.19 gives an example. There, two CONS are employed: one corresponds to a closed curve in the domain; the other one is the perimeter of the domain. The inside/outside test works just as it does for only one CONS.

Another example for trimmed surfaces is given in Color Plate III. Toward the lower-right quadrant of that figure, we see a small "patch" surface that blends the central part of the hood to the part over the fender. (Such surfaces, by the way, are extremely tedious to design.) If you take a close look at Color Plate III, you will see that the surfaces covered by the patch surface are not drawn where the patch surface is drawn. In fact, they are not defined there. The parts occupied by the patch surface are not part of the "regular" surfaces—they are "trimmed away."

Trimmed surfaces should be viewed as an "engineering" extension of tensor product patches. That is to say, they are not a panacea to all surface problems either. Consider, for example, the problem of joining two trimmed surfaces in a smooth way. If they are to join along trim curves, there is no known method to ensure exact tangent plane continuity between them, as was the case for standard tensor patches. Such smoothness questions must be dealt with on a case-by-case basis, which is clearly not very desirable. Just consider the problem of fitting the aforementioned blend surface from Color Plate III between its neighbors!

Literature on trimmed surfaces: Farouki and Hinds [223], Shantz and Chang [571], Casale and Bobrow [102], Miller [427], Lasser and Bonneau [373], Brunnett [95], Vigo and Brunet [602].

16.10 Implementation

The routines in this section are written for rational surfaces. By setting all weights equal to one, the standard piecewise polynomial case is recovered.

The routine that converts a rational bicubic B-spline control net into the piecewise bicubic Bézier form:

```
 void ratbspl_to_bez_surf(bspl_x,bspl_y,bspl_w,lu,lv,knot_u,
                          knot_v,bez_x,bez_y,bez_w,aux_x,aux_y,aux_w)
/*    Converts B-spline control net into piecewise
      Bezier control net (bicubic).
Input:      bspl_x,bspl_y:     B-spline control net (one coordinate only)
            bspl_w:            B-spline weights
            lu,lv:             no. of intervals in u- and v-direction
            knot_u, knot_v:    knot vectors in u- and v-direction
Output:     bez_x,bez_y:       piecewise bicubic Bezier net.
            bez_w:             Bezier weights.
Work space:aux_x,aux_y,aux_w: needed to store intermediate results.

Remark:     The  piecewise Bezier net only stores each control point once,
            i.e., neighboring patches share the same boundary.
            Knots are simple (but, in the language of Chapter 10, the
            boundary knots have multiplicity three).
*/
```

Once the piecewise rational Bézier representation of a bicubic spline surface is achieved, the following routine plots the whole surface:

```
void plot_ratbez_surfaces(bez_x,bez_y,bez_w,lu,lv,u_points,v_points,
                          scale_x,scale_y,value)
/*      Plots piecewise cubic surface, i.e., generates postscript output
Input:  bez_x, bez_y:     control nets
        lu,lv:            no. of segments in u- and v- direction
        u_points,v_points: per patch: v_points many
                          isoparametric curves with u_points
                          points on each
        value:            minmax box of all control nets.
        scale_x,scale_y:  scale factors for  postscript
*/
```

Tensor product spline interpolation (bicubic) is carried out by the following routine. It uses Bessel end conditions.

```
void spline_surf_int(data_x,data_y,bspl_x,bspl_y,lu,lv,knot_u,
                     knot_v,aux_x,aux_y)
/*      Interpolates to an array of size [0,lu+2]x[0,lv+2]
Input:     data_x, data_y:   data array (one coordinate only)
           lu,lv:            no. of intervals in u- and v-direction
           knot_u, knot_v:   knot vectors in u- and v-direction
Output:    bspl_x,bspl_y:    B-spline control net.

Work space: aux_x, aux_y.

Remark:    On input, it is assumed that data_x and data_y have rows
           1 and lu+1 and columns 1 and lv+1 empty, i.e., they are
           not filled with data points. Example for lu=4, lv=7:

          x0xxxxxx0x
          0000000000
          x0xxxxxx0x                x=data coordinate,
          x0xxxxxx0x                0=unused  input array
          x0xxxxxx0x                  element.
          0000000000                The 0's will be filled with
          x0xxxxxx0x                'tangent Bezier points'.

     This approach makes it easy to feed in clamped end conditions
     if so desired: put in values in the 0's and delete the calls
     to bessel_ends below.
*/
```

Next is the header of a program that plots the control net of a Bézier surface or of a composite surface.

```
void psplot_net(lu,lv,bx,by,step_u,step_v,scale_x,scale_y,value)
/*      plots  control net      into postscript-file.
Input:  lu,lv:          dimensions of net
        bx,by:          net vertices
        step_u,step_v:  subnet sizes (e.g. both=3 for pw bicubic net)
        scale_x,scale_y:scale factors for ps
        value:          window size in world coords
Output:                 written into postscript file
*/
```

16.11 Problems

1 Generalize the B-spline knot insertion algorithm to the tensor product case.

***2** Show that if two polynomial surfaces are C^1 across a common boundary, then they are also twist continuous across that boundary.

***3** Suppose we want to find a parametrization $\{u_i\}$ for a tensor product interpolant. We may parametrize all rows of data points and then form the averages of the parametrizations thus obtained. Or we could average all rows of data points, for example, by setting $\mathbf{p}_i = \sum_j \frac{j}{n}\mathbf{x}_{i,j}$ and we could then parametrize the $\mathbf{p}_i$. Do we get the same result? Discuss both methods.

P1 Embed your Bézier surface from Problem P3 of Chapter 14 in a tricubic grid, similar to Figure 16.17. Then "stretch" your surface, leaving the front part unchanged.

P2 Model a Klein bottle as a closed bicubic B-spline surface. Literature: [323], [170]. If you have the graphics capabilities, display your result as a translucent surface.

P3 Generate an array of points on a sphere. For latitudes, take $\phi_i = 0, 10, 20, \ldots, 90$ degrees. For longitudes, take $\psi_i = 45, 50, 55, \ldots, 75$ degrees. Pass several tensor product interpolants through the data and compare their deviations from the true sphere. For the bicubic C^2 spline interpolant, also compare uniform and chord length parametrizations.

Bézier Triangles 17

When de Casteljau invented Bézier curves in 1959, he realized the need for the extension of the curve ideas to surfaces. Interestingly enough, the first surface type that he considered was what we now call Bézier triangles. This historical "first" of triangular patches is reflected by the mathematical statement that they are a more "natural" generalization of Bézier curves than are tensor product patches. We should note that although de Casteljau's work was never published, Bézier's was; therefore, the corresponding field now bears Bézier's name. For the placement of triangular Bernstein–Bézier surfaces in the field of CAGD, see Barnhill [24].

Though de Casteljau's work (established in two internal Citroën technical reports [145] and [146]) remained unknown until its discovery by W. Boehm around 1975, other researchers realized the need for triangular patches. M. Sabin [516] worked on triangular patches in terms of Bernstein polynomials, unaware of de Casteljau's work. Among the people concerned with the development of triangular patches we name L. Frederickson [249], P. Sablonnière [522], and D. Stancu [580]. All of their Bézier-type approaches relied on the fact that piecewise surfaces were defined over regular triangulations; arbitrary triangulations were considered by Farin [189]. Two surveys on the field of triangular Bézier patches are Farin [197] and de Boor [139].

17.1 The de Casteljau Algorithm

The de Casteljau algorithm for triangular patches is a direct generalization of the corresponding algorithm for curves. The curve algorithm uses repeated linear interpolation, and that process is also the key ingredient in the triangle case. The

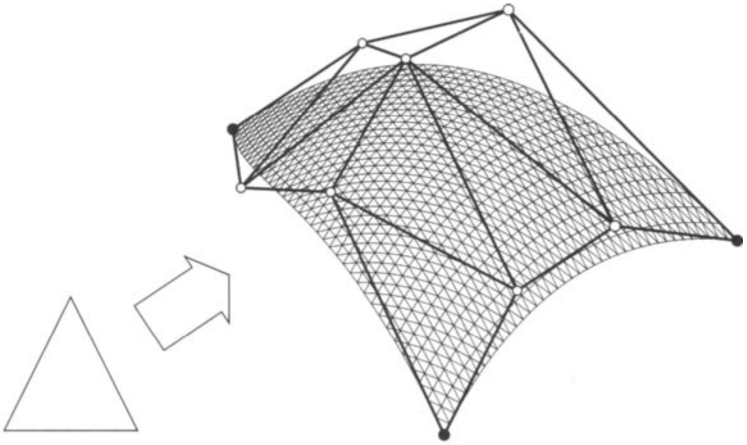


Figure 17.1 Bézier triangles: a cubic patch with its control net.

"triangular" de Casteljau algorithm is completely analogous to the univariate one, the main difference being notation. The control net is now of a triangular structure; in the quartic case, the control net consists of vertices

$$\begin{matrix} \mathbf{b}_{040} \\ \mathbf{b}_{031}\mathbf{b}_{130} \\ \mathbf{b}_{022}\mathbf{b}_{121}\mathbf{b}_{220} \\ \mathbf{b}_{013}\mathbf{b}_{112}\mathbf{b}_{211}\mathbf{b}_{310} \\ \mathbf{b}_{004}\mathbf{b}_{103}\mathbf{b}_{202}\mathbf{b}_{301}\mathbf{b}_{400} \end{matrix}$$

Note that all subscripts sum to 4. In general, the control net consists of $\frac{1}{2}(n+1)(n+2)$ vertices. The numbers $\frac{1}{2}(n+1)(n+2)$ are called *triangle numbers*. Figure 17.1 gives an example of a cubic patch with its control net.

Some notation: we denote the point $\mathbf{b}_{ijk}$ by $\mathbf{b}_{\mathbf{i}}$. Moreover, we use the abbreviations $\mathbf{e1} = (1,0,0)$, $\mathbf{e2} = (0,1,0)$, $\mathbf{e3} = (0,0,1)$, and $|\mathbf{i}| = i + j + k$. When we say $|\mathbf{i}| = n$, we mean $i + j + k = n$, always assuming $i, j, k \geq 0$.

The de Casteljau algorithm follows.

de Casteljau algorithm

Given: A triangular array of points $\mathbf{b}_{\mathbf{i}} \in \mathbb{E}^3$; $|\mathbf{i}| = n$ and a point in $\mathbb{E}^2$ with barycentric coordinates $\mathbf{u}$.

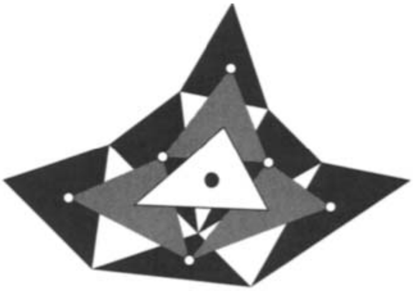


Figure 17.2 The "triangular" de Casteljau algorithm: a point is constructed by repeated linear interpolation.

Set:

$$\mathbf{b}_{\mathrm{i}}^{r}(\mathbf{u}) = u\mathbf{b}_{\mathrm{i+e1}}^{r-1}(\mathbf{u}) + v\mathbf{b}_{\mathrm{i+e2}}^{r-1}(\mathbf{u}) + w\mathbf{b}_{\mathrm{i+e3}}^{r-1}(\mathbf{u}), \qquad (17.1)$$

where

$$r = 1, \ldots, n \quad \text{and} \quad |\mathbf{i}| = n - r$$

and $\mathbf{b}_{\mathrm{i}}^{0}(\mathbf{u}) = \mathbf{b}_{\mathrm{i}}$. Then $\mathbf{b}_{0}^{n}(\mathbf{u})$ is the point with parameter value **u** on the *Bézier triangle*[1] $\mathbf{b}^{n}$.

Figure 17.2 illustrates the construction of a point on a cubic Bézier triangle. We give a simple example: for $n = 3, r = 1$, and $\mathbf{i} = (2, 0, 0)$, we would obtain $\mathbf{b}_{200}^{1} = u\mathbf{b}_{300} + v\mathbf{b}_{210} + w\mathbf{b}_{201}$. A complete numerical example is given in Example 17.1.

Based on the de Casteljau algorithm, we can state many properties of Bézier triangles:

Affine invariance: This property follows since linear interpolation is an affine map and since the de Casteljau algorithm makes use of linear interpolation only.

Invariance under affine parameter transformations: This property is guaranteed since such a reparametrization amounts to choosing a new domain triangle, but we have not even specified any particular domain triangle. More precisely, a point **u** will have the same barycentric coordinates **u** after an affine transformation of the domain triangle.

[1] More precisely, a triangular Bézier patch.

Example 17.1 **Computing a point with the de Casteljau algorithm.**

Let the coefficients $\mathbf{b}_\mathbf{i}$ of a quadratic patch be given by

$$\begin{matrix} & & \begin{bmatrix}0\\6\\0\end{bmatrix} & & \\ & \begin{bmatrix}0\\3\\0\end{bmatrix} & & \begin{bmatrix}3\\3\\6\end{bmatrix} & \\ \begin{bmatrix}0\\0\\0\end{bmatrix} & & \begin{bmatrix}3\\0\\0\end{bmatrix} & & \begin{bmatrix}6\\0\\9\end{bmatrix}, \end{matrix}$$

and let $\mathbf{u} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Further, we make the assumption that $\mathbf{b}_{300} = [6, 0, 9]^\mathrm{T}$ is the image of **e1**, and $\mathbf{b}_{030} = [0, 6, 0]^\mathrm{T}$ is the image of **e2**.

The de Casteljau steps are as follows: for $r = 1$, the $\mathbf{b}_\mathbf{i}^1$ are given by

$$\begin{matrix} & \begin{bmatrix}1\\4\\2\end{bmatrix} & \\ \begin{bmatrix}1\\1\\0\end{bmatrix} & & \begin{bmatrix}4\\1\\5\end{bmatrix} \end{matrix}$$

The result $\mathbf{b}_0^2$ is

$$\begin{bmatrix}2\\2\\7/3\end{bmatrix}.$$

Convex hull property: Guaranteed since for $0 \le u, v, w \le 1$, each $\mathbf{b}_\mathbf{i}^r$ is a convex combination of the previous $\mathbf{b}_\mathbf{i}^{r-1}$.

Boundary curves: For a triangular patch, these curves are determined by the boundary control vertices. For example, a point on the boundary curve $\mathbf{b}^n(u, 0, w)$ is generated by

$$\mathbf{b}_\mathbf{i}^r(u, 0, w) = u\mathbf{b}_{\mathbf{i}+\mathbf{e1}}^{r-1} + w\mathbf{b}_{\mathbf{i}+\mathbf{e3}}^{r-1}; \quad u + w = 1,$$

which is the univariate de Casteljau algorithm for Bézier curves.

17.2 Triangular Blossoms

The blossoming principle was introduced in Section 4.4 and also proves useful here. Our development follows a familiar flavor: we feed different arguments into the de Casteljau algorithm. At level r of the algorithm, we will use $\mathbf{u}_r$ as its argument, arriving finally at level n, with the blossom value $\mathbf{b}[\mathbf{u}_1, \ldots, \mathbf{u}_n]$. Note that all arguments are triples of numbers, because they represent points in the domain plane. The multivariate polynomial $\mathbf{b}[\mathbf{u}_1, \ldots, \mathbf{u}_n]$ is called the *blossom* of the triangular patch $\mathbf{b}(\mathbf{u})$. This blossom has all the properties that we encountered earlier: it agrees with the patch if all arguments are equal: $\mathbf{b}(\mathbf{u}) = \mathbf{b}[\mathbf{u}^{<n>}]$ (recall that $\mathbf{u}^{<n>}$ is short for n-fold repetition of $\mathbf{u}$), it is multiaffine, and it is symmetric.

The last property is perhaps the least obvious one. We derived the symmetry property of curve blossoms as a consequence of Menelaos' theorem; see Section 3.3. A similar theorem holds when dealing with triangular blossoms: let $\mathbf{b}_\mathbf{i}$; $|\mathbf{i}| = n = 2$ be an array of six control points, and let $\mathbf{u}$ and $\mathbf{v}$ be (the barycentric coordinates of) two points. Then $\mathbf{b}[\mathbf{u}, \mathbf{v}] = \mathbf{b}[\mathbf{v}, \mathbf{u}]$.

This is seen by verifying that both expressions equal

$$\mathbf{b}_{002}u_3v_3 + \mathbf{b}_{020}u_2v_2 + \mathbf{b}_{200}u_1v_1 + \mathbf{b}_{011}(u_3v_2 + u_2v_3) \\ + \mathbf{b}_{101}(u_1v_3 + u_3v_1) + \mathbf{b}_{110}(u_1v_2 + u_2v_1).$$

Let us consider a special case, namely, that of fixing one argument and letting the remaining ones be equal, similar to the developments of polars in Section 5.6. So consider $\mathbf{b}[\mathbf{e1}, \mathbf{u}^{<n-1>}]$. We have to carry out one de Casteljau step with respect to $\mathbf{e1}$, and then continue as in the standard algorithm. Since a step with respect to $\mathbf{e1}$ yields

$$\mathbf{b}_\mathbf{i}^1(\mathbf{e1}) = \mathbf{b}_{i+1,j,k}; \quad |\mathbf{i}| = n - 1,$$

we end up with a triangular patch of degree $n - 1$ whose vertices are the original vertices with the exception of the $\mathbf{b}_{0,j,k}$—that row of control points is "peeled off."

We may continue this experiment: if we next use $\mathbf{e2}$, we peel off another layer of coefficients, and so on. Let us use $\mathbf{e1}$ i times, $\mathbf{e2}$ j times, and $\mathbf{e3}$ k times. We are then left with a single control point:

$$\mathbf{b}_\mathbf{i} = \mathbf{b}[\mathbf{e1}^{<i>}, \mathbf{e2}^{<j>}, \mathbf{e3}^{<k>}] \quad |\mathbf{i}| = n. \tag{17.2}$$

So again the Bézier control points are obtainable as special blossom values!

We may also write the intermediate points of the de Casteljau algorithm as special blossom values:

$$\mathbf{b}_\mathbf{i}^r(\mathbf{u}) = \mathbf{b}[\mathbf{u}^{<r>}, \mathbf{e1}^{<i>}, \mathbf{e2}^{<j>}, \mathbf{e3}^{<k>}]; \quad i + j + k + r = n. \tag{17.3}$$

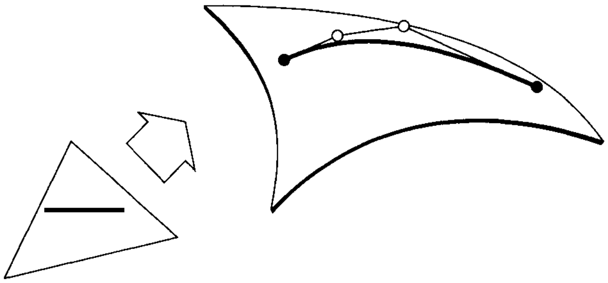


Figure 17.3 A curve on a surface: a line segment in the domain is mapped to a curve on a triangular patch.

If we are interested in the control vertices $\mathbf{c_i}$ with respect to a triangle with vertices $\mathbf{f1}, \mathbf{f2}, \mathbf{f3}$, all we have to do is to evaluate the blossom:

$$\mathbf{c_i} = \mathbf{b}[\mathbf{f1}^{<i>}, \mathbf{f2}^{<j>}, \mathbf{f3}^{<k>}]. \tag{17.4}$$

This relationship, without use of the blossoming principle, is discussed by Goldman [261] and by Boehm and Farin [81]. See also Seidel [564].

The triangular analogue to the Leibniz formula (3.22) is as follows:

$$\mathbf{b}[(\alpha\mathbf{r} + \beta\mathbf{s})^{<n>}] = \sum_{i=0}^{n} \binom{n}{i} \alpha^i \beta^{n-i} \mathbf{b}[\mathbf{r}^{<i>}, \mathbf{s}^{<n-i>}]. \tag{17.5}$$

There is an immediate practical ramification of this formula: suppose we are given two points $\mathbf{r}$ and $\mathbf{s}$ in the domain. The straight line through them is mapped to a curve of degree n on the triangular patch. This curve may be written as a Bézier curve, and its Bézier points are given by $\mathbf{b}[\mathbf{r}^{<n>}], \mathbf{b}[\mathbf{r}^{<n-1>}, \mathbf{s}], \ldots, \mathbf{b}[\mathbf{s}^{<n>}]$. See Figure 17.3.

If we use three points $\mathbf{r}, \mathbf{s}, \mathbf{t}$ in the domain, we obtain

$$\mathbf{b}[(\alpha\mathbf{r} + \beta\mathbf{s} + \gamma\mathbf{t})^{<n>}] = \sum_{|\mathbf{i}|=n} \binom{n}{\mathbf{i}} \alpha^i \beta^j \gamma^k \mathbf{b}[\mathbf{r}^{<i>}, \mathbf{s}^{<j>}, \mathbf{t}^{<k>}]. \tag{17.6}$$

Here, $\binom{n}{\mathbf{i}}$ is the generalized binomial coefficient and is defined as

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

and the summation $\sum_{|\mathbf{i}|=n}$ is meant to sum over all triples (i, j, k) with $i, j, k \geq 0$ and $|\mathbf{i}| = i + j + k = n$.

17.3 Bernstein Polynomials

Univariate Bernstein polynomials are the terms of the binomial expansion of $[t + (1 - t)]^n$. In the bivariate case, Bernstein polynomials $B_{\mathbf{i}}^n$ are defined by[2]

$$B_{\mathbf{i}}^n(\mathbf{u}) = \binom{n}{\mathbf{i}} u^i v^j w^k; \quad |\mathbf{i}| = n. \tag{17.7}$$

We define $B_{\mathbf{i}}^n(\mathbf{u}) = 0$ if $i, j, k < 0$ or $i, j, k > n$. This follows standard convention for the *trinomial coefficients* $\binom{n}{\mathbf{i}}$. Some of the cubic Bernstein polynomials are shown in Figure 17.4.

As a further example, the quartic Bernstein polynomials may be arranged in the following triangular scheme (corresponding to the control point arrangement in the de Casteljau algorithm):

$$\begin{matrix} & & & & v^4 & & & & \\ & & & 4v^3w & & 4uv^3 & & & \\ & & 6v^2w^2 & & 12uv^2w & & 6u^2v^2 & & \\ & 4vw^3 & & 12uvw^2 & & 12u^2vw & & 4u^3v & \\ w^4 & & 4uw^3 & & 6u^2w^2 & & 4u^3w & & u^4 \end{matrix}$$

Bernstein polynomials satisfy the following recursion:

$$B_{\mathbf{i}}^n(\mathbf{u}) = uB_{\mathbf{i}-\mathbf{e1}}^{n-1}(\mathbf{u}) + vB_{\mathbf{i}-\mathbf{e2}}^{n-1}(\mathbf{u}) + wB_{\mathbf{i}-\mathbf{e3}}^{n-1}(\mathbf{u}); \quad |\mathbf{i}| = n. \tag{17.8}$$

This follows from their definition, as given in (17.7), and the use of the identity

$$\binom{n}{\mathbf{i}} = \binom{n-1}{\mathbf{i}-\mathbf{e1}} + \binom{n-1}{\mathbf{i}-\mathbf{e2}} + \binom{n-1}{\mathbf{i}-\mathbf{e3}}.$$

Just as in the curve case, Bernstein polynomials may be used to define a Bézier triangular patch:

$$\mathbf{b}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u}). \tag{17.9}$$

The proof is a straightforward application of (17.6) and the observation that $\mathbf{u} = u\mathbf{e1} + v\mathbf{e2} + w\mathbf{e3}$.

[2] Keep in mind that although $B_{\mathbf{i}}^n(u, v, w)$ *looks* trivariate, it is not, since $u + v + w = 1$.

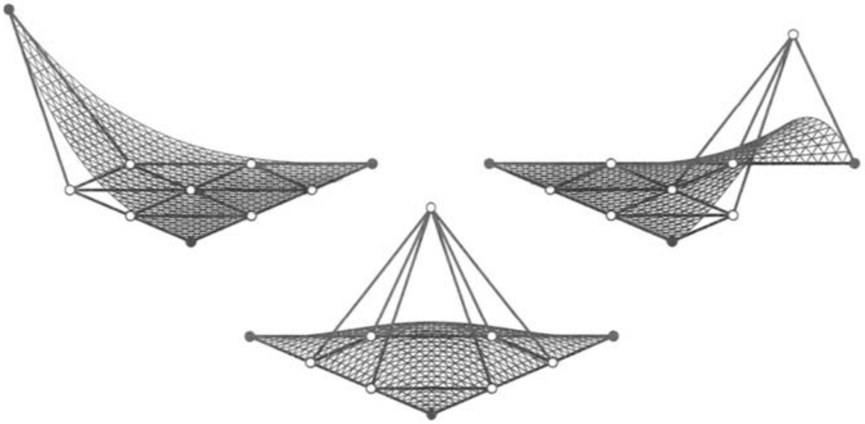


Figure 17.4 Bernstein polynomials: the basis functions B^3_{003}, B^3_{120} (top row) and B^3_{111} (bottom).

The intermediate points $\mathbf{b}^r_{\mathrm{i}}$ in the de Casteljau algorithm may also be expressed in terms of Bernstein polynomials:

$$\mathbf{b}^r_{\mathrm{i}}(\mathbf{u}) = \sum_{|\mathrm{j}|=r} \mathbf{b}_{\mathrm{i}+\mathrm{j}} B^r_{\mathrm{j}}(\mathbf{u}); \qquad |\mathrm{i}| = n - r. \tag{17.10}$$

We can generalize (17.9) just as we could in the univariate case:

$$\mathbf{b}^n(\mathbf{u}) = \sum_{|\mathrm{j}|=n-r} \mathbf{b}^r_{\mathrm{j}}(\mathbf{u}) B^{n-r}_{\mathrm{j}}(\mathbf{u}); \qquad 0 \le r \le n. \tag{17.11}$$

17.4 Derivatives

When we discussed derivatives for tensor product patches (Section 14.6), we considered *partials* because they are easily computed for those surfaces. The situation is different for triangular patches; the appropriate derivative here is the *directional derivative*. Let $\mathbf{u}_1$ and $\mathbf{u}_2$ be two points in the domain. Their difference $\mathbf{d} = \mathbf{u}_2 - \mathbf{u}_1$ defines a vector.[3] The directional derivative of a surface at $\mathbf{x}(\mathbf{u})$ with respect to $\mathbf{d}$ is given by

$$D_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = \lim_{\mathbf{d}\to 0} \frac{1}{|\mathbf{d}|}[\mathbf{x}(\mathbf{u} + \mathbf{d}) - \mathbf{x}(\mathbf{u})].$$

[3] In barycentric coordinates, a point $\mathbf{u}$ is characterized by $u + v + w = 1$, while a vector $\mathrm{d} = (d, e, f)$ is characterized by $d + e + f = 0$.

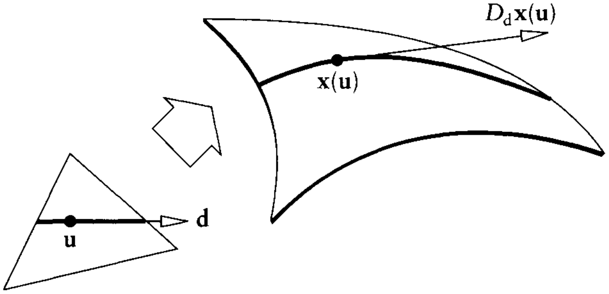


Figure 17.5 Directional derivatives: a straight line in the domain is mapped to a curve on the patch.

A geometric interpretation: in the domain, draw the straight line through **u** that is parallel to **d**. This straight line will be mapped to a curve on the patch. Its tangent vector at **x**(**u**) is the desired directional derivative; it is shown in Figure 17.5.

Using the proof technique from Section 5.3, we obtain

$$D_{\mathrm{d}}\mathbf{b}(\mathbf{u}) = n\mathbf{b}[\mathbf{d}, \mathbf{u}^{<n-1>}]. \tag{17.12}$$

Equation (17.12) has two possible interpretations, just as in the curve case. We may perform one step of the de Casteljau algorithm with respect to the direction vector **d**, and $n-1$ more with respect to the position **u**:

$$D_{\mathrm{d}}\mathbf{b}(\mathbf{u}) = n \sum_{|\mathbf{i}|=n-1} \mathbf{b}_{\mathbf{i}}^{1}(\mathbf{d})B_{\mathbf{i}}^{n-1}(\mathbf{u}). \tag{17.13}$$

Alternatively, we may first carry out $n-1$ de Casteljau steps and then follow up with one step with respect to **d**:

$$D_{\mathrm{d}}\mathbf{b}(\mathbf{u}) = n(d\mathbf{b}_{\mathrm{e1}}^{n-1} + e\mathbf{b}_{\mathrm{e2}}^{n-1} + f\mathbf{b}_{\mathrm{e3}}^{n-1}). \tag{17.14}$$

We may continue taking derivatives, arriving at:

$$D_{\mathrm{d}}^{r}\mathbf{b}(\mathbf{u}) = \frac{n!}{(n-r)!}\mathbf{b}[\mathbf{d}^{<r>}, \mathbf{u}^{<n-r>}]. \tag{17.15}$$

The rth directional derivative at **b**(**u**) is therefore found by performing r steps of the de Casteljau algorithm with respect to **d**, and then by performing $n-r$ more steps with respect to **u**. Of course, it is irrelevant in which order we take these steps (first noted in [188]).

In the same way, we may compute mixed directional derivatives: if $\mathbf{d}_1$ and $\mathbf{d}_2$ are two vectors in the domain, then their mixed directional derivatives are

$$D^{r,s}_{\mathbf{d}_1,\mathbf{d}_2}\mathbf{b}(\mathbf{u}) = \frac{n!}{(n-r-s)!}\mathbf{b}[\mathbf{d}_1^{<r>}, \mathbf{d}_2^{<s>}, \mathbf{u}^{<n-r-s>}]. \tag{17.16}$$

This blossom result may also be expressed in terms of Bernstein polynomials. Taking $n-r$ steps of the de Casteljau algorithm with respect to **u**, and then r more with respect to **d** gives

$$D^r_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = \frac{n!}{(n-r)!}\sum_{|\mathbf{j}|=r}\mathbf{b}^{n-r}_{\mathbf{j}}(\mathbf{u})B^r_{\mathbf{j}}(\mathbf{d}). \tag{17.17}$$

Or we might have taken r steps with respect to **d** first, and then $n-r$ ones with respect to **u**. This gives:

$$D^r_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = \frac{n!}{(n-r)!}\sum_{|\mathbf{j}|=n-r}\mathbf{b}^{r}_{\mathbf{j}}(\mathbf{d})B^{n-r}_{\mathbf{j}}(\mathbf{u}). \tag{17.18}$$

Let us now spend some time interpreting our results. First, we note that (17.17) is the analogue of (5.21) in the univariate case. This sounds surprising at first, since (17.17) does not contain differences. Recall, however, that some of the components of **d** must be negative (since $d+e+f=0$). Then the $B^r_{\mathbf{j}}(\mathbf{d})$ yield positive and negative values. We may therefore view terms involving $B^r_{\mathbf{j}}(\mathbf{d})$ as generalized differences. Similarly, (17.18) may be viewed as a generalization of the univariate (5.24).

For $r=1$, the terms $\mathbf{b}^1_{\mathbf{j}}(\mathbf{d})$ in (17.18) have a simple geometric interpretation: since $\mathbf{b}^1_{\mathbf{j}}(\mathbf{d}) = d\mathbf{b}_{\mathbf{j}+\mathbf{e}1} + e\mathbf{b}_{\mathbf{j}+\mathbf{e}2} + f\mathbf{b}_{\mathbf{j}+\mathbf{e}3}$ and $|\mathbf{j}| = n-1$, they denote the affine map of the vector $\mathbf{d} \in \mathbb{E}^2$ to the triangle formed by $\mathbf{b}_{\mathbf{j}+\mathbf{e}1}, \mathbf{b}_{\mathbf{j}+\mathbf{e}2}, \mathbf{b}_{\mathbf{j}+\mathbf{e}3}$. The directional derivative of $\mathbf{b}^n$ is thus a triangular patch whose coefficients are the images of **d** on each subtriangle in the control net (see Figure 17.6). For computational purposes, we would compute the net of the $n\mathbf{b}^1_{\mathbf{j}}(\mathbf{d})$ and use them as the input for a de Casteljau algorithm of an $(n-1)^{\text{st}}$ degree Bézier patch.

Similarly, let us set $r=1$ in (17.17). Then,

$$\begin{aligned} D_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) &= n\sum_{|\mathbf{j}|=1}\mathbf{b}^{n-1}_{\mathbf{j}}(\mathbf{u})B^1_{\mathbf{j}}(\mathbf{d}) \\ &= n(d\mathbf{b}^{n-1}_{\mathbf{e}1} + e\mathbf{b}^{n-1}_{\mathbf{e}2} + f\mathbf{b}^{n-1}_{\mathbf{e}3}). \end{aligned}$$

Since this is true for all directions $\mathbf{d} \in \mathbb{E}^2$, it follows that $\mathbf{b}^{n-1}_{\mathbf{e}1}, \mathbf{b}^{n-1}_{\mathbf{e}2}, \mathbf{b}^{n-1}_{\mathbf{e}3}$ define the *tangent plane* at $\mathbf{b}^n(\mathbf{u})$. This is the direct generalization of the corresponding univariate result. In particular, the three vertices $\mathbf{b}_{0,n,0}, \mathbf{b}_{0,n-1,1}, \mathbf{b}_{1,n-1,0}$ span the tangent plane at $\mathbf{b}_{0,n,0}$ with analogous results for the remaining two corners. Again, we see that the de Casteljau algorithm produces derivative information

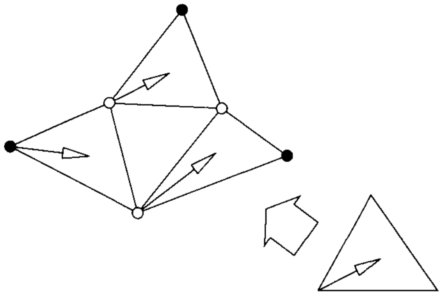


Figure 17.6 Directional derivatives: a vector $\mathbf{d}$ in the domain (bottom right) is mapped to the control net subtriangles to form the vectors $\mathbf{b}_j^1(\mathbf{d})$.

Example 17.2 **Computing a directional derivative.**

Let the coefficients $\mathbf{b}_i$ of a quadratic patch be those from Example 18.1. Let us pick the direction $\mathbf{d} = (1, 0, -1)$. We can then compute the $\mathbf{b}_i^1(\mathbf{d})$:

$$\begin{matrix} & \begin{bmatrix} 3 \\ 0 \\ 6 \end{bmatrix} & \\ \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} & & \begin{bmatrix} 3 \\ 0 \\ 9 \end{bmatrix} \end{matrix}$$

If we now evaluate at $\mathbf{u} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, we obtain the vector

$$\begin{bmatrix} 3 \\ 0 \\ 5 \end{bmatrix},$$

which must still be multiplied by a factor of $n - 1 = 2$ to obtain the directional derivative $D_{(1,0,-1)}\mathbf{b}^2\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$.

Alternatively, we might have taken the $\mathbf{b}_i^1$ from Example 17.1 and evaluated them at $\mathbf{d}$. The result is the same!

as a by-product of the evaluation process; see Example 17.2 for a numerical example.

We next discuss *cross boundary derivatives* of Bézier triangles. Consider the edge $u = 0$ and a direction $\mathbf{d}$ not parallel to it. The directional derivative with

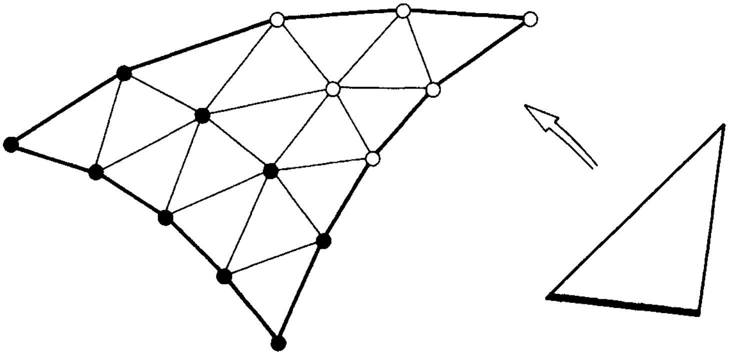


Figure 17.7 Cross boundary derivatives: any first-order cross boundary directional derivative of a quartic, evaluated along the indicated edge, depends only on the two indicated rows of Bézier points.

respect to **d**, evaluated along $u = 0$, is the desired cross boundary derivative. It is given by

$$D^r_{\mathbf{d}}\mathbf{b}^n(\mathbf{u})\Big|_{u=0} = \frac{n!}{(n-r)!}\sum_{|\mathbf{i}_0|=n-r} \mathbf{b}^r_{\mathbf{i}_0}(\mathbf{d})B^{n-r}_{\mathbf{i}_0}(\mathbf{u})\Big|_{u=0}, \qquad (17.19)$$

where $\mathbf{i}_0 = (0, j, k)$. Since $\mathbf{u} = (0, v, w) = (0, v, 1 - v)$, this is a univariate expression, that is, a Bézier curve in terms of barycentric coordinates. Note that it depends only on the $r + 1$ rows of Bézier points closest to the boundary under consideration. Analogous results hold for the other two boundaries; see Figure 17.7. This result is the straightforward generalization of the corresponding univariate result. We will use it for the construction of composite surfaces, just as we did for curves.

17.5 Subdivision

We will later study surfaces that consist of several triangular patches forming a composite C^r surface. Now, we start with a surface consisting of just two triangular patches. Let their domain triangles be defined by points $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$, as shown in Figure 17.8. If the common boundary is through $\mathbf{b}$ and $\mathbf{c}$, then the (domain!) point $\mathbf{d}$ can be expressed in terms of barycentric coordinates of $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

$$\mathbf{d} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c}.$$

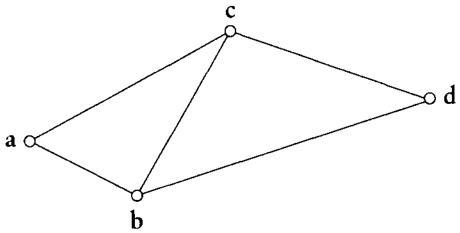


Figure 17.8 Subdivision: the domain geometry.

Suppose now that a Bézier triangle $\mathbf{b}^n$ is given that has the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ as its domain, such that we have barycentric coordinates $\mathbf{a} = \mathbf{e1}, \mathbf{b} = \mathbf{e2}, \mathbf{c} = \mathbf{e3}$. Of course, the patch is defined over the whole domain plane, in particular over the triangle $\mathbf{d}, \mathbf{c}, \mathbf{b}$. What are the Bézier points $\mathbf{c_i}$ of $\mathbf{b}^n$ if we consider only the part of it that is defined over $\mathbf{d}, \mathbf{c}, \mathbf{b}$?

The answer was already given with (17.4):

$$\mathbf{c_i} = \mathbf{b}[\mathbf{d}^{<i>}, \mathbf{e2}^{<j>}, \mathbf{e3}^{<k>}]; \quad i + j + k = n. \tag{17.20}$$

So all we have to do is compute the point $\mathbf{b}(\mathbf{d})$ using the de Casteljau algorithm, and the intermediate points are the desired patch control vertices!

Some of the $\mathbf{c_i}$ deserve special attention: the common boundary of the two patches is characterized by $u = 0$. The Bézier points of the corresponding boundary curve must be same for both patches; we have

$$\mathbf{c_{j_0}} = \mathbf{b}[\mathbf{e2}^{<j>}, \mathbf{e3}^{<k>}] = \mathbf{b_{j_0}}; \quad j + k = n, \tag{17.21}$$

where $\mathbf{j}_0 = (0, j, k)$. We also have

$$\mathbf{c}_{n00} = \mathbf{b}[\mathbf{d}^{<n>}],$$

thus asserting, as expected, that we find $\mathbf{c}_{n00}$ as a point on the surface, evaluated at $\mathbf{d}$. A numerical example is given in Example 17.3.

We thus have an algorithm that allows us to construct the Bézier points of the "extension" of $\mathbf{b}^n$ to an adjacent patch. It should be noted that this algorithm does not use convex combinations (when $\mathbf{d}$ is outside $\mathbf{a}, \mathbf{b}, \mathbf{c}$). It performs piecewise linear extrapolation, and should therefore not be expected to be numerically very stable.

If $\mathbf{d}$ is inside $\mathbf{a}, \mathbf{b}, \mathbf{c}$, then we do use convex combinations only, and (17.20) provides a *subdivision algorithm*. Just as $\mathbf{d}$ subdivides the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ into three subtriangles, the point $\mathbf{b}^n(\mathbf{d})$ subdivides the triangular patch into three

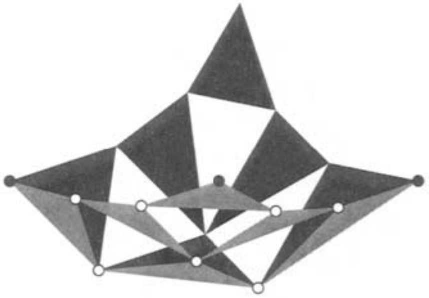


Figure 17.9 Subdivision: the intermediate points from the de Casteljau algorithm form the three subpatch control nets.

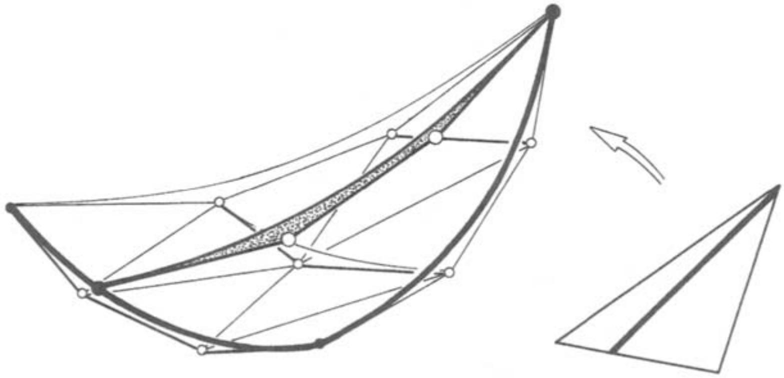


Figure 17.10 Subdivision: the Bézier points of a surface curve that is the image of a straight line through one of the domain triangle vertices.

subpatches. Equation (17.20) provides the corresponding Bézier points. Figure 17.9 gives an illustration.

Just as in the curve case, if we insert a dense sequence of points into the domain triangle, the resulting sequence of control nets will converge to the surface. This fact may be used in rendering techniques or in intersection algorithms.

A special case arises if $\mathbf{d}$ is on one of the edges of the domain triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Then (17.20) generates the Bézier points of the surface curve through $\mathbf{b}^n(\mathbf{d})$ and the opposite patch corner; see Figure 17.10. Such curves, joining a vertex to a point on the opposite edge, are called radial lines.

17.6 Differentiability

Consider two triangular patches that are maps of two adjacent domain triangles as shown in Figure 17.8. Any straight line in the domain that crosses the common edge is mapped onto a composite curve in $\mathbb{E}^3$, having one segment in each patch. If all composite curves that can be obtained in this way are C^r curves, then we say that the two patches are C^r continuous.

Equation (17.20) gives a condition by which two adjacent patches $\mathbf{b}$ and $\hat{\mathbf{b}}$ can be part of one global polynomial surface. Both patches share the line $u = 0$, and are clearly C^n along it. If we relax this to some lower degree r of continuity, we have to consider only $r + 1$ layers of control points of each patch, and we have a condition for C^r continuity between adjacent patches:

$$\hat{\mathbf{b}}_{(\rho,j,k)} = \mathbf{b}^{\rho}_{\mathrm{j}0}(\mathbf{d});\ \rho = 0, \ldots, r. \tag{17.22}$$

Equation (17.22) is a necessary and sufficient condition for the C^r continuity of two adjacent patches. If the patches share the boundaries $v = 0$ or $w = 0$, the conditions are analogous.

For $r = 0$, (17.22) states that the two patches must share a common boundary control polygon. The case $r = 1$ is more interesting because (17.22) becomes

$$\hat{\mathbf{b}}_{(1,j,k)} = v_1\mathbf{b}_{1,j,k} + v_2\mathbf{b}_{0,j+1,k} + v_3\mathbf{b}_{0,j,k+1}. \tag{17.23}$$

Thus each $\hat{\mathbf{b}}_{(1,j,k)}$ is obtained as a barycentric combination of the vertices of a boundary subtriangle of the control net of $\mathbf{b}^n$. Moreover, for all $j + k = n - 1$, these barycentric combinations are identical. Thus all pairs of subtriangles shown in Figure 17.11 are coplanar, and each pair is an affine map of the pair of domain

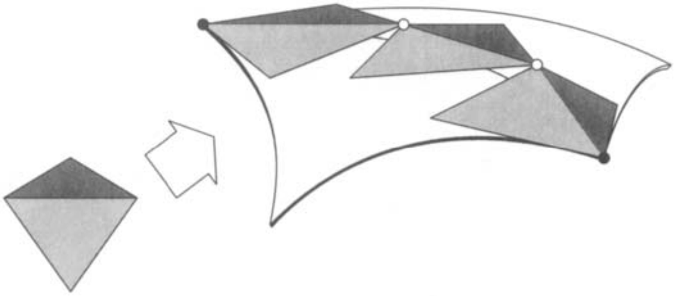


Figure 17.11 C^1 continuity: the shaded pairs of triangles must be coplanar and be an affine map of the two domain triangles.

Example 17.3 **Computing a C^1 patch extension.**

We refer to Example 17.1. Let us define a second domain triangle that shares the edge $w = 0$ with the given domain triangle and that has a third vertex **d**. Let **d**'s barycentric coordinates with respect to the initial domain triangle be $(1,1,-1)$. We can perform one de Casteljau algorithm step and obtain for the $\mathbf{b}_i^1(\mathbf{d})$:

$$\begin{array}{c} \begin{bmatrix} 3 \\ 6 \\ 6 \end{bmatrix} \\ \begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix} \begin{bmatrix} 6 \\ 3 \\ 15 \end{bmatrix} . \end{array}$$

A quadratic patch that is C^1 with the given one along its edge $w = 0$ is then given by the control net

$$\begin{array}{c} \begin{bmatrix} 0 \\ 6 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 3 \\ 6 \\ 6 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 6 \end{bmatrix} \\ \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} \begin{bmatrix} 6 \\ 3 \\ 15 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \\ 9 \end{bmatrix} , \end{array}$$

where the $\bullet$ entries could be any numbers; they have no influence on C^1 continuity between the two patches.

triangles of the two patches.[4] We call the pairs of coplanar subtriangles that satisfy this condition *affine pairs*. Example 17.3 gives more details.

Figure 17.12 shows a composite C^1 surface that consists of several Bézier triangles. The (wire frame) plot of the surface does not look very smooth. This is due to the different spacing of isoparametric lines in the plot, not to discontinuities in the surface. The generation of planar slices of the surfaces shows that it is in fact C^1.

[4] It is *not* sufficient that the pairs are coplanar—this does not even guarantee a continuous tangent plane.

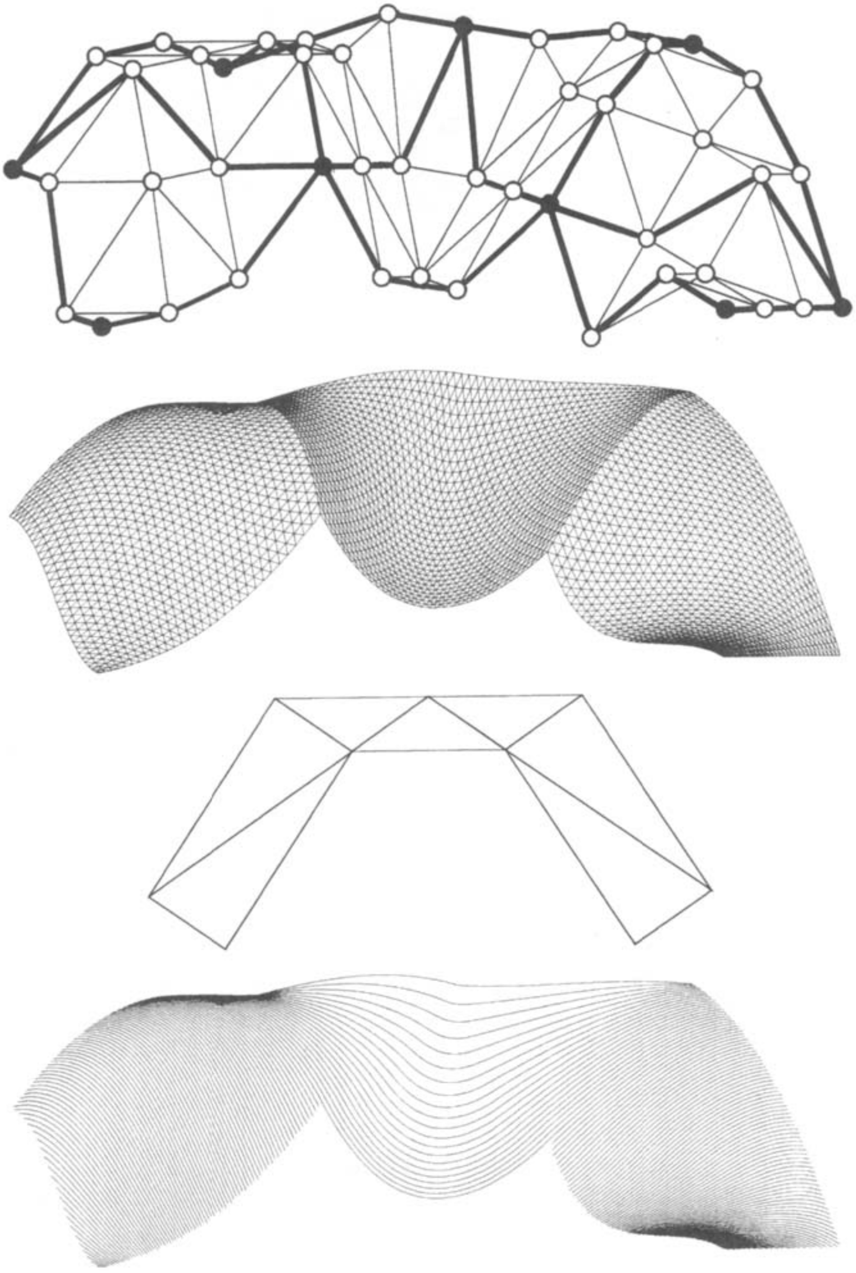


Figure 17.12 Bézier triangles: a composite C^1 surface. Top: the control net; next: the piecewise cubic surface; next: the domain triangles; next: planar slices through the surface.

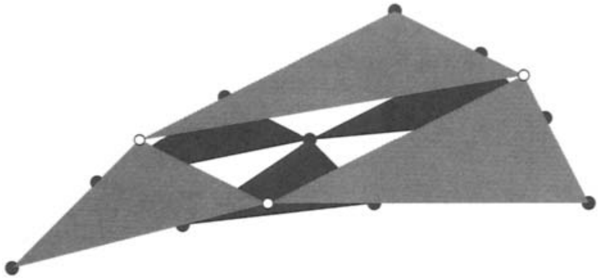


Figure 17.13 Degree elevation: a quadratic control net together with the equivalent cubic net.

17.7 Degree Elevation

It is possible to write $\mathbf{b}^n$ as a Bézier triangle of degree $n+1$:

$$\sum_{|\mathbf{i}|=n} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u}) = \sum_{|\mathbf{i}|=n+1} \mathbf{b}_{\mathbf{i}}^{(1)} B_{\mathbf{i}}^{n+1}(\mathbf{u}). \tag{17.24}$$

The control points $\mathbf{b}_{\mathbf{i}}^{(1)}$ are obtained from

$$\mathbf{b}_{\mathbf{i}}^{(1)} = \frac{1}{n+1}[i\mathbf{b}_{\mathbf{i}-\mathbf{e}1} + j\mathbf{b}_{\mathbf{i}-\mathbf{e}2} + k\mathbf{b}_{\mathbf{i}-\mathbf{e}3}]. \tag{17.25}$$

For a proof, we multiply the left-hand side of (17.24) by $u+v+w$ and compare coefficients of like powers. Figure 17.13 illustrates the case $n=2$. Degree elevation is performed by piecewise linear interpolation of the original control net. Therefore, the degree elevated control net lies in the convex hull of the original one.

As in the univariate case, degree elevation may be repeated. That process generates a sequence of control nets that have the surface patch as their limit (Farin [188]). More details are in Farin [197].

17.8 Nonparametric Patches

In an analogy to the univariate case, we may write the function

$$z = \sum_{|\mathbf{i}|=n} b_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u}) \tag{17.26}$$

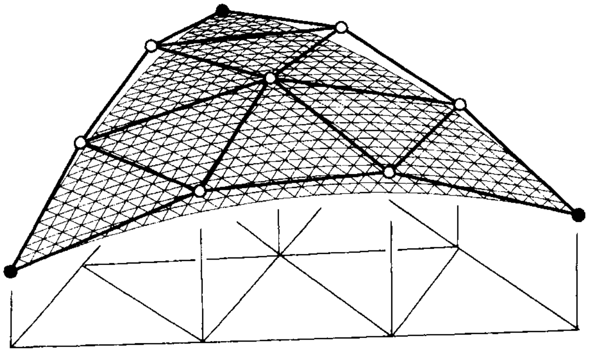


Figure 17.14 Nonparametric patches: the abscissas of the control net are the n-partition points of the domain triangle.

as a surface

$$\begin{bmatrix} u \\ v \\ w \\ z \end{bmatrix} = \sum_{|\mathbf{i}|=n} \begin{bmatrix} i/n \\ j/n \\ k/n \\ b_{\mathbf{i}} \end{bmatrix} B_{\mathbf{i}}^n(\mathbf{u}).$$

Thus the abscissa values of the control polygon of a nonparametric patch are given by the triples $\mathbf{i}/n$, as illustrated in Figure 17.14. The last equation holds because of the *linear precision* property of the Bernstein polynomials $B_{\mathbf{i}}^n$,

$$u = \sum_{|\mathbf{i}|=n} \frac{i}{n} B_{\mathbf{i}}^n(\mathbf{u}),$$

and analogous formulas for v and w. The proof is by degree elevation from 1 to n of the linear function u. Example 17.4 shows a nonparametric patch.

Nonparametric Bézier triangles play an important role in the investigation of spaces of piecewise polynomials, as studied in approximation theory. Their use has facilitated the investigation of one of the main open questions in that field: what is the dimension of those function spaces? (See, for instance, Alfeld and Schumaker [7].) They have also been useful in defining nonparametric piecewise polynomial interpolants; see, for example, Barnhill and Farin [29], Farin [194], Petersen [471], or Sablonnière [525].

Example 17.4 **A nonparametric quadratic patch.**

The bivariate function $z = x^2 + y^2$ may be written as a quadratic nonparametric Bézier patch over the triangle $(0,0), (2,0), (0,2)$. Its coefficients are:

$$\begin{array}{lll} \begin{bmatrix}0\\2\\4\end{bmatrix} & & \\ \begin{bmatrix}0\\1\\0\end{bmatrix} & \begin{bmatrix}1\\1\\0\end{bmatrix} & \\ \begin{bmatrix}0\\0\\0\end{bmatrix} & \begin{bmatrix}1\\0\\0\end{bmatrix} & \begin{bmatrix}2\\0\\4\end{bmatrix} \end{array}.$$

17.9 The Multivariate Case

The concepts of triangular Bézier patches are easily taken to higher dimensions. To begin with, let four points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ define a 3D tetrahedron. If $\mathbf{p}$ is a fifth point, then it may uniquely be written as

$$\mathbf{p} = u_1\mathbf{p}_1 + u_2\mathbf{p}_2 + u_3\mathbf{p}_3 + u_4\mathbf{p}_4$$

where the u_i sum to one and are called the barycentric coordinates of $\mathbf{p}$ with respect to the four $\mathbf{p}_i$. See Figure 17.15 for an illustration. Similar to the 2D case, we have

$$u_1 = \frac{\det[\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]}{\det[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]}, u_2 = \frac{\det[\mathbf{p}_1, \mathbf{p}, \mathbf{p}_3, \mathbf{p}_4]}{\det[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]}, \quad \text{and so on.}$$

We abbreviate $\mathbf{u} = (u_1, u_2, u_3, u_4)$, $\mathbf{i} = (i_1, i_2, i_3, i_4)$ and define Bernstein polynomials

$$B_{\mathbf{i}}^n(\mathbf{u}) = \frac{n!}{i_1!i_2!i_3!i_4!} u_1^{i_1} u_2^{i_2} u_3^{i_3} u_4^{i_4}.$$

This allows us to define Bézier functions:

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

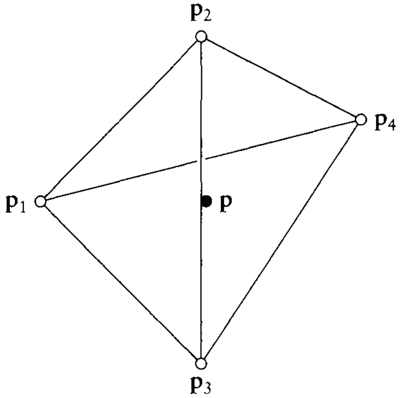


Figure 17.15 Trivariate polynomials: an illustration of the domain.

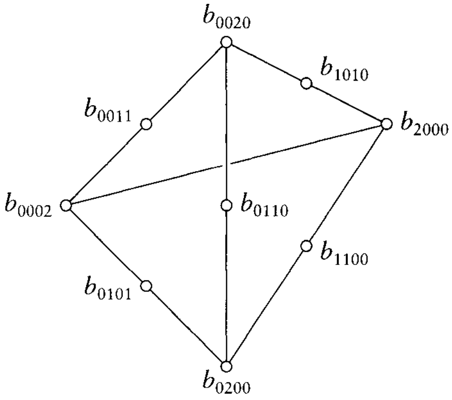


Figure 17.16 Trivariate polynomials: the Bézier ordinates for a trivariate quadratic.

These are *trivariate* functions whose domain is the tetrahedron defined by the four $\mathbf{p}_i$. The $b_\mathbf{i}$ are the Bézier ordinates. Essentially, all properties of Bézier triangular patches carry over to this case; we mention the de Casteljau algorithm:

$$b_\mathbf{i}^r(\mathbf{u}) = u_1 b_{\mathbf{i}+\mathbf{e}1}^{r-1}(\mathbf{u}) + \ldots + u_4 b_{\mathbf{i}+\mathbf{e}4}^{r-1}(\mathbf{u}).$$

For an example of the geometry for the case $n = 2$ see Figure 17.16.

The generalization to higher dimensions is straightforward; we simply have to replace the 4 by a higher dimension d. The use of these multivariate functions is in scientific computing and visualization; see [6], [72], [115], [547], [139], [197], [261], [371], [618].

17.10 S-Patches

Bézier patches in higher dimensions have a fairly abstract flair; yet, they may be used to design "normal" surfaces in 3D. One method to do this is due to T. DeRose and C. Loop; see [167], [168], [398]. They devised a method to contruct patches with an arbitrary number s of edges.

In order to build an s-sided patch, we take as its domain a convex 2D polygon $\mathcal{P}_s$ with vertices $\mathbf{p}_1, \ldots, \mathbf{p}_s$, ordered counterclockwise. For every point $\mathbf{p}$ inside this polygon, we construct "generalized barycentric coordinates" $u_1, \ldots, u_s$, by considering all triangles formed by $\mathbf{p}$ and the polygon vertices $\mathbf{p}_i$—this approach is due to Gregory and Charrot [110]. If the area[5] of triangle $\mathbf{p}, \mathbf{p}_i, \mathbf{p}_{i+1}$ is denoted by Δ_i, then we define

$$\pi_i = \Delta_1 \cdot \ldots \cdot \Delta_{i-2} \cdot \Delta_{i+1} \cdot \ldots \cdot \Delta_s.$$

Since these π_i do not sum to one, we normalize and obtain

$$u_i = \frac{\pi_i}{\pi_1 + \ldots + \pi_s}. \tag{17.27}$$

For $s = 3$, we obtain standard barycentric coordinates in a triangle. If $\mathbf{p}$ is on the edge $\mathbf{p}_i, \mathbf{p}_{i+1}$ of $\mathcal{P}_s$, then only u_i and u_{i+1} are nonzero.

The key idea now is this: since the u_i sum to one, they may be interpreted as barycentric coordinates of an $(s-1)$-dimensional simplex $\mathcal{S}_s$; see Section 3.5. We call the vertices of the simplex $\mathbf{s}_\mathbf{i}$; $|\mathbf{i}| = 1$. In particular, if $s = 3$, the simplex is a triangle; if $s = 4$, it is a tetrahedron. We associate the point $\mathbf{p}_i$ of $\mathcal{P}_s$ with the point $[0^{<i-1>}, 1, 0^{<s-i>}]$ of $\mathcal{S}_s$. All points $\mathbf{u} = (u_1, \ldots, u_s)$ inside this simplex that have barycentric coordinates (17.27) trace out a two-dimensional subset of the simplex. It has the property that if $\mathbf{p}$ is on one of the edges of $\mathcal{P}_s$, then $\mathbf{u}$ is on the corresponding edge of $\mathcal{S}_s$.

Next, we may define a Bézier patch over $\mathcal{S}_s$. Recall that $\mathcal{S}_s$ is just the domain of that patch; the corresponding Bézier points may "live" in any dimension. We restrict them to be 3D points $\mathbf{b}_\mathbf{i}$, with $|\mathbf{i}| = n$, the degree of the Bézier patch. This number n is referred to as the depth of the S-patch. The equation of the S-patch becomes

$$\mathbf{x}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_\mathbf{i} B_\mathbf{i}^n(\mathbf{u}). \tag{17.28}$$

[5] If $\mathbf{p}$ is outside the polygon, we use signed areas.

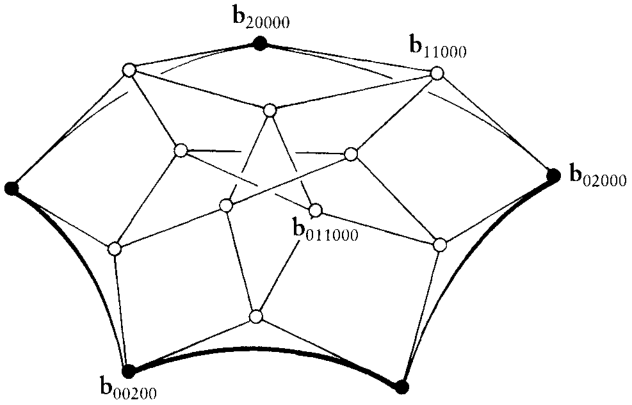


Figure 17.17 S-patches: an example for $s = 4$ and $n = 2$.

Since we are interested in a surface patch, that is, a map of the inside of $\mathcal{P}$ to that patch, we constrain $\mathbf{u}$ in (17.28) to satisfy (17.27).

In order to define an s-sided S-patch of depth n, we thus have to specify a control net with vertices $\mathbf{b}_\mathbf{i}$, having the connectivity of $\mathcal{S}_s$. The polygon $\mathcal{P}_s$ is its domain; if we want to evaluate at a point $\mathbf{p} \in \mathcal{P}_s$, we first find $\mathbf{p}$'s generalized barycentric coordinates $\mathbf{u}$, and then evaluate (17.28) by carrying out an $s -$ 1-dimensional de Casteljau algorithm with the 3D points $\mathbf{b}_\mathbf{i}$ as control net. See Figure 17.17 for an illustration.

We note that formally the degree of $\mathbf{x}(\mathbf{u})$ is n. But since each u_i is a rational linear function of $\mathbf{p}$'s location, the actual structure of the S-patch is that of a rational polynomial of degree $n(s - 2)$. However, when $s = 3$, S-patches are standard polynomial Bézier triangles of degree n.

We should note that more approaches exist for s-sided patches: refer to the survey articles [595] and [290] and also to [320], [332], [353], [484], [518], [520], [581], [596], [597], [610], [628].

17.11 Implementation

We include a function for the evaluation of a Bézier triangle. It uses a linear array for the coefficients $\mathbf{b}_\mathbf{i}$ in order to avoid a waste of storage by putting them into a square matrix.

```
tri_decas(bpts, tri_num, ndeg, u, b, patch_pt )
/*
 Function: Triangular de Casteljau algorithm for an n^th
           degree triangular Bezier patch.
           Algorithm is applied once for a given (u,v,w) and works on one
           coordinate only.

 Input:    bpts[i] ........... Bezier points (of one coordinate)
                               as a linear array (see below).
                               i=0...tri_num
           tri_num ........... Based on the degree of the patch.
                               (n+1)(n+2)/2
           ndeg .............. Degree (n) of the patch.
           u[i] .............. Barycentric coordinates (u,v,w) of
                               evaluation point.  i=0,2
           b[i]............... A working array with dimension >=
                               to bpts[].

 Output:   patch_pt .......... One coordinate of the point on
                               the patch evaluated at (u,v,w).
           b[] ............... Contents have been changed.

 Linear array structure: It is assumed that the usual (i,k,j) structure
                         of the Bezier net has been put into a linear
                         array in the following manner.
                         (E.g., for n=3)
                         b_(300) --> bpts[0] (u=1)
                         b_(030) --> bpts[6] (v=1)
                         b_(003) --> bpts[9] (w=1)
*/
```

17.12 Problems

1 Find the barycentric coordinates of the incenter of a triangle.

***2** Work out exactly how terms involving $B^r_{\mathbf{j}}(\mathbf{d})$ generalize the univariate difference operator.

3 Show that the Bernstein polynomials $B^n_{\mathbf{i}}(\mathbf{u})$ are linearly independent.

4 What is the geometric interpretation of the quadratic blossom $\mathbf{b}[\mathbf{u}_1, \mathbf{u}_2]$? Of the cubic blossom $\mathbf{b}[\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3]$?

P1 In Section 16.8, we saw how to modify surfaces by embedding them in the unit cube and then distorting it using trivariate tensor product schemes. Experiment with the following: instead of embedding a surface in a cube, embed it in a tetrahedron and distort it using a *Bézier tetrahedron.*

P2 Program up S-patches with $n = 3$ and varying values of s.

P3 Work out a degree reduction procedure for Bézier triangles. Literature: Petersen [471].

P4 Problem P1 of Chapter 7 addressed the similarity between Aitken and de Casteljau algorithms. Generalize to the triangular patch case and write a program that modifies `tri_decas` accordingly.

This Page Intentionally Left Blank

Practical Aspects of Bézier Triangles 18

Bézier triangular patches are not only the most natural generalization of Bézier curves, they also offer a variety of applications. They lend themselves easily to patches on spheres or quadrics, and they may be used for *scattered data interpolation.* Although rectangular patches are traditionally dominating CAD/CAM, some recent applications of triangular patches are in gaming and animation.

18.1 Rational Bézier Triangles

Following the familiar theme of generating rational curve and surface schemes, we define a rational Bézier triangle to be the projection of a polynomial 4D Bézier triangle. We thus have:

$$\mathbf{b}^n(\mathbf{u}) = \mathbf{b}_0^n(\mathbf{u}) = \frac{\sum\limits_{|\mathbf{i}|=n} w_{\mathbf{i}} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u})}{\sum\limits_{|\mathbf{i}|=n} w_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u})}, \tag{18.1}$$

where, as usual, the $w_{\mathbf{i}}$ are the weights associated with the control vertices $\mathbf{b}_{\mathbf{i}}$. For positive weights, we have the convex hull property. We also have affine and projective invariance.

Rational Bézier triangles may be evaluated by a de Casteljau algorithm in a not too surprising way.

Rational de Casteljau algorithm

Given: A triangular array of points $\mathbf{b}_{\mathbf{i}} \in \mathbb{E}^3$; $|\mathbf{i}| = n$, corresponding weights $w_{\mathbf{i}}$, and a point in a domain triangle with barycentric coordinates $\mathbf{u}$.

Set:

$$\mathbf{b}_{\mathrm{i}}^{r}(\mathbf{u}) = \frac{u w_{\mathrm{i+e1}}^{r-1}\mathbf{b}_{\mathrm{i+e1}}^{r-1} + v w_{\mathrm{i+e2}}^{r-1}\mathbf{b}_{\mathrm{i+e2}}^{r-1} + w w_{\mathrm{i+e3}}^{r-1}\mathbf{b}_{\mathrm{i+e3}}^{r-1}}{w_{\mathrm{i}}^{r}}, \qquad (18.2)$$

where

$$w_{\mathrm{i}}^{r} = w_{\mathrm{i}}^{r}(\mathbf{u}) = u w_{\mathrm{i+e1}}^{r-1}(\mathbf{u}) + v w_{\mathrm{i+e2}}^{r-1}(\mathbf{u}) + w w_{\mathrm{i+e3}}^{r-1}(\mathbf{u})$$

and

$$r = 1, \ldots, n \quad \text{and} \quad |\mathrm{i}| = n - r$$

and $\mathbf{b}_{\mathrm{i}}^{0}(\mathbf{u}) = \mathbf{b}_{\mathrm{i}}$, $w_{\mathrm{i}}^{0} = w_{\mathrm{i}}$. Then $\mathbf{b}_{0}^{n}(\mathbf{u})$ is the point with parameter value $\mathbf{u}$ on the rational Bézier triangle $\mathbf{b}^{n}$.

This algorithm works since we can interpret each intermediate $\mathbf{b}_{\mathrm{i}}^{r}$ as the projection of the corresponding point in the de Casteljau algorithm of the nonrational 4D preimage of our patch.

Something surprising happens now. Everything thus far was yet another exercise in generating rational schemes. In the case of rational Bézier curves, the initial weights could be used to define weight points $\mathbf{q}_i$, as described in Section 13.2. In the triangle case, we can also define weight points $\mathbf{q}_{\mathrm{i}}$ by setting

$$\mathbf{q}_{\mathrm{i}} = \frac{w_{\mathrm{i+e1}}\mathbf{b}_{\mathrm{i+e1}} + w_{\mathrm{i+e2}}\mathbf{b}_{\mathrm{i+e2}} + w_{\mathrm{i+e3}}\mathbf{b}_{\mathrm{i+e3}}}{w_{\mathrm{i+e1}} + w_{\mathrm{i+e2}} + w_{\mathrm{i+e3}}}; \quad |\mathrm{i}| = n - 1.$$

The usefulness of the $\mathbf{q}_i$ in the curve case stemmed from the fact that they could be used as a design handle: we could define points $\mathbf{q}_i$ and then retrieve the weights w_i. Now, in the triangle case, this is no longer possible (first noted by Ramshaw [498]). We can see why just by considering the quadratic case $n = 2$, illustrated in Figure 18.1.

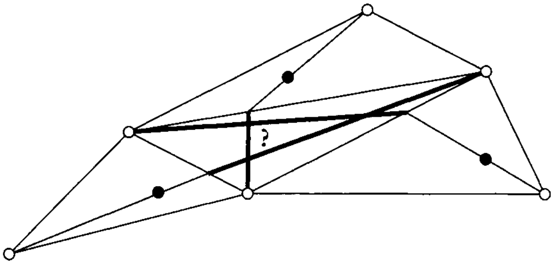


Figure 18.1 Weight points: an arbitrary choice of the three weight points (solid) would overdetermine the location of $\mathbf{p}$.

If we were given a set of weights $w_{\mathbf{i}}$; $|\mathbf{i}| = 2$, we would not only generate the weight points $\mathbf{q}_{\mathbf{i}}$, but also the point

$$\mathbf{p} = \frac{w_{011}\mathbf{b}_{011} + w_{101}\mathbf{b}_{101} + w_{110}\mathbf{b}_{110}}{w_{011} + w_{101} + w_{110}}.$$

The point $\mathbf{p}$ is then at the intersection of the three straight lines $\overline{\mathbf{q}_{001}, \mathbf{b}_{110}}$, and so on. If we prescribed the three $\mathbf{q}_{\mathbf{i}}$ arbitrarily (as shown in Figure 18.1), those straight lines would not intersect in one point any more, thus leaving $\mathbf{p}$ overdetermined. Since the existence of a set of weights implies the existence of $\mathbf{p}$, the nonexistence of $\mathbf{p}$ implies that we cannot find a set of weights if we prescribe the $\mathbf{q}_{\mathbf{i}}$ arbitrarily.

For higher degrees, the situation is analogous. So far, no geometric means is known that could define weights similar to the weight point approach for curves. A first step could be the paper by G. Albrecht [4] or H. Theisel [586].

We now give a formula for the *directional derivative* of a rational Bézier triangle. Just as in Section 17.4, let $\mathbf{d}$ denote a direction in the domain triangle, expressed derivative $D_{\mathbf{d}}$ of a rational triangular Bézier patch $\mathbf{b}^n(\mathbf{u})$. Proceeding exactly as in the curve case (see Section 12.4), we obtain:

$$D_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = \frac{1}{w(\mathbf{u})}[\dot{\mathbf{p}}(\mathbf{u}) - D_{\mathbf{d}}(\mathbf{u})\mathbf{b}^n(\mathbf{u})],$$

where we have set

$$\mathbf{p}(\mathbf{u}) = w(\mathbf{u})\mathbf{b}^n(\mathbf{u}) = \sum_{|\mathbf{i}|=n} w_{\mathbf{i}}\mathbf{b}_{\mathbf{i}}B^n_{\mathbf{i}}(\mathbf{u}).$$

Higher derivatives follow the pattern outlined in Section 12.4, that is,

$$D^r_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = \frac{1}{w(\mathbf{u})}\left[D^r_{\mathbf{d}}\mathbf{p}(\mathbf{u}) - \sum_{j=1}^{r} D^j_{\mathbf{d}}w(\mathbf{u})D^{r-j}_{\mathbf{d}}(\mathbf{u})\right].$$

18.2 Quadrics

There were (at least) two motivations for the use of rational Bézier curves: they are projectively invariant, and they allow us to represent conics in the form of rational quadratics. While the first argument holds trivially for rational Bézier triangles, the second one does not carry over immediately.

The proper generalization of a conic to the case of surfaces is a *quadric surface*, quadric for short. A conic (curve) has the implicit equation $q(x, y) = 0$, where q

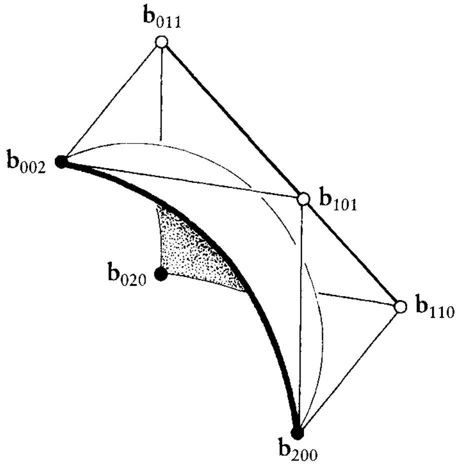


Figure 18.2 The octant of a sphere: an attempt to write it as a quadratic rational Bézier triangle. The weights of the solid control points are unity; the others have weight 1/2.

is a quadratic polynomial in x and y (see Section 12.5). Similarly, a quadric has the implicit equation $q(x, y, z) = 0$, where q is quadratic in x, y, z.

Quadrics are of importance in almost all solid modeling systems—these systems rely heavily on the ability to decide quickly if a given point is inside or outside a given object. If that object is bounded by simple implicit surfaces, such a decision is simple and reliable. If the object's boundaries are made up from, say, bicubics, the same decision is much more time consuming and error prone.

Every finite arc of a conic could be written as a quadratic rational Bézier curve—can we also write every triangle-shaped region on a quadric as a rational quadratic Bézier triangle? Let us try an octant of a sphere. In rational quadratic form, it would have six control net coefficients and six associated weights. Since a rational quadratic has no interior Bézier points, we only have to concentrate on the boundary curve representation. They are quarters of circles, and their representation is given in Section 12.7; see also Figure 18.2. Thus we should be done, but actually, we are stuck: if we try to evaluate our rational quadratic at $\mathbf{u} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, we do not end up on the sphere! Thus *not every triangular quadric patch can be represented as a triangular rational quadratic patch.*

We have seen that rational quadratic triangular patches and quadric triangular patches are far from being in a one-to-one relationship. Luckily, this does not mean that we cannot express quadric patches as rational Bézier triangles—we just have to use higher degrees. In fact, rational *quartic* Bézier triangles are always sufficient for this purpose. Our initial example, the octant of the sphere, has the following rational quartic representation (Farin, Piper, and Worsey [213]). The control net (for the octant of the *unit sphere*) is given by

$$\begin{array}{c}
[0,0,1] \\
[\alpha,0,1] \quad [0,\alpha,1] \\
[\beta,0,\beta] \quad [\gamma,\gamma,1] \quad [0,\beta,\beta] \\
[1,0,\alpha] \quad [1,\gamma,\gamma] \quad [\gamma,1,\gamma] \quad [0,1,\alpha] \\
[1,0,0] \quad [1,\alpha,0] \quad [\beta,\beta,0] \quad [\alpha,1,0] \quad [0,1,0]
\end{array}$$

where

$$\alpha = (\sqrt{3}-1)/\sqrt{3}, \qquad \beta = (\sqrt{3}+1)/2\sqrt{3}, \qquad \gamma = 1-(5-\sqrt{2})(7-\sqrt{3})/46.$$

The weights are:

$$w_{040} = 4\sqrt{3}(\sqrt{3}-1), w_{031} = 3\sqrt{2}, w_{202} = 4, w_{121} = \frac{\sqrt{2}}{\sqrt{3}}(3+2\sqrt{2}-\sqrt{3}),$$

the other ones following by symmetry.

To represent the whole sphere, we would assemble eight copies of this octant patch. Other representations are also possible: each octant may be written as a rational biquadratic patch (introducing singularities at the north and south poles); see [480]. A representation of the whole sphere as two rational bicubics (Piegl [475]) turned out to be incorrect (see Cobb [119]). Quite a different way of representing the sphere is also due to J. Cobb: he covers it with six rational bicubics having a cubelike connectivity [118].

Let us now discuss the following question: given a rational quadratic Bézier triangle, what are the conditions under which it represents a quadric patch? The following will be useful for that purpose.

We defined a conic section as the projective map of a parabola, the only conic allowing a polynomial parametrization. For surfaces, we again consider those quadrics that permit polynomial parametrizations: these are the paraboloids, consisting of elliptic and hyperbolic paraboloids and of the parabolic cylinders. Every quadric surface may be defined as a projective image of one of these paraboloids.[1]

A paraboloid may be represented by a parametric polynomial surface of degree two. However, as we have seen, not every parametric quadratic is a paraboloid. We need an extra condition, which is easily formulated if we write the quadratic

[1] W. Boehm (1990), private communication.

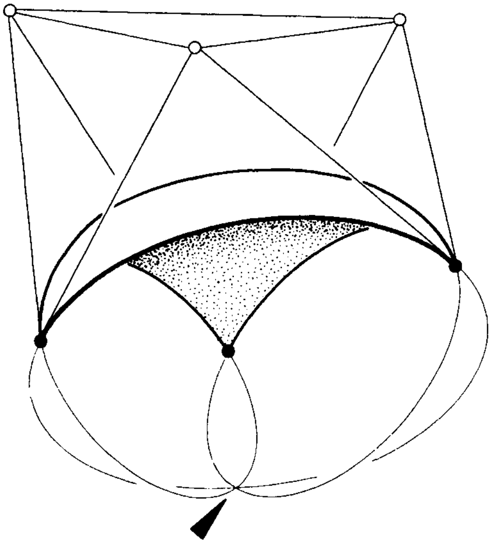


Figure 18.3 Quadrics as rational quadratics: the defining condition is that the extensions of the three boundary curves meet in one point (marked by an arrow) and have coplanar tangents there.

surface in triangular Bézier form: *a quadratic Bézier triangle is an elliptic or hyperbolic paraboloid if and only if the second derivative vectors of the three boundary curves are parallel to each other. It is a parabolic cylinder if those three vectors are only coplanar.* This statement is due to W. Boehm.

For a proof, we observe that nonparametric or functional quadratic polynomials [i.e., of the form $z = f(x, y)$] include all three types of paraboloids, and all three satisfy the conditions of the theorem. Next, we observe that every paraboloid may be obtained as an affine map of a paraboloid of the same type. Thus every paraboloid may be obtained as an affine map of a functional quadratic surface. Consequently, the control net of any paraboloid must be an affine image of the control net of a functional quadratic Bézier triangle.

In a projective setting, we would say that the boundary curves of functional paraboloids intersect in one point (this may be the point at infinity) and have coplanar tangents there. Since all quadrics may be obtained from the functional ones by projective maps, we obtain the following characterization of quadric surfaces: *a rational quadratic Bézier triangle is a quadric if and only if all three boundary curves meet in a common point and have coplanar tangents there.* Figure 18.3 gives an illustration; for more details, see Boehm and Hansford [83].

A quadric is determined by nine points. If nine points $(x_1, y_1, z_1), \ldots, (x_9, y_9, z_9)$ are given, then their interpolating quadric may be written in implicit form as follows:

$$q(x,y,z) = \begin{vmatrix} x^2 & y^2 & z^2 & xy & xz & yz & x & y & z & 1 \\ x_1^2 & y_1^2 & z_1^2 & x_1y_1 & x_1z_1 & y_1z_1 & x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_9^2 & y_9^2 & z_9^2 & x_9y_9 & x_9z_9 & y_9z_9 & x_9 & y_9 & z_9 & 1 \end{vmatrix} = 0. \quad (18.3)$$

Though theoretically a solution to the nine-point interpolation problem, (18.3) does not lend itself to successful numerical treatment. The reason is the = 0 part of that equation: floating point numbers rarely *equal* zero, and so tolerances must be devised. It is not obvious how to do this here.

18.3 Interpolation

Triangular patches may also be used for *scattered data interpolation*: in that context, we are given a set of points $\mathbf{x}_i$ in the plane, each associated with a function value z_i and a tangent plane T_i.[2] We wish to find a surface $z(\mathbf{x})$ that interpolates to the given data, that is, $z(\mathbf{x}_i) = z_i$. As a preprocessing step, the data sites $\mathbf{x}_i$ are triangulated according to Section 3.6. We then wish to construct a triangular patch over each of the triangles. These patches will not be parametric, but rather functional as in Section 17.8.

We should note that more methods exist for the problem of scattered data interpolation; excellent surveys are in [246], [247], [248], [339].

18.4 Cubic and Quintic Interpolants

In order to illustrate the basic methodology, we discuss the so-called *nine-parameter interpolant*. We now assume that we are given function values and also gradients (i.e., x- and y-partials, or tangent planes).[3]

A nine-parameter interpolant will be cubic over each triangle, thus being determined by ten coefficients or Bézier ordinates. Nine of these are immediately determined by the given data; see Figure 18.4: clearly the three values $b_{300}, b_{030}, b_{003}$ are simply the given function values at the triangle's vertices. Instead of considering all remaining coefficients, we restrict ourselves to b_{012}. Its

[2] The assumption of *given* tangent planes or gradients is not always realistic. Where tangent plane information is not supplied, it will have to be estimated.

[3] The assumption of given tangent planes is not too realistic; most likely, these will have to be estimated from the given function values.

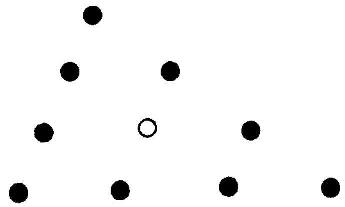


Figure 18.4 The nine-parameter interpolant: nine of its ten Bézier ordinates are directly determined from the given data.

location $\mathbf{a}_{012}$ in the x, y-plane is given by

$$\mathbf{a}_{012} = \frac{2}{3}\mathbf{x}_{003} + \frac{1}{3}\mathbf{x}_{030},$$

where $\mathbf{x}_{003}$ and $\mathbf{x}_{030}$ are given data sites (relabeled here for convenience). The given tangent plane at $\mathbf{x}_{003}$ is of the form $z(x, y) = Z_{003} + xX_{003} + yY_{003}$, where $Z_{003}, X_{003}, Y_{003}$ are given position, x- and y-partials. Then we simply have to evaluate that plane at $\mathbf{a}$ and we have b_{012}:

$$b_{012} = z(\mathbf{a}).$$

The remaining Bézier ordinates are found in the same way. Note that this process is simply univariate cubic Hermite curve interpolation as outlined in Section 7.5!

The tenth coefficient, $b_{1,1,1}$, is not determined by the data. It is independent of the prescribed data and can be assigned any arbitrary value. A reasonable choice is to select $b_{1,1,1}$ such that *quadratic precision* of the interpolant is maintained, that is, if the nine prescribed data were read off from a quadratic, then the interpolant would reproduce this quadratic. If a quadratic is degree elevated to cubic, the coefficient $b_{1,1,1}$ may be expressed as

$$\begin{aligned} b_{1,1,1} = {} & \frac{1}{4}(b_{2,0,1} + b_{1,0,2} + b_{0,2,1} + b_{0,1,2} + b_{2,1,0} + b_{1,2,0}) \\ & - \frac{1}{6}(b_{3,0,0} + b_{0,3,0} + b_{0,0,3}). \end{aligned} \tag{18.4}$$

Thus choosing $b_{1,1,1}$ according to (18.4) ensures quadratic precision of the interpolant. P. Alfeld[4] has pointed out that other choices of $b_{1,1,1}$ also ensure

[4] Private communication, 1985.

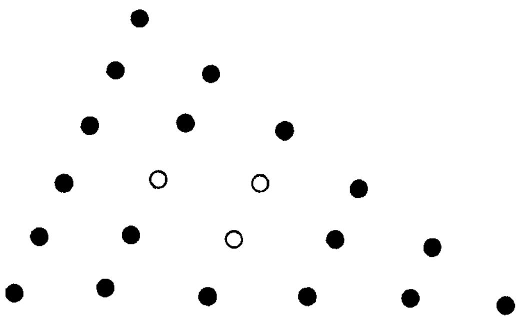


Figure 18.5 Quintic C^1 interpolants: the solid points are derived from the C^2 data at the vertices; the open ones have to be chosen in order to ensure C^1 continuity across patch boundaries.

quadratic precision; it is an open question if any of these choices can be sensibly labeled "best."

In a situation where data are prescribed at several triangles in a triangulation, the preceding interpolant has a serious drawback: it requires C^1 data, but the produced overall surface is only C^0. You are invited to construct an example!

If we insist on C^1 continuity, we could raise the degree from three to five. We then have to prescribe position, two first and three second derivatives at the vertices of each triangle. This fixes 18 of the quintic's 21 coefficients. The remaining three are used to ensure C^1 continuity between neighboring patches, in the same way as described later for the Clough–Tocher interpolant. This method for C^1 interpolation is described in detail in [29]; see Figure 18.5 for an illustration. Again, we have the drawback that second-order information has to be supplied, yet only first-order smoothness is obtained.

18.5 The Clough–Tocher Interpolant

This interpolant is conceptually the simplest of all so-called *split-triangle interpolants*, and it has been known in the finite element literature for some time; see Strang and Fix [582]. It is characterized by the "simplest" symmetric split of a triangle: each vertex is joined to the triangle's centroid; thus a macro-triangle[5] is split into three mini-triangles. Any other interior point other than the centroid would do; the centroid is chosen for symmetry reasons.

[5] The given triangles in a triangulation are referred to as macro-triangles in order to differentiate between them and the triangles resulting from the splitting process, the mini-triangles.

The first-order data that this interpolant requires are position and gradient value at the vertices of the macro-triangle plus some cross-boundary derivative at the midpoint of each edge. The prescribed cross-boundary derivative could be in any direction not parallel to its edge; but since adjacent macro-triangles should share the same data along the common edge, it is most natural to choose the direction perpendicular to that edge. We then speak of a *cross-boundary normal derivative.*

In summary, we have 12 data per macro-triangle. It is easily seen that interpolation to this data produces a globally C^1 surface if cubic polynomials are employed over each mini-triangle.

We shall now turn to the description of the actual interpolant; we refer to Figure 18.6. The Bézier ordinates of the three boundary curves (marked by full circles) are found exactly as for the nine-parameter interpolant. The next "layer" of ordinates, marked by full circles and diamonds, is determined if we enforce interpolation to the cross-boundary derivatives: the cross-boundary derivative, evaluated along an edge, is a univariate quadratic polynomial. It can be written as a univariate Bézier polynomial with three coefficients according to (17.19). The first and last of the three coefficients is determined by the gradients at the vertices, the center one as well by the cross-boundary derivative at the midpoints of that edge.

We are still left with the task of specifying the ordinates marked by open cirlces in Figure 18.6. Since the interpolant must be C^1 over each macro-triangle, those ordinates must satisfy the C^1 conditions. Thus each of the three outer ordinates of the four ones under consideration must be the average of the adjacent three ordinates that have already been determined. Finally, the center ordinate must be the average of the three just found.

In many applications, we will not be given the required cross-boundary derivatives at the edge midpoints. The most obvious method to estimate this derivative

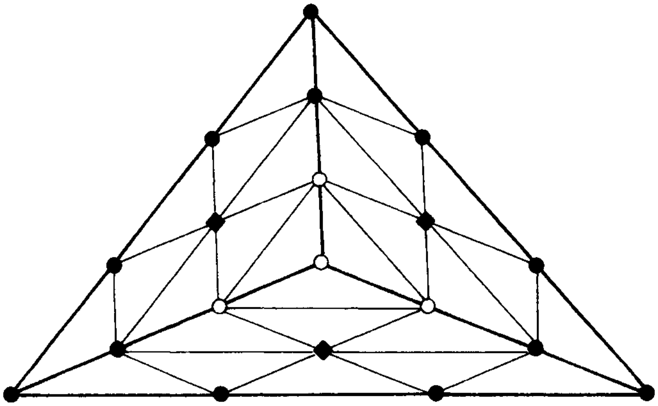


Figure 18.6 The Clough–Tocher interpolant: each macro-triangle is split into three mini-triangles.
]

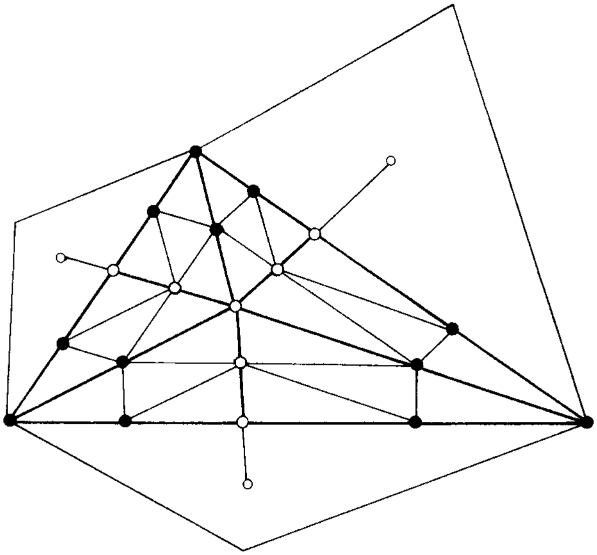


Figure 18.7 The Powell–Sabin interpolant: a macro-triangle is split into six mini-triangles.

is *condensation of parameters:* for each edge of the macro-triangle, the cross-boundary derivatives can be computed at its two endpoints. The midpoint cross-boundary derivative is simply set to be the average of those values. A more involved, but generally better, result is described in [194]. That method attempts to minimize the jump in the second cross-boundary derivative across two neighboring patches; see also [212] and [412].

We conclude this section with a somewhat surprising result (recall that the Clough–Tocher interpolant is designed to be C^1): the Clough–Tocher interpolant is C^2 at the centroid of the domain triangle, although it was designed to be just C^1! For a proof, consult [197].

18.6 The Powell–Sabin Interpolant

These interpolants produce C^1 piecewise quadratic interpolants to C^1 data at the vertices of a triangulated data set; see Powell and Sabin [490]. Each macro-triangle is split into six mini-triangles using the incenter as the interior split point[6]. Edges are split by joining incenters of neighboring triangles; see Figure 18.7. The Bézier ordinates of the quadratic mini-patches are determined in three steps as shown in Figure 18.7. Any two adjacent macro-triangles of this type will be

[6] The original Powell–Sabin interpolant was more involved.

differentiable across their common edge: by construction, each cross-boundary derivative is just one linear function instead of being piecewise linear.

The Powell–Sabin interpolant uses more triangles than does the Clough–Tocher method—but it is easier to *contour*. Each Powell–Sabin patch is a quadratic of the form $z = f(x, y)$, and a contour is of the form $c = f(x, y)$, that is, a conic. This conic may be written as a rational quadratic Bézier curve according to Chapter 12. Its Bézier points and weights may be determined by solving a number of quadratic equations; see [619].

18.7 Least Squares

Although we covered interpolation methods so far, approximation is a possibility as well. In that context, we would be given a triangle in the x, y-plane and a set of points $\mathbf{u}_1, \ldots, \mathbf{u}_P$ in the x, y-plane, expressed in terms of barycentric coordinates of the given triangle. Each of these points has a function value z_i associated with it. We now seek a triangular functional Bézier patch of a given degree n that approximates the given data:

$$z_k \approx \sum_{|\mathbf{i}|=n} b_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{u}_k). \tag{18.5}$$

In order to tackle this problem, we linearize this equation, similar to the approach of Section 15.6. With $L = (n+1)(n+2)/2$, a linearized version of (18.5) becomes

$$z_k \approx [\, B_{00n}^n(\mathbf{u}_k) \quad \ldots \quad B_{n00}^n(\mathbf{u}_k) \,] \begin{bmatrix} b_{00n} \\ \vdots \\ b_{n00} \end{bmatrix}. \tag{18.6}$$

There are several ways to construct the linearization; a simple one would be the following:

$$l(\mathrm{i}) = j(n+1) - \frac{1}{2} j(j-1) + i.$$

For the case $n = 2$, it would produce this ordering of indices:

$$(0,0,2), (1,0,1), (2,0,0), (0,1,1), (1,1,0), (0,2,0).$$

Combining all P equations (18.6), we obtain a linear system

$$\begin{bmatrix} B^n_{00n}(\mathbf{u}_1) & \dots & B^n_{n00}(\mathbf{u}_1) \\ \vdots & & \\ \vdots & & \\ B^n_{00n}(\mathbf{u}_L) & \dots & B^n_{n00}(\mathbf{u}_L) \end{bmatrix} \begin{bmatrix} b_{00n} \\ \vdots \\ b_{n00} \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ \vdots \\ z_L \end{bmatrix}. \tag{18.7}$$

We abbreviate it as

$$M\mathbf{b} = \mathbf{z},$$

noting that M has many more rows than columns. The optimal solution to our problem is obtained by solving the system of normal equations

$$M^{\mathrm{T}}M\mathbf{b} = M^{\mathrm{T}}\mathbf{z}. \tag{18.8}$$

It is also possible to formulate this type of least squares approximation for the parametric case. Then, the function values z_k are replaced by data points $\mathbf{p}_k$ with associated parameter values $\mathbf{u}_k$.

18.8 Problems

1 Suppose that in Figure 18.1 we prescribed $\mathbf{p}$ instead of the three $\mathbf{q}_i$. We could then find several sets of weights. Can that idea be generalized to higher degrees?

2 Discuss how we could add shape equations to the least squares problem of Section 18.7, in analogy to the approach taken in Section 15.6.

***3** The Clough–Tocher interpolant turns out to be C^2 at the centroid "for free." What can you say about the Powell–Sabin interpolant?

P1 Non-split interpolants of degrees higher than three or five may be defined for triangular patches. Experiment. Try to reproduce the Runge phenomenon.

P2 Program up a triangulation algorithm for a 2D point set.

P3 Write a routine for constructing the Powell–Sabin interpolant over a 2D triangulation.

This Page Intentionally Left Blank

19

W. Boehm

Differential Geometry II

19.1 Parametric Surfaces and Arc Element

A surface may be given by an *implicit* form $f(x, y, z) = 0$ or, more useful for CAGD, by its parametric form

$$\mathbf{x} = \mathbf{x}(u, v) = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}; \qquad \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \in [\mathbf{a}, \mathbf{b}] \subset \mathbb{R}^2, \tag{19.1}$$

where the cartesian coordinates x, y, z of a surface point are differentiable functions of the parameters u and v and $[\mathbf{a}, \mathbf{b}]$ denotes a rectangle in the u, v-plane; see Figure 19.1 (sometimes other domains are used, for example, triangles). To avoid potential problems with undefined normal vectors, we will assume

$$\mathbf{x}_u \wedge \mathbf{x}_v \neq 0 \text{ for } \mathbf{u} \in [\mathbf{a}, \mathbf{b}],$$

that is, that both families of isoparametric lines are regular (see Section 10.1) and are nowhere tangent to each other. Such a parametrization is called *regular*.[1]

Any change $\mathbf{r} = \mathbf{r}(\mathbf{u})$ of the parameters will not change the shape of the surface; the new parametrization is regular if $\det[\mathbf{r}_u, \mathbf{r}_v] \neq 0$ for $\mathbf{u} \in [\mathbf{a}, \mathbf{b}]$, that is, if one can find the inverse $\mathbf{u} = \mathbf{u}(\mathbf{r})$ of $\mathbf{r}$.

A regular curve $\mathbf{u} = \mathbf{u}(t)$ in the u, v-plane defines a regular curve $\mathbf{x}[\mathbf{u}(t)]$ on the surface. One can easily compute the (squared) arc element (see Section 10.1) of

[1] Examples of irregular parametrizations are shown in Figures 14.11, 14.12, and 14.13.

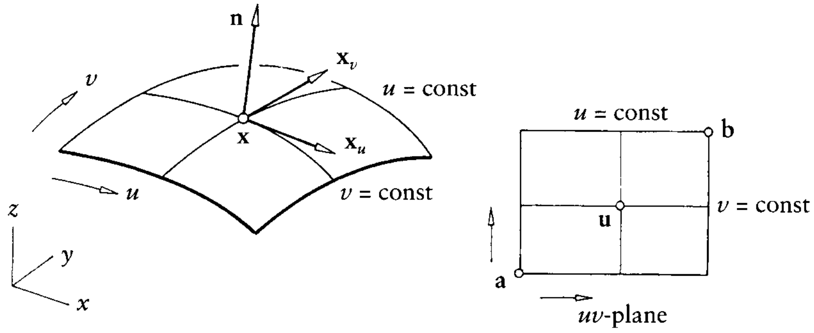


Figure 19.1 A parametric surface.

this curve: from $\dot{\mathbf{x}} = \mathbf{x}_u\dot{u} + \mathbf{x}_v\dot{v}$, we immediately obtain

$$\mathrm{d}s^2 = \|\dot{\mathbf{x}}\|^2\mathrm{d}t^2 = (\mathbf{x}_u^2\dot{u}^2 + 2\mathbf{x}_u\mathbf{x}_v\dot{u}\dot{v} + \mathbf{x}_v^2\dot{v}^2)\mathrm{d}t^2,$$

which will be written as

$$\mathrm{d}s^2 = E\mathrm{d}u^2 + 2F\mathrm{d}u\mathrm{d}v + G\mathrm{d}v^2, \tag{19.2}$$

where

$$E = E(u, v) = \mathbf{x}_u\mathbf{x}_u,$$
$$F = F(u, v) = \mathbf{x}_u\mathbf{x}_v,$$
$$G = G(u, v) = \mathbf{x}_v\mathbf{x}_v.$$

The squared arc element (19.2) is called the *first fundamental form* in classical differential geometry. It is of great importance for the further development of our material. Note that the arc element ds, being a geometric invariant of the curve through the point $\mathbf{x}$, does not depend on the particular parametrization chosen for the representation (19.1) of the surface.

For the arc length of the surface curve defined by $\mathbf{u} = \mathbf{u}(t)$, we obtain

$$\int_{t_0}^{t} \|\dot{\mathbf{x}}\|\mathrm{d}t = \int_{t_0}^{t} \sqrt{E\dot{u}^2 + 2F\dot{u}\dot{v} + G\dot{v}^2}\mathrm{d}t.$$

Remark 1 The *area element* corresponding to the element dudv of the u, v-plane is given by

$$\mathrm{d}A = \|\mathbf{x}_u\mathrm{d}u \wedge \mathbf{x}_v\mathrm{d}v\| = \|\mathbf{x}_u \wedge \mathbf{x}_v\|\mathrm{d}u\mathrm{d}v;$$

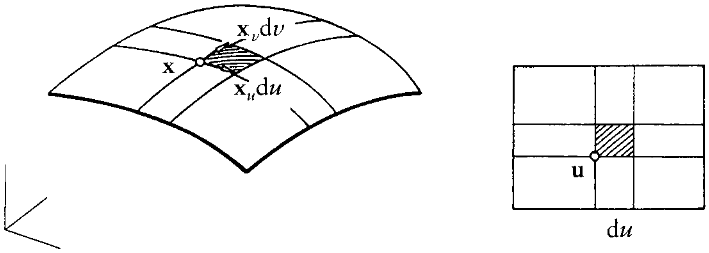


Figure 19.2 Area element.

see Figure 19.2. From $\|\mathbf{a} \wedge \mathbf{b}\|^2 = \mathbf{a}^2\mathbf{b}^2 - (\mathbf{ab})^2$, we obtain

$$D = \|\mathbf{x}_u \wedge \mathbf{x}_v\| = \sqrt{EG - F^2}. \tag{19.3}$$

The quantity D is called the *discriminant* of (19.2). Thus the surface area A corresponding to a region U of the u, v-plane is given by

$$A = \int\int_U \sqrt{EG - F^2} \mathrm{d}u\mathrm{d}v.$$

Remark 2 If $F = 0$ at a point of the surface, the two isoparametric lines that meet there are orthogonal to each other. Moreover, if $F \equiv 0$ at every point of the surface, the net of isoparametric lines is orthogonal everywhere.

Remark 3 Note that for any real[2] $\mathrm{d}u, \mathrm{d}v$, the first fundamental form $\mathrm{d}s^2$ is strictly positive. However, if $\mathrm{d}s^2 = 0$, we have two *imaginary* directions. These are called *isotropic directions* at $\mathbf{x}$.

Remark 4 Let $\mathbf{u}_1 = \mathbf{u}_1(t_1)$ and $\mathbf{u}_2 = \mathbf{u}_2(t_2)$ define two surface curves, intersecting at $\mathbf{x}$. Both curves are intersecting orthogonally if the *polar form* of $\dot{\mathbf{x}}^2$, given by

$$\dot{\mathbf{x}}_1\dot{\mathbf{x}}_2 = E\dot{u}_1\dot{u}_2 + F(\dot{u}_1\dot{v}_2 + \dot{u}_2\dot{v}_1) + G\dot{v}_1\dot{v}_2,$$

vanishes at $\mathbf{x}$.

[2] Note that the vector $[\mathrm{d}u, \mathrm{d}v]$ defines a direction at a point $\mathbf{x}$.

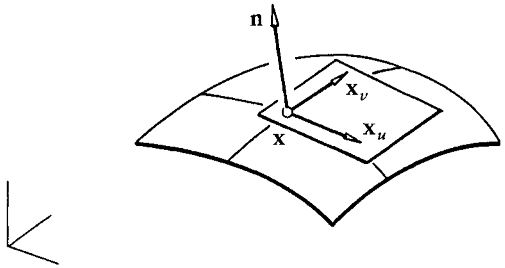


Figure 19.3 The local frame and the tangent plane.

19.2 The Local Frame

The partials x_u and x_v at a point x span the tangent plane to the surface at x. Let y be any point on this plane. Then

$$\det[y - x, x_u, x_v] = 0$$

is the implicit equation of the tangent plane. The parametric equation is

$$y(u, v) = x + \Delta u x_u + \Delta v x_v.$$

The normal $x_u \wedge x_v$ of the tangent plane coincides with the normal to the surface at x. The normalized normal

$$n = \frac{x_u \wedge x_v}{\|x_u \wedge x_v\|} = \frac{1}{D}[x_u \wedge x_v]$$

together with the unnormalized vectors x_u, x_v form a local coordinate system, a *frame*, at x (see Figure 19.3). This frame plays the same important role for surfaces as does the Frenet frame (see Section 10.2) for curves. The normal is of unit length and is perpendicular to x_u and x_v, that is, $n^2 = 1$ and $nx_u = nx_v = 0$. In general, the local coordinate system with origin x and axes x_u, x_v forms only an affine system; it is also (unlike the Frenet frame) dependent on the parametrization (19.1).

19.3 The Curvature of a Surface Curve

Let $u(t)$ define a curve on the surface $x(u)$. From curve theory we know that its curvature $\kappa = \frac{1}{\rho}$ is defined by $t' = \kappa m$; the prime denotes differentiation with

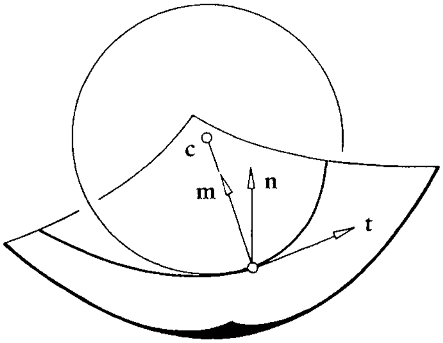


Figure 19.4 Osculating circle.

respect to the arc length of the curve. We will now reformulate this expression in surface terms. Since $\mathbf{t} = \mathbf{x}'$ and $u' = \mathrm{d}u/\mathrm{d}s$, $v' = \mathrm{d}v/\mathrm{d}s$, we have

$$\mathbf{t}' = \mathbf{x}'' = \mathbf{x}_{uu}(u')^2 + 2\mathbf{x}_{uv}u'v' + \mathbf{x}_{vv}(v')^2 + \mathbf{x}_u u'' + \mathbf{x}_v v''.$$

Let ϕ be the angle between the main normal $\mathbf{m}$ of the curve and the surface normal $\mathbf{n}$ at the point $\mathbf{x}$ under consideration, as illustrated in Figure 19.4. Then

$$\mathbf{t}'\mathbf{n} = \kappa\mathbf{m}\mathbf{n} = \kappa \cos\phi.$$

Inserting $\mathbf{t}'$ and keeping in mind that $\mathbf{n}\mathbf{x}_u = \mathbf{n}\mathbf{x}_v = 0$, we have

$$\kappa \cos\phi = \mathbf{n}\mathbf{x}_{uu}(u')^2 + 2\mathbf{n}\mathbf{x}_{uv}u'v' + \mathbf{n}\mathbf{x}_{vv}(v')^2. \tag{19.4}$$

Furthermore, $\mathbf{n}\mathbf{x}_u = 0$ implies $\mathbf{n}_u\mathbf{x}_u + \mathbf{n}\mathbf{x}_{uu} = 0$, and so on. Thus, using the abbreviations

$$\begin{aligned} L = L(u,v) &= -\mathbf{x}_u\mathbf{n}_u &&= \mathbf{n}\mathbf{x}_{uu}, \\ M = M(u,v) &= -\tfrac{1}{2}(\mathbf{x}_u\mathbf{n}_v + \mathbf{x}_v\mathbf{n}_u) &&= \mathbf{n}\mathbf{x}_{uv}, \\ N = N(u,v) &= -\mathbf{x}_v\mathbf{n}_v &&= \mathbf{n}\mathbf{x}_{vv}, \end{aligned} \tag{19.5}$$

Eq. (19.4) can be written as

$$\kappa \cos\phi \, \mathrm{d}s^2 = L\mathrm{d}u^2 + 2M\mathrm{d}u\mathrm{d}v + N\mathrm{d}v^2. \tag{19.6}$$

This expression is called the *second fundamental form* in classical differential geometry. For any given direction $\mathrm{d}u/\mathrm{d}v$ in the u, v-plane and any given angle ϕ, the second fundamental form, together with the first fundamental form (19.2), allows us to compute the curvature κ of a surface curve having that tangent direction.

Remark 5 Note that the arc length in the preceding development was only used in a theoretical context; for applications, it does not have to be actually computed.

Remark 6 Note that κ depends only on the tangent direction and the angle ϕ. It will change its sign, however, if there is a change in the orientation of $\mathbf{n}$.

19.4 Meusnier's Theorem

The right-hand side of (19.4) does not contain terms involving ϕ. For $\phi = 0$, that is, $\cos\phi = 1$, we have that $\mathbf{m} = \mathbf{n}$: the osculating plane of the curve is perpendicular to the surface tangent plane at $\mathbf{x}$. The curvature κ_0 of such a curve is called the *normal curvature* of the surface at $\mathbf{x}$ in the direction of $\mathbf{t}$ (defined by du/dv). The normal curvature is given by

$$\kappa_0 = \kappa_0(\mathbf{x}; \mathbf{t}) = \frac{1}{\rho_0} = \frac{\text{2nd fundamental form}}{\text{1st fundamental form}}. \tag{19.7}$$

Now (19.6) takes the very short form

$$\rho = \rho_0 \cos\phi. \tag{19.8}$$

This simple formula has an interesting and important interpretation, known as Meusnier's theorem. It is illustrated in Figure 19.5: the osculating circles of all surface curves through $\mathbf{x}$ having the same tangent $\mathbf{t}$ there form a sphere. This sphere and the surface have a common tangent plane at $\mathbf{x}$; the radius of the sphere is ρ_0.

As a consequence of Meusnier's theorem, it is sufficient to study curves at $\mathbf{x}$ with $\mathbf{m} = \mathbf{n}$; moreover, these curves may be planar. Such curves, called *normal sections*, can be thought of as the intersection of the surface with a plane through $\mathbf{x}$ and containing $\mathbf{n}$, as illustrated in Figure 19.6.

Remark 7 If the direction of the normal is chosen as in Figure 19.5, we have $0 \le \rho \le \rho_0$, and $\rho = 0$ only if $\phi = \pi/2$, that is, if the osculating plane O coincides with the tangent plane.

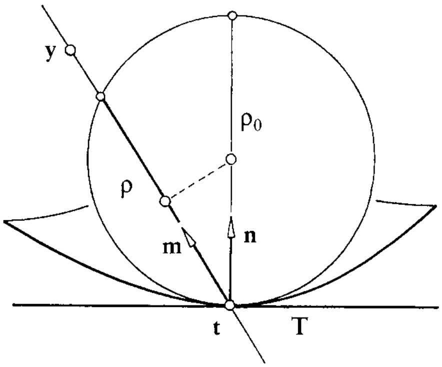


Figure 19.5 Meusnier's sphere viewed in the direction of **t**.

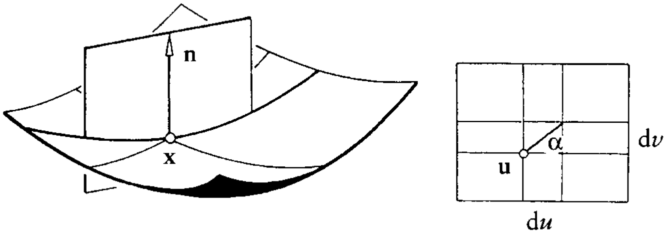


Figure 19.6 Normal section of a surface.

19.5 Lines of Curvature

For Meusnier's theorem, we considered (osculating) planes that contained a fixed tangent at a point on a surface; we will now look at (osculating) planes containing the normal vector at a fixed point **x**. We will drop the subscript of κ_0 to simplify the notation.

Setting $\lambda = \mathrm{d}v/\mathrm{d}u = \tan\alpha$ (see Figure 19.6), we can rewrite (19.7) as

$$\kappa(\mathbf{x}, \mathbf{t}) = \kappa(\lambda) = \frac{L + 2M\lambda + N\lambda^2}{E + 2F\lambda + G\lambda^2}.$$

In the special case where $L : M : N = E : F : G$, the normal curvature κ is independent of λ. Points **x** with that property are called *umbilical points.*

In the general case, where κ changes as λ changes, $\kappa = \kappa(\lambda)$ is a rational quadratic function, as illustrated in Figure 19.7. The extreme values κ_1 and κ_2 of $\kappa(\lambda)$ occur at the roots λ_1 and λ_2 of

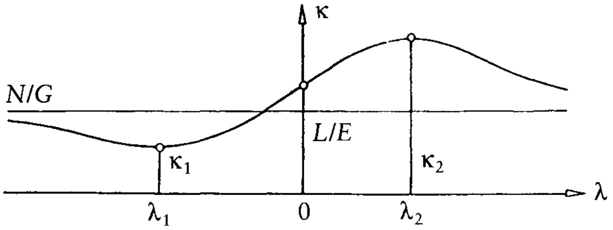


Figure 19.7 The function $\kappa = \kappa(\lambda)$.

$$\det \begin{bmatrix} \lambda^2 & -\lambda & 1 \\ E & F & G \\ L & M & N \end{bmatrix} = 0. \tag{19.9}$$

It can be shown that λ_1 and λ_2 are always real. The extreme values κ_1 and κ_2 are the roots of

$$\det \begin{bmatrix} \kappa E - L & \kappa F - M \\ \kappa F - M & \kappa G - N \end{bmatrix} = 0. \tag{19.10}$$

The quantities λ_1 and λ_2 define directions in the u, v-plane; the corresponding directions in the tangent plane are called *principal directions*. The net of lines that have these directions at all of their points is called the net of *lines of curvature*. If necessary, it may be constructed by integrating (19.9).

Therefore, this net of lines of curvature can be used as a parametrization of the surface; then (19.9) must be satisfied by $\mathrm{d}u = 0$ and by $\mathrm{d}v = 0$. This implies, excluding umbilical points, that

$$F \equiv 0 \quad \text{and} \quad M \equiv 0.$$

The first equation, $F \equiv 0$, states that lines of curvature are orthogonal to each other; the second equation states that they are conjugate to each other as defined in Section 19.9.

At an umbilical point, the principal directions are undefined; see also Remark 9.

Remark 8 For a surface of revolution, the net of lines of curvature is defined by the meridians and the parallels; an example is shown in Figure 19.8.

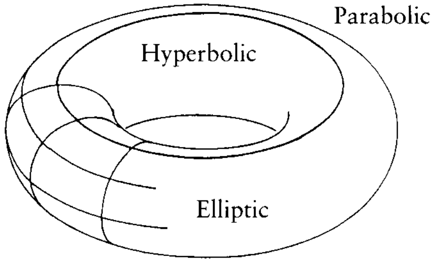


Figure 19.8 Lines of curvature on a torus. Also shown are the regions of elliptic, parabolic, and hyperbolic points.

19.6 Gaussian and Mean Curvature

The extreme values κ_1 and κ_2 of $\kappa = \kappa(\lambda)$ are called *principal curvatures* of the surface at $\mathbf{x}$. A comparison of (19.10) with $\kappa^2 - (\kappa_1 + \kappa_2)\kappa + \kappa_1\kappa_2 = 0$ yields

$$\kappa_1\kappa_2 = \frac{LN - M^2}{EG - F^2} \tag{19.11}$$

and

$$\kappa_1 + \kappa_2 = \frac{NE - 2MF + LG}{EG - F^2}. \tag{19.12}$$

The term $K = \kappa_1\kappa_2$ is called *Gaussian curvature*, whereas $H = \frac{1}{2}(\kappa_1 + \kappa_2)$ is called *mean curvature*. Note that both κ_1 and κ_2 change sign if the normal $\mathbf{n}$ is reversed, but K is not affected by such a reversal.

If κ_1 and κ_2 are of the same sign, that is, if $K > 0$, the point $\mathbf{x}$ under consideration is called *elliptic*. For example, all points of an ellipsoid are elliptic points. If κ_1 and κ_2 have different signs, that is, $K < 0$, the point $\mathbf{x}$ under consideration is called *hyperbolic*. For example, all points of a hyperboloid are hyperbolic points. Finally, if either $\kappa_1 = 0$ or $\kappa_2 = 0$, K vanishes, the point $\mathbf{x}$ under consideration is called *parabolic*. For example, all points of a cylinder are parabolic points. In the special case where both K and H vanish, one has a *flat* point.

The Gaussian curvature K depends on the coefficients of the first and second fundamental forms. It is a very important result, due to Gauss, that K can also be expressed only in terms of E, F, and G and their derivatives. This is known as the *Theorema Egregium*, which states that K depends only on the *intrinsic geometry* of the surface. This means it does not change if the surface is deformed in a way that does not change length measurement within it.

Remark 9 All points of a sphere are umbilic. The Gaussian and mean curvatures of a sphere are constant.

Remark 10 Any *developable*, that is, a surface that can be deformed to planar shape without changing length measurements in it, must have $K \equiv 0$. Conversely, every surface with $K \equiv 0$ can be developed into a plane (if necessary, by applying cuts). See also Section 19.10.

19.7 Euler's Theorem

The normal curvatures in different directions **t** at a point **x** are not independent of each other. For simplicity, let the isoparametric curves of a surface be lines of curvature; then we have $F \equiv M \equiv 0$ (see Section 19.5). As a consequence, we have

$$\kappa_1 = \frac{L}{E} \quad \text{and} \quad \kappa_2 = \frac{N}{G},$$

and $\kappa(\lambda)$ may be written as

$$\kappa(\lambda) = \frac{L + N\lambda^2}{E + G\lambda^2} = \kappa_1 \frac{E}{E + G\lambda^2} + \kappa_2 \frac{G\lambda^2}{E + G\lambda^2}. \tag{19.13}$$

The coefficients of κ_1 and κ_2 have a nice geometric meaning: let Ψ denote the angle between $\mathbf{x}_u$ and the tangent vector $\dot{\mathbf{x}} = \mathbf{x}_u\dot{u} + \mathbf{x}_v\dot{v}$ of the curve under consideration, as illustrated in Figure 19.9. We obtain

$$\cos\Psi = \frac{\dot{\mathbf{x}}\mathbf{x}_u}{\|\dot{\mathbf{x}}\|\,\|\mathbf{x}_u\|} = \frac{\sqrt{E}}{\sqrt{E + G\lambda^2}}$$

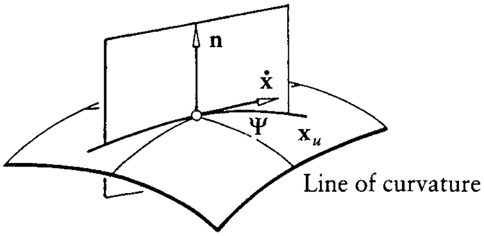


Figure 19.9 Configuration for Euler's theorem.

and

$$\sin \Psi = \frac{\dot{\mathbf{x}}\mathbf{x}_v}{\|\dot{\mathbf{x}}\| \, \|\mathbf{x}_v\|} = \frac{\sqrt{G}\lambda}{\sqrt{E + G\lambda^2}},$$

where $\lambda = \dot{v}/\dot{u}$ as before. Hence $\kappa(\lambda)$ may be written as

$$\kappa(\lambda) = \kappa_1 \cos^2 \Psi + \kappa_2 \sin^2 \Psi.$$

This important result was found by L. Euler.

19.8 Dupin's Indicatrix

Euler's theorem has the following geometric implication. If we introduce polar coordinates $r = \sqrt{\rho}$ and Ψ for a point $\mathbf{y}$ of the tangent plane at $\mathbf{x}$ by setting $y_1 = \sqrt{\rho}\cos\Psi$, $y_2 = \sqrt{\rho}\sin\Psi$, then setting $\kappa_1 = \frac{1}{\rho_1}$ and $\kappa_2 = \frac{1}{\rho_2}$, Euler's theorem can be written

$$\frac{y_1^2}{\rho_1} + \frac{y_2^2}{\rho_2} = 1.$$

This is the equation of a conic section, the *Dupin's indicatrix* (see Figure 19.10). Its points $\mathbf{y}$ in the direction given by Ψ have distance $\sqrt{\rho}$ from $\mathbf{x}$. Taking into account that a reversal of the direction of $\mathbf{n}$ will effect a change in the sign of ρ, this conic section is an ellipse if $K > 0$, a pair of hyperbolas if $K < 0$ (corresponding to $\sqrt{\rho}$ and $\sqrt{-\rho}$), and a pair of parallel lines if $K = 0$ (but $H \neq 0$).

Dupin's indicatrix has a nice geometric interpretation: we may approximate our surface at $\mathbf{x}$ by a paraboloid, that is, a Taylor expansion with terms up to second order. Then Remark 7 of Chapter 10 leads to a very simple interpretation of Dupin's indicatrix: the indicatrix, scaled down by a factor of $1 : m$, can be viewed as the intersection of the surface with a plane parallel to the tangent plane at $\mathbf{x}$ in the distance $\epsilon = \frac{1}{2m^2}$. This is illustrated in Figure 19.11.

Remark 11 This illustrates the appearance of a pair of hyperbolas in Figure 19.10; they appear when intersecting the surface in distances $\epsilon = \pm\frac{1}{2m^2}$. We can thus assign a *sign* to the Dupin indicatrix, depending on its being "above" or "below" the tangent plane.

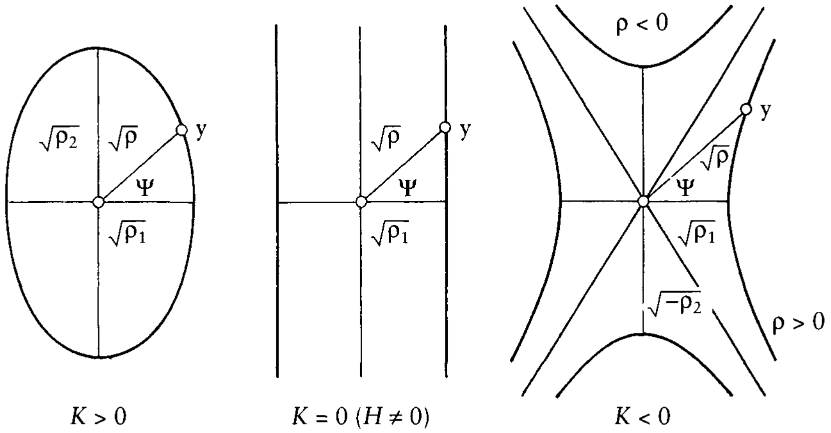


Figure 19.10 Dupin's indicatrix for an elliptic, a parabolic, and a hyperbolic point.

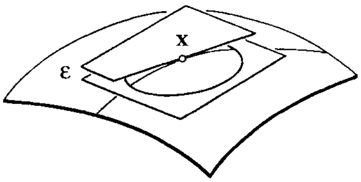


Figure 19.11 Dupin's indicatrix, scale $1:m$.

19.9 Asymptotic Lines and Conjugate Directions

The asymptotic directions of Dupin's indicatrix have a simple geometric meaning: surface curves passing through $\mathbf{x}$ and having a tangent in an asymptotic direction there have zero curvature at $\mathbf{x}$; in other words, these directions are defined by

$$L\mathrm{d}u^2 + 2M\mathrm{d}u\mathrm{d}v + N\mathrm{d}v^2 = 0. \tag{19.14}$$

They are real and different if $K < 0$, real but coalescing if $K = 0$, and complex if $K > 0$.

The net of lines having these directions in all of their points is called the *net of asymptotic lines*. If necessary, it may be calculated by integrating (19.14). In a hyperbolic region of the surface, it is real and regular and can be used for a real parametrization.

For this parametrization, one has

$$L \equiv 0 \quad \text{and} \quad N \equiv 0,$$

and vice versa.

As earlier, let y be a point on Dupin's indicatrix at a point **x**. Let $\dot{y}$ denote its tangent direction at y. The direction $\dot{y}$ is called *conjugate* to the direction $\dot{x}$ from x to y. Consider two surface curves $\mathbf{u}_1(t_1)$ and $\mathbf{u}_2(t_2)$ that have tangent directions $\dot{x}_1$ and $\dot{x}_2$ at **x**. Some elementary calculations yield that $\dot{x}_1$ is conjugate to $\dot{x}_2$ if

$$L\dot{u}_1\dot{u}_2 + M(\dot{u}_1\dot{v}_2 + \dot{u}_2\dot{v}_1) + N\dot{v}_1\dot{v}_2 = 0.$$

Note that this expression is symmetric in $\dot{\mathbf{u}}_1, \dot{\mathbf{u}}_2$. By definition asymptotic directions are *self-conjugate.*

Remark 12 Isoparametric curves of a surface are conjugate if $M \equiv 0$ and vice versa.

Remark 13 The principal directions, defined by (19.9), are orthogonal and conjugate; they bisect the angles between the asymptotic directions; that is, they are the axis directions of Dupin's indicatrix (see Figure 19.10).

Remark 14 The tangent planes of two "consecutive" points on a surface curve intersect in a straight line s. Let the curve have direction **t** at a point **x** on the surface. Then s and **t** are conjugate to each other. In particular, if **t** is an asymptotic direction, s coincides with **t**. If **t** is one of the principal directions at **x**, then s is orthogonal to **t** and vice versa. These properties characterize lines of curvature and asymptotic lines and may be used to define them geometrically.

19.10 Ruled Surfaces and Developables

If a surface contains a family of straight lines, it is called a *ruled surface.* It is convenient to use these straight lines as one family of isoparametric lines. Then the ruled surface may be written

$$\mathbf{x} = \mathbf{x}(t, v) = \mathbf{p}(t) + v\mathbf{q}(t), \tag{19.15}$$

where **p** is a point and **q** is a vector, both depending on t. The isoparametric lines $t = const$ are called the *generatrices* of the surface; see Figure 19.12.

The partials of a ruled surface are given by $\mathbf{x}_t = \dot{\mathbf{p}} + v\dot{\mathbf{q}}$ and $\mathbf{x}_v = \mathbf{q}$. The normal **n** at **x** is given by

$$\mathbf{n} = \frac{(\dot{\mathbf{p}} + v\dot{\mathbf{q}}) \wedge \mathbf{q}}{\|(\dot{\mathbf{p}} + v\dot{\mathbf{q}}) \wedge \mathbf{q}\|}.$$

A point y on the tangent plane at **x** satisfies

$$\det[\mathbf{y} - \mathbf{p}, \dot{\mathbf{p}}, \mathbf{q}] + v\det[\mathbf{y} - \mathbf{p}, \dot{\mathbf{q}}, \mathbf{q}] = 0;$$

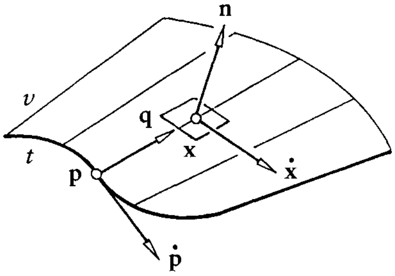


Figure 19.12 Ruled surface.

in other words, the tangent planes along a generatrix form a pencil of planes. However, if $\dot{\mathbf{p}}$, $\dot{\mathbf{q}}$, and $\mathbf{q}$ are linearly dependent, that is, if

$$\det[\dot{\mathbf{p}}, \dot{\mathbf{q}}, \mathbf{q}] = 0, \tag{19.16}$$

the tangent plane does not vary with v.

If (19.16) holds for all t, the tangent planes are fixed along each generatrix; hence all tangent planes of the surface form a one-parameter family of planes. Conversely, any one-parameter family of planes envelopes a developable surface that may be written as a ruled surface (19.15), satisfying condition (19.16); see also Remark 10.

Remark 15 The generatrices of any ruled surface coalesce with one family of its asymptotic lines. As a consequence of asymptotic lines being real, we have $K \leq 0$.

Remark 16 In particular, the generatrices of a developable surface agree with its coalescing asymptotic lines, also forming one family of its lines of curvature. The second family of lines of curvature is formed by the orthogonal trajectories of the generatrices.

Remark 17 It can be shown that any developable surface is a cone $\mathbf{p} = \mathbf{const}$, a cylinder $\mathbf{q} = \mathbf{const}$, or a surface formed by all tangents of a space curve, that is, $\mathbf{q} = \dot{\mathbf{p}}$; see Figure 19.13.

Remark 18 The normals along a line of curvature of any surface form a developable surface. This property characterizes and defines lines of curvature.

Remark 19 The tangent planes along a curve $\mathbf{x} = \mathbf{x}[\mathbf{u}(t)]$ on any surface form a developable surface. It may be developed into a plane; if by this process the curve $\mathbf{x}[\mathbf{u}(t)]$

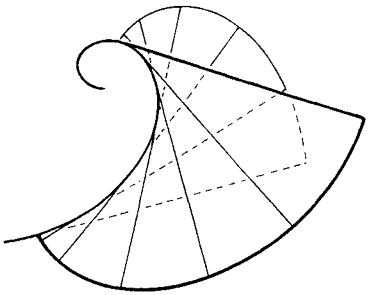


Figure 19.13 General developable surface.

happens to be developed into a straight line, the curve $\mathbf{x}[\mathbf{u}(t)]$ is called a *geodesic*. At any point $\mathbf{x}$ of a geodesic, we find that

$$\det[\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \mathbf{n}] = 0. \tag{19.17}$$

Equation (19.17) is the differential equation of a geodesic; it is of second order. Geodesics may also be characterized as providing the shortest path between two points on the surface.

19.11 Nonparametric Surfaces

Let $z = f(x, y)$ be a function of two variables as shown in Figure 19.14. The surface

$$\mathbf{x} = \mathbf{x}(u, v) = \begin{bmatrix} u \\ v \\ z(u, v) \end{bmatrix}$$

is then called a nonparametric surface. From this, we immediately find

$$E = 1 + z_u^2, \qquad F = z_u z_v, \qquad G = 1 + z_v^2,$$

$$D^2 = EG - F^2 = 1 + z_u^2 + z_v^2,$$

$$\mathbf{n} = \frac{1}{D}\begin{bmatrix} -z_u \\ -z_v \\ 1 \end{bmatrix},$$

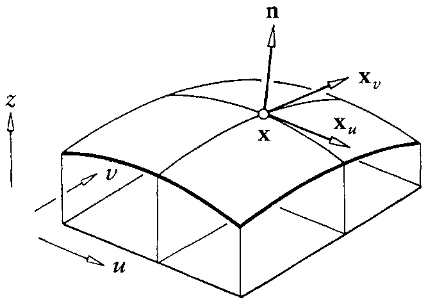


Figure 19.14 Nonparametric surface.

$$L = \frac{1}{D} z_{uu}, \qquad M = \frac{1}{D} z_{uv}, \qquad N = \frac{1}{D} z_{vv},$$

$$K = \frac{1}{D^2}(z_{uu} z_{vv} - z_{uv}^2).$$

19.12 Composite Surfaces

A surface $\mathbf{x} = \mathbf{x}(u, t)$ with global parameters u and t may be composed of patches or segments of different surfaces. Let $\mathbf{x}_- = \mathbf{x}_-(t)$ denote the right boundary curve and $\mathbf{x}_+ = \mathbf{x}_+(t)$ the left boundary curve of two such patches, connected along $\mathbf{x} = \mathbf{x}(t)$, $t \in [a, b]$, as illustrated in Figure 19.15. Both patches are tangent plane continuous if $\mathbf{n}_- = \pm \mathbf{n}_+$ at all $\mathbf{x}(t)$. This may also be written

$$\alpha \mathbf{x}_{-u} = \beta \mathbf{x}_{+v} + \gamma \dot{\mathbf{x}}, \tag{19.18}$$

where α, β, γ are functions of t and the product $\alpha\beta$ is nonvanishing.

The two patches are curvature continuous if they are tangent plane continuous and both Dupin's indicatrices agree along the common boundary, in the sense of Remark 11 and as illustrated in Figure 19.16.

If the common boundary is C^1, both indicatrices have a pair of points in common that are opposed to each other in the direction of the boundary tangent. Although a conic section is defined by five points (see Section 12.5), or by its midpoint and three nondiametrical points, it can be shown that both Dupin's indicatrices coincide if there exists a family of curvature continuous surface curves crossing the common boundary. In particular, this family may be one of asymptotic lines (Pegna and Wolter [459]) or any family of isolines (Boehm [74])

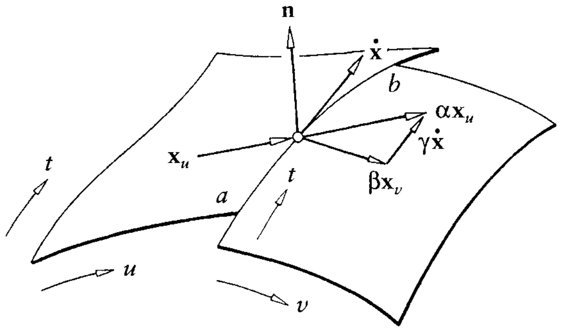


Figure 19.15 Composite surface.

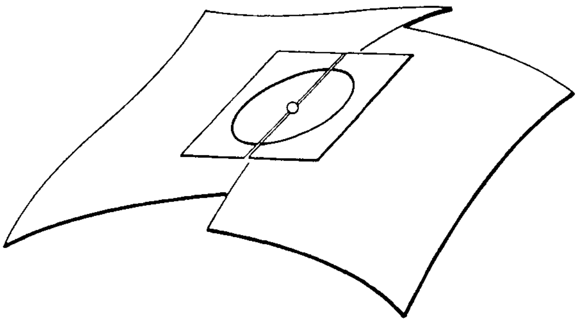


Figure 19.16 Common Dupin's indicatrix.

or even any family of unordered directions (Pegna and Wolter [459]). Moreover, if the boundary is C^0 only at a point, there are two directions of curvature continuity there, and so no further conditions have to be met.

Remark 20 A surface is called tangent or curvature continuous if any plane section is tangent or curvature continuous, respectively.

Remark 21 Note that the asymptotic directions of both patches may coincide even if they are imaginary.

Remark 22 A consequence of (19.18) is the following:

$$\alpha\mathbf{x}_{-ut} = \beta\mathbf{x}_{+ut} + \gamma\ddot{\mathbf{x}} - \dot{\alpha}\mathbf{x}_{-u} + \dot{\beta}\mathbf{x}_{+v} + \dot{\gamma}\dot{\mathbf{x}}.$$

Remark 23 Note that, although Dupin's indicatrix is a euclidean invariant only, curvature continuity of surfaces is an affinely and projectively invariant property.

This Page Intentionally Left Blank

Geometric Continuity for Surfaces

20

The concept of geometric continuity is not restricted to curves. Surfaces in this regard are much more complex to deal with, so we will restrict ourselves to the case of first-order geometric continuity. Here are some pointers to the literature: Boehm [77], Charrot and Gregory [110], DeRose [161], Farin [190], Gregory [291], Hahn [306], Herron [322], Liu and Hoschek [395], Kahmann [350], Kiciak [356], Jensen [343], Jones [349], Nielson [447], Piper [483], Sarraga [536], [537], Shirman and Séquin [573], van Wijk [594], Vinacua and Brunet [603], and Veron et al. [600].

20.1 Introduction

Let us take a look at Figure 20.1. It shows the (potential) boundary curves of two cubic triangular Bézier patches. In each, the interior control point $\mathbf{b}_{111}$ is missing. Can we determine these two missing points such that the resulting two patches form a C^1 surface? We must thus produce two control nets that satisfy the C^1 condition as illustrated in Figure 17.11. But this is not possible in our example: the two shaded pairs of triangles in Figure 20.1 are not affine pairs in the sense of Section 17.6, that is, they cannot be obtained as an affine map of *one* pair of domain triangles.

We have a better chance of solving the problem if we relax the requirement of C^1 continuity to that of G^1 continuity: *two patches with a common boundary curve are called G^1 if they have a continuously varying tangent plane along that boundary curve.* The concept of G^1 continuity is a genuine generalization of C^1 continuity: all (nondegenerate) C^1 surfaces are G^1, but not the other way around.

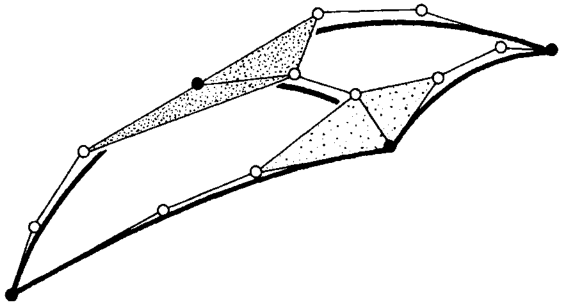


Figure 20.1 G^1 continuity: the shown cubic curves cannot be the boundary curves of two C^1 cubic Bézier triangles since no suitable pair of domain triangles can be found.

An example (if somewhat simplistic) of a G^1 yet non-C^1 surface is easily constructed: take two triangles formed by the diagonal of a square in the x, y-plane and interpret them as two linear Bézier triangular patches. They are clearly G^1, but they are not C^1 if we pick as their domain the two adjacent triangles with vertices $(0,0), (1,0), (0,1)$ and $(0,0), (1,0), (-1,0)$.

One important aspect of G^1 continuity is that it is completely independent of the domains of the two involved surface patches. For C^1 continuity, the interplay between range and domain geometry was crucial, but now the domains are only needed so that we can evaluate each patch.

We will next discuss the different configurations of G^1 continuity between triangular and rectangular patches.

20.2 Triangle-Triangle

In this section, we shall construct a (sufficient) condition for two adjacent triangular Bézier patches to be G^1. We only have to consider the control polygon of the common boundary curve and the two "parallel" rows of control points in each patch. The situation is illustrated in Figure 20.2, where some suitable abbreviations are introduced.

Let $\mathbf{x}(t)$ be a point on the common boundary curve of the two patches. It may be constructed using the de Casteljau algorithm from either patch since the de Casteljau algorithm yields the tangent plane at a point (see Section 17.4), being spanned by the $\mathbf{b}_{\mathrm{i}}^{n-1}$. These points are labeled $\mathbf{p}(t), \mathbf{q}^0(t), \mathbf{q}^n(t)$, and $\mathbf{r}(t)$ in Figure 20.2. The two patches are G^1 if these four points are coplanar for all t,

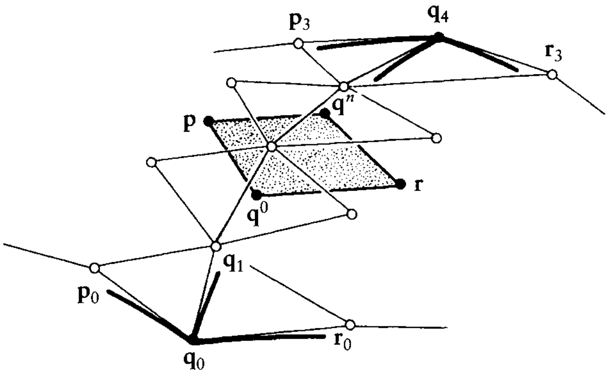


Figure 20.2 G^1 continuity: the G^1 constraints may be expressed in terms of the de Casteljau algorithm. The quartic case is shown.

that is, if the lines $\overline{\mathbf{pr}}$ and $\overline{\mathbf{q}^0\mathbf{q}^n}$ always intersect.[1] This means that functions $\lambda(t)$ and $\mu(t)$ exist such that

$$(1-\lambda(t))\mathbf{p}(t)+\lambda(t)\mathbf{r}(t)=(1-\mu(t))\mathbf{q}^0(t)+\mu(t)\mathbf{q}^n(t) \tag{20.1}$$

for all t.

We can write explicit expressions for $\mathbf{p}(t)$, $\mathbf{q}^0(t)$, $\mathbf{q}^n(t)$, and $\mathbf{r}(t)$:

$$\mathbf{p}(t)=\sum_{i=0}^{n-1}\mathbf{p}_iB_i^{n-1}(t),$$

$$\mathbf{q}^0(t)=\sum_{i=0}^{n-1}\mathbf{q}_iB_i^{n-1}(t),$$

$$\mathbf{q}^n(t)=\sum_{i=0}^{n-1}\mathbf{q}_{i+1}B_i^{n-1}(t),$$

$$\mathbf{r}(t)=\sum_{i=0}^{n-1}\mathbf{r}_iB_i^{n-1}(t).$$

[1] To be precise, we must also require that $\mathbf{p}$ and $\mathbf{r}$ be on *different sides* of $\mathbf{q}^n$ and $\mathbf{q}^0$; otherwise, we might generate surfaces that double back on themselves along the common boundary curve.

We know that the first and last pair of triangles must be coplanar. This means that numbers $\lambda_0, \lambda_1, \mu_0, \mu_1$ exist such that (with the definition $\bar{x} = 1 - x$)

$$\bar{\lambda}_0\mathbf{p}_0 + \lambda_0\mathbf{r}_0 = \bar{\mu}_0\mathbf{q}_0 + \mu_0\mathbf{q}_1. \tag{20.2}$$

and

$$\bar{\lambda}_1\mathbf{p}_{n-1} + \lambda_1\mathbf{r}_{n-1} = \bar{\mu}_1\mathbf{q}_{n-1} + \mu_1\mathbf{q}_n. \tag{20.3}$$

The numbers λ_0 and μ_0 describe the geometry of the first pair of triangles, whereas λ_1 and μ_1 describe that of the last pair.

We now attempt to simplify our task of formulating G^1 conditions: we make the assumption that $\lambda(t)$ and $\mu(t)$ are both *linear*, that is,

$$\lambda(t) = (1-t)\lambda_0 + t\lambda_1, \quad \mu(t) = (1-t)\mu_0 + t\mu_1.$$

We also observe that

$$1 - \lambda(t) = (1-t)\bar{\lambda}_0 + t\bar{\lambda}_1, \quad 1 - \mu(t) = (1-t)\bar{\mu}_0 + t\bar{\mu}_1.$$

The G^1 condition (20.1) then becomes

$$\begin{aligned}((1-t)\bar{\lambda}_0 + t\bar{\lambda}_1)\sum_{i=0}^{n-1}\mathbf{p}_iB_i^{n-1}(t) + ((1-t)\lambda_0 + t\lambda_1)\sum_{i=0}^{n-1}\mathbf{r}_iB_i^{n-1}(t)\\ = ((1-t)\bar{\mu}_0 + t\bar{\mu}_1)\sum_{i=0}^{n-1}\mathbf{q}_iB_i^{n-1}(t) + ((1-t)\mu_0 + t\mu_1)\sum_{i=0}^{n-1}\mathbf{q}_{i+1}B_i^{n-1}(t).\end{aligned}$$

Using (5.32) and (5.33) and regrouping, we obtain

$$\begin{aligned}\sum_{i=0}^{n}\frac{n-i}{n}(\bar{\lambda}_0\mathbf{p}_i + \lambda_0\mathbf{r}_i)B_i^n(t) + \sum_{i=0}^{n}\frac{i+1}{n}(\bar{\lambda}_1\mathbf{p}_i + \lambda_1\mathbf{r}_i)B_{i+1}^n(t)\\ = \sum_{i=0}^{n}\frac{n-i}{n}(\bar{\mu}_0\mathbf{q}_i + \mu_0\mathbf{q}_{i+1})B_i^n(t) + \sum_{i=0}^{n}\frac{i+1}{n}(\bar{\mu}_1\mathbf{q}_i + \mu_1\mathbf{q}_{i+1})B_{i+1}^n(t).\end{aligned}$$

After an index transformation involving the B_{i+1}^n terms, we may compare coefficients for $i = 0, \ldots, n$:

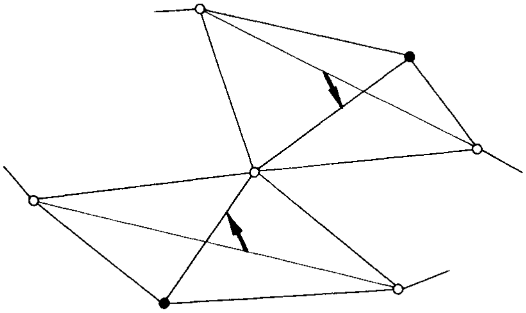


Figure 20.3 G^1 continuity: for two quadratics, the shape difference vectors $\mathbf{h}_1^2$ and $\mathbf{h}_1^0$ are shown. Note that they sum to 0.

$$\begin{aligned}&\frac{i}{n}(\bar\lambda_1\mathbf{p}_{i-1}+\lambda_1\mathbf{r}_{i-1})+\frac{n-i}{n}(\bar\lambda_0\mathbf{p}_i+\lambda_0\mathbf{r}_i)\\ &\quad=\frac{i}{n}(\bar\mu_1\mathbf{q}_{i-1}+\mu_1\mathbf{q}_i)+\frac{n-i}{n}(\bar\mu_0\mathbf{q}_i+\mu_0\mathbf{q}_{i+1}).\end{aligned}\tag{20.4}$$

As a sanity check, we see that for $i=0$ and $i=n$, we recover conditions (20.2) and (20.3).

It seems natural to define points

$$\mathbf{h}_i^0=\bar\lambda_0\mathbf{p}_i+\lambda_0\mathbf{r}_i,\qquad \mathbf{h}_i^n=\bar\lambda_1\mathbf{p}_i+\lambda_1\mathbf{r}_i$$

as well as

$$\mathbf{v}_i^0=\bar\mu_0\mathbf{q}_i+\mu_0\mathbf{q}_{i+1},\qquad \mathbf{v}_i^n=\bar\mu_1\mathbf{q}_i+\mu_1\mathbf{q}_{i+1}.$$

Then (20.4) takes on the simple form

$$\frac{i}{n}\mathbf{h}_{i-1}^0+\frac{n-i}{n}\mathbf{h}_i^n=\frac{i}{n}\mathbf{v}_{i-1}^0+\frac{n-i}{n}\mathbf{v}_i^n.\tag{20.5}$$

The quadratic case is illustrated in Figure 20.3. This case is of special interest: our G^1 condition is only sufficient, not necessary, in general. However, for quadratics, it is both sufficient *and* necessary.[2]

[2] This was observed by T. DeRose (1989), private communication.

20.3 Rectangle-Rectangle

We now consider two tensor product Bézier patches with a common boundary curve of degree n, illustrated in Figure 20.4. Consider any point $\mathbf{x}(t)$ on this curve. It may be constructed using the (univariate) de Casteljau algorithm. The de Casteljau algorithm yields the tangent of a curve as a by-product, namely, as the difference of the two intermediate points $\mathbf{b}_1^{n-1}$ and $\mathbf{b}_0^{n-1}$. In Figure 20.4, those two points are labeled $\mathbf{q}_b$ and $\mathbf{q}_t$. It now follows that the tangent plane of the left patch in Figure 20.4 is spanned by the points $\mathbf{p}, \mathbf{q}_b, \mathbf{q}_t$ and that of the right patch is spanned by $\mathbf{q}_b, \mathbf{q}_t, \mathbf{r}$, where

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

and

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

We could now follow a similar development, as in the previous section, but a little trick will give us our desired G^1 condition easily. Let us simply degree elevate the common boundary curve from degree n to $n+1$. Call the degree elevated control points $\hat{\mathbf{q}}_0, \ldots, \hat{\mathbf{q}}_{n+1}$ (see Section 6.1 for the degree elevation procedure). Now we are in the situation of the previous section! Namely, we have $n+1$ control points

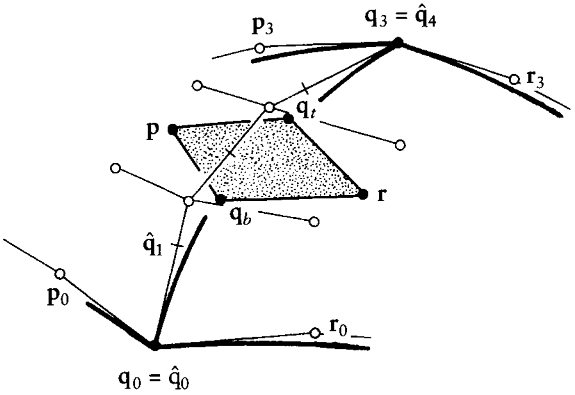


Figure 20.4 G^1 continuity for rectangular patches: the shaded quadrilateral must be planar as it varies along the common boundary. The case of a cubic boundary curve is shown.

$\mathbf{p}_i$, $n+2$ control points $\hat{\mathbf{q}}_i$, and $n+1$ control points $\mathbf{r}_i$. Our situation is equivalent to that of a G^1 join between two triangular patches of degree $n+1$.

The desired G^1 condition is therefore given by

$$\begin{aligned}&\frac{i}{n+1}\left[[\bar{\lambda}_1\mathbf{p}_{i-1}+\lambda_1\mathbf{r}_{i-1}]-[\bar{\mu}_1\hat{\mathbf{q}}_{i-1}+\mu_1\hat{\mathbf{q}}_i]\right]\\ &\quad=-\left(1-\frac{i}{n+1}\right)\left[[\bar{\lambda}_0\mathbf{p}_i+\lambda_0\mathbf{r}_i]-[\bar{\mu}_0\hat{\mathbf{q}}_i+\mu_0\hat{\mathbf{q}}_{i+1}]\right],\end{aligned} \tag{20.6}$$

where i varies from 0 to $n+1$.

The geometric interpretation is analogous to that of the preceding section.

20.4 Rectangle-Triangle

This situation, illustrated in Figure 20.5, is now treated in a completely analogous way. We assume that both patches have a common boundary curve of degree n. If we degree elevate the triangular patch (Section 17.7), we have the control point rows $\hat{\mathbf{p}}_0, \ldots, \hat{\mathbf{p}}_n$ (from the tensor product patch), $\hat{\mathbf{q}}_0, \ldots, \hat{\mathbf{q}}_{n+1}$ (the degree elevated common boundary curve), and $\mathbf{r}_0, \ldots, \mathbf{r}_n$ (from the degree elevated triangular patch). Thus the G^1 condition is

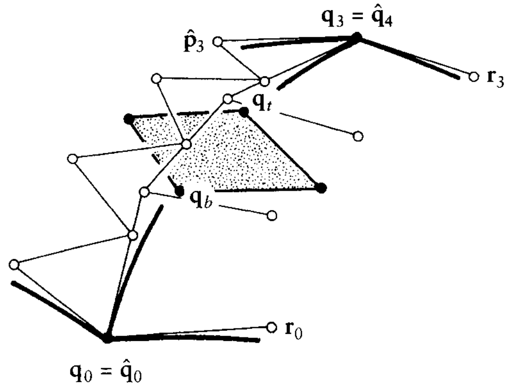


Figure 20.5 G^1 continuity for triangular and rectangular patches: the shaded quadrilateral must be planar as it varies along the common boundary. The case of a cubic boundary curve is shown.

$$\begin{aligned}\frac{i}{n+1}&\Big[[\bar{\lambda}_1\hat{\mathbf{p}}_{i-1}+\lambda_1\mathbf{r}_{i-1}]-[\bar{\mu}_1\hat{\mathbf{q}}_{i-1}+\mu_1\hat{\mathbf{q}}_i]\Big]\\ &=-\left(1-\frac{i}{n+1}\right)\Big[[\bar{\lambda}_0\hat{\mathbf{p}}_i+\lambda_0\mathbf{r}_i]-[\bar{\mu}_0\hat{\mathbf{q}}_i+\mu_0\hat{\mathbf{q}}_{i+1}]\Big],\end{aligned} \tag{20.7}$$

again with i ranging from 0 to $n+1$.

20.5 "Filling in" Rectangular Patches

Suppose that we were given the boundary curves of two bicubic patches, as shown in Figure 20.4. At the endpoints of the common boundary curve, we assume that the three curves meeting there have coplanar tangents. Can we find interior Bézier points $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{r}_1, \mathbf{r}_2$ such that the two patches will have a continuously varying tangent plane along the common boundary? We shall employ the G^1 condition (20.6) for this purpose. As a first step, we determine α_0 and β_0 from $\mathbf{p}_0, \hat{\mathbf{q}}_0, \mathbf{r}_0, \hat{\mathbf{q}}_1$ and α_1, β_1 from $\mathbf{p}_3, \hat{\mathbf{q}}_3, \mathbf{r}_3, \hat{\mathbf{q}}_4$.

There are three equations in (20.6) that involve our four unknowns $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{r}_1, \mathbf{r}_2$. After some suitable modification, they are of the form

$$\begin{aligned}\alpha_0\mathbf{p}_1+(1-\alpha_0)\mathbf{r}_1&=\mathbf{rhs1},\\ \alpha_1\mathbf{p}_1+(1-\alpha_1)\mathbf{r}_1+\alpha_0\mathbf{p}_2+(1-\alpha_0)\mathbf{r}_2&=\mathbf{rhs2},\\ \alpha_1\mathbf{p}_2+(1-\alpha_1)\mathbf{r}_2&=\mathbf{rhs3}.\end{aligned}$$

We have abbreviated the right-hand sides of the equations somewhat.

These may be written in matrix form:

$$A\mathbf{x}=\mathbf{b}, \tag{20.8}$$

with

$$A=\begin{bmatrix}\alpha_0 & 1-\alpha_0 & & \\ \alpha_1 & 1-\alpha_1 & \alpha_0 & 1-\alpha_0\\ & & \alpha_1 & (1-\alpha_1)\end{bmatrix}$$

$\mathbf{x}=[\mathbf{p}_1,\mathbf{p}_2,\mathbf{r}_1,\mathbf{r}_2]^{\mathrm{T}}$, and $\mathbf{b}=[\mathbf{rhs1},\mathbf{rhs2},\mathbf{rhs3}]^{\mathrm{T}}$. Since the rows of A are linearly independent, the underdetermined system (20.8) always has solutions. One way of finding one is as follows: suppose we already have an estimate $\bar{\mathbf{x}}=[\bar{\mathbf{p}}_1,\bar{\mathbf{p}}_2,\bar{\mathbf{r}}_1,\bar{\mathbf{r}}_2]$ for the unknowns. Then one solves the system

$$AA^{\mathrm{T}}\mathbf{y}=\mathbf{b}-A\bar{\mathbf{x}}$$

for y, and the solution is given by

$$\mathbf{x} = \bar{\mathbf{x}} + A^{\mathrm{T}}\mathbf{y}.$$

This solution stays as close as possible to the initial guess $\bar{\mathbf{x}}$; see [86].

The $\bar{\mathbf{r}}_i$ and $\bar{\mathbf{p}}_i$ could, for instance, be generated by Adini's twists (see Section 16.3). This idea was carried out by Sarraga [536], [537] in a slightly different context.

20.6 "Filling in" Triangular Patches

Let us reconsider Figure 20.1. Do our G^1 conditions allow us to complete the cubic curve network such that the resulting surface will be G^1? In our notation, the undetermined points are $\mathbf{p}_1$ and $\mathbf{r}_1$. There are two G^1 equations that involve these points. They are of the form

$$\alpha_1\mathbf{p}_1 + (1-\alpha_1)\mathbf{r}_1 = \mathbf{rhs1},$$

$$\alpha_0\mathbf{p}_1 + (1-\alpha_0)\mathbf{r}_1 = \mathbf{rhs2}.$$

The coefficient matrix of this system:

$$A = \begin{bmatrix} \alpha_1 & 1-\alpha_1 \\ \alpha_0 & 1-\alpha_0 \end{bmatrix}$$

is singular if $\alpha_0 = \alpha_1$. Therefore, in this case, we are not guaranteed to have a solution (see Piper [483] for an explicit counterexample).

We can, however, solve the problem if we resort to quartics. After we degree elevate all boundary curves, we now have to determine unknown control points $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{r}_1, \mathbf{r}_2$. This is exactly the situation from the preceding section and is solved in exactly the same way. B. Piper [483] first used quartics to solve this kind of Hermite interpolation problem.

20.7 Theoretical Aspects

We have developed an approach to geometric continuity that is powerful enough to solve several applications-oriented problems. It is practical, but it is not *general*: there are G^1 surfaces that do not satisfy the condition (20.4); see Problems. T. DeRose has developed conditions that are both necessary and sufficient for G^1 continuity of adjacent Bézier patches.

Several authors (consult the surveys by Boehm [77], Herron [322], and Gregory [291]) define geometric continuity of surfaces in the following way: two surfaces that share a common boundary curve are called G^r if, for every point of the boundary curve, a reparametrization exists such that both surfaces are C^r in a neighborhood of that point. For the case $r = 1$, this definition yields tangent plane continuity. Its advantage is that it also works for higher orders; the price to be paid is that it is rather abstract. For the case of G^2 continuity, another popular definition is to require a common Dupin's indicatrix along the boundary curve (see Section 19.12, Kahmann [350], and Pegna and Wolter [459]).

20.8 **Problems**

1 In Section 20.1 we saw an example of triangular patch boundaries that could not be completed to construct an overall C^1 surface. Find similar examples for tensor product patches.

2 Show that the G^1 conditions (20.4) include the case of strict C^1 continuity (17.23) between triangular patches.

3 Construct surfaces that are G^1 yet do not satisfy (20.4).

4 Consider eight triangular patches, assembled so as to form eight octants of a spherelike surface. Show that this closed surface cannot be C^1, that is, one cannot find a region in the plane that comprises eight triangles that have a C^1 map that maps them to the surface.

P1 Construct a spherelike G^1 surface that is made up of six biquartic patches having a cubelike connectivity.

Surfaces with Arbitrary Topology 21

The surfaces that we have met so far are best suited for shapes that are the image of some part of the plane—of a rectangle in the case of B-spline or Bézier surfaces, of a triangulated region in the case of composite Bézier triangles. This limits the *topology* of these surfaces; for example, it is not possible to construct even a sphere without introducing degenerate patches while using a C^1 map of a part of the plane. Even shapes that have the topology of a planar region may be too complex to model with one tensor product surface; just imagine modeling a glove that way. The complexity issue may be tackled using the approach of hierarchical B-spline surfaces as proposed by Forsey and Bartels [245]. But even with this method, arbitrary topology is not achievable.

In this chapter, we will investigate methods that are suited for the construction of shapes of arbitrary complexity and/or topology. We can present only a brief selection of methods; more literature on the topic: [177], [596], [595], [594], [289], [290], [293], [321], [431], [627], [398], [561]. The first in-depth study of recursive subdivision processes goes back to U. Reif [503]. The basic concepts of surface topology are nicely explained in [323].

21.1 Recursive Subdivision Curves

We already encountered Chaikin's algorithm in Section 8.4. Starting with a polygon, it produces a sequence of refined polygons that ultimately converge to the uniform quadratic B-spline curve defined by the initial polygon.

This principle may be applied to higher-degree uniform splines; we discuss the cubic case here. If we double a uniform knot sequence by inserting the midpoint

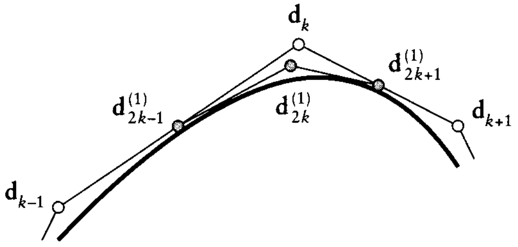


Figure 21.1 Subdivision curves: one step of cubic B-spline curve subdivision.

of every knot, new control points $\mathbf{d}^1_i$ are generated from the existing ones by setting

$$\begin{aligned}\mathbf{d}^{(1)}_{2i} &= \frac{1}{8}\mathbf{d}_{i-1} + \frac{3}{4}\mathbf{d}_i + \frac{1}{8}\mathbf{d}_{i+1}\\ \mathbf{d}^{(1)}_{2i+1} &= \frac{1}{2}\mathbf{d}_i + \frac{1}{2}\mathbf{d}_{i+1}\end{aligned} \tag{21.1}$$

See Figure 21.1 for an illustration. We may analyze the resulting curve by exploiting the fact that we are dealing with a cubic B-spline curve. However, that analysis may also be carried out without this knowledge. Let us investigate the limit of the sequence $\mathbf{d}_k, \mathbf{d}^{(1)}_{2k}, \mathbf{d}^{(2)}_{4k}, \ldots$. We may write the recursion in matrix form as

$$\begin{bmatrix}\mathbf{d}^{(1)}_{2k-1}\\ \mathbf{d}^{(1)}_{2k}\\ \mathbf{d}^{(1)}_{2k+1}\end{bmatrix} = \frac{1}{8}\begin{bmatrix}4 & 4 & 0\\ 1 & 6 & 1\\ 0 & 4 & 4\end{bmatrix}\begin{bmatrix}\mathbf{d}_{k-1}\\ \mathbf{d}_k\\ \mathbf{d}_{k+1}\end{bmatrix}$$

and abbreviate it as

$$\mathrm{D}^{(1)} = A\mathrm{D}. \tag{21.2}$$

The matrix A has eigenvalues $1, \frac{1}{2}, \frac{1}{4}$ and may be diagonalized[1] as

$$A = E\Lambda E^{-1},$$

where

[1] The process of diagonalization is a standard linear algebra procedure.

$$E = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & -1 \\ 1 & -1 & 2 \end{bmatrix}, \qquad E^{-1} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 \\ 3 & 0 & -3 \\ 1 & -2 & 1 \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}.$$

The matrix E contains A's eigenvectors, and the diagonal matrix Λ contains A's eigenvalues.

For the next iteration, we have

$$\mathrm{D}^{(2)} = A\mathrm{D}^{(1)} = A^2\mathrm{D}^{(1)}. \tag{21.3}$$

Using our diagonalization for A, we get

$$A^2 = E\Lambda E^{-1}E^{-1}\Lambda E = E\Lambda^2 E^{-1}$$

and further

$$A^r = E\Lambda^r E^{-1}.$$

Taking the limit $r \to \infty$ yields

$$A^\infty = E\Lambda^\infty E^{-1}.$$

Since

$$\Lambda^\infty = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

this becomes

$$A^\infty = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 \\ 1 & 4 & 1 \\ 1 & 4 & 1 \end{bmatrix},$$

implying that all three points $\mathbf{d}_{k-1}, \mathbf{d}_k, \mathbf{d}_{k+1}$ converge to the same point $(\mathbf{d}_{k-1} + 4\mathbf{d}_k + \mathbf{d}_{k+1})/6$. This result is no surprise since we know we are dealing with B-spline curves. It does illustrate, however, a general principle that is ubiquitous in all subdivision theory, namely, that convergence analysis is normally carried out via an eigenvalue analysis.

Subdivision may also be used to generate *interpolating* curves. The basic *four-point* principle was developed by Dyn, Levin, and Gregory [178], [176]. Let a

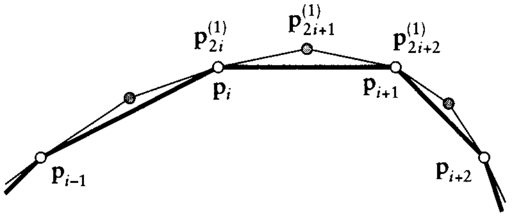


Figure 21.2 Subdivision curves: one step of the four-point scheme.

sequence of points $\mathbf{p}_i$ be given; then successively construct new sequences by setting

$$\begin{aligned} \mathbf{p}_{2i}^{(1)} &= \mathbf{p}_i, \\ \mathbf{p}_{2i+1}^{(1)} &= \frac{1}{16}[-\mathbf{p}_{i-1} + 9\mathbf{p}_i + 9\mathbf{p}_{i+1} - \mathbf{p}_{i+2}]. \end{aligned} \tag{21.4}$$

For an illustration, see Figure 21.2. The point $\mathbf{p}_{2i+1}^{(1)}$ is the result of applying cubic Lagrange interpolation at uniform knots, see [519].

At each level of the subdivision process, the points of the previous step are retained; this causes the scheme to interpolate to the initial set of points. The limit curve is C^1, but fails to be C^2.

21.2 Doo–Sabin Surfaces

The fundamental idea of this kind of surface goes back to Chaikin's algorithm; see Section 8.4. There, we started with a polygon, iteratively applied a refinement procedure to it, and observed that in the limit we ended up with a smooth curve. M. Sabin and D. Doo asked if this principle could be carried over to surfaces: start with a polyhedron, iteratively apply a refinement procedure to it, and see if a smooth surface would result.

They then came up with the following algorithm, illustrated in Fig. 21.3 and documented in [174]: input: an arbitrary closed polyhedron with vertices $\mathbf{p}_i$. These vertices form (straight) edges and (not necessarily planar) faces, thus defining the topology of the polyhedron. The refinement step now becomes:

1. For each face, compute new vertices $\mathbf{v}_i^{(1)}$ by averaging the vertices $\mathbf{v}_i$ of the face as follows:

$$\mathbf{v}_i^{(1)} = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j \tag{21.5}$$

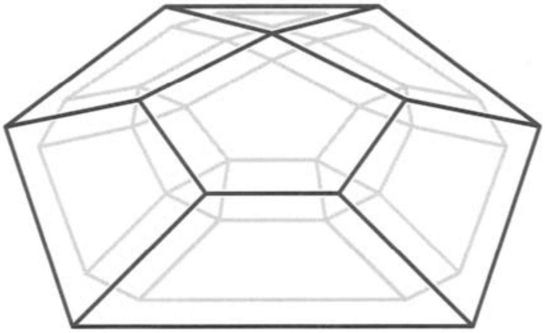


Figure 21.3 The Doo–Sabin algorithm: a new polyhedron is constructed by a refinement procedure.

where the α_{ij} are given by

$$\begin{aligned} \alpha_{ii} &= \frac{n+5}{4n} \\ \alpha_{ij} &= \frac{3+2\cos\frac{2\pi(i-j)}{n}}{4n}. \end{aligned} \tag{21.6}$$

2. Construct a new polyhedron from these new vertices.

Step 2 needs some more explanation. The new polyhedron will have faces that are constructed according to three different rules:

2a. The *F-faces* are found by cyclically connecting the $V_i^{(1)}$ of each original face.

2b. The *E-faces* are found by considering any two original faces sharing a common edge: there are exactly four midpoints on the lines connecting each face centroid with the edge endpoints. These four points produce a four-sided face.

2c. The *V-faces* are formed by considering all E-faces around an original vertex. They surround a face that is "centered" on that vertex.

As we keep repeating the algorithm, it produces mostly four-sided faces. The only non–four-sided faces are V-faces generated by those initial vertices whose valency[2] is not four, or F-faces whose initial faces are not four sided. These faces give rise to limit points whose valency is not four, so-called *extraordinary vertices*. In fact, it is not hard to see that after the first step the valency of every new vertex is four. In this manner, large regions of the new polyhedra are covered with nets that have a tensor product structure. Doo–Sabin surfaces are thus "mostly" biquadratic B-splines. See Figure 21.4 for an example.

[2] The valency of a vertex is the number of edges emanating from it.

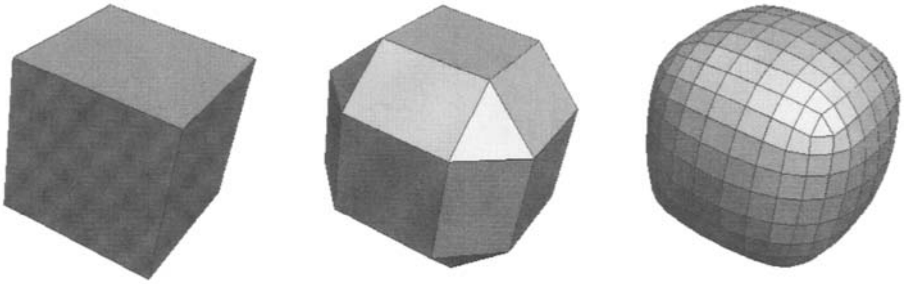


Figure 21.4 The Doo–Sabin algorithm: an example. Left, the original mesh (a cube). Middle, after one iteration. Right, after three iterations. Figure courtesy A. Nasri.

Let us briefly analyze the structure of the Doo–Sabin algorithm. New face points $\mathbf{v}_i^{(1)}$ are created from previous ones in a linear fashion that lends itself to a matrix formulation. Let us consider a five-sided face such as the center face in Figure 21.3. We obtain

$$\begin{bmatrix} \mathbf{v}_1^{(1)} \\ \mathbf{v}_2^{(1)} \\ \mathbf{v}_3^{(1)} \\ \mathbf{v}_4^{(1)} \\ \mathbf{v}_5^{(1)} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} & \alpha_{25} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} & \alpha_{35} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} & \alpha_{45} \\ \alpha_{51} & \alpha_{52} & \alpha_{53} & \alpha_{54} & \alpha_{55} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \\ \mathbf{v}_5 \end{bmatrix} \tag{21.7}$$

We may write this as

$$\mathbf{V}^{(1)} = A_5\mathbf{V},$$

which becomes

$$\mathbf{V}^{(r+1)} = A_5^r\mathbf{V}$$

after r applications of the algorithm. In the general case, we have the same structure but have to replace A_5 by a matrix A_n for faces with n edges.

As r increases, the behavior of the subdivision process depends crucially on the eigenvalues of A_n since we may write A_n^r as

$$A_n^r = E_n\Lambda_n^r E_n^{-1}$$

following the approach of Section 21.1. Since all rows of A_n sum to unity, one eigenvalue is $\lambda_1 = 1$. The remaining ones are all real since A is symmetric and are all between 0 and 1 since each $\mathrm{V}^{(k)}$ is contained in the convex hull of $\mathrm{V}^{(k-1)}$. An exact analysis of this process is fairly involved and is omitted here. It reveals, however, that Doo–Sabin surfaces are G^1 everywhere. For more details, see [174] or [18], [19], [503]. The matrices A_n are *circulant*, and their eigenstructure has been studied in detail by P. Davis [134]. A matrix is circulant if each row may be obtained from the previous row by a "right shift."

Figure 21.4 gives an example of several steps of the algorithm.

As a rather trivial observation, Doo–Sabin surfaces have the convex hull and local control properties, and their construction is affinely invariant. But they also do not need an underlying parametrization, which makes them more geometric than tensor product B-spline surfaces. A drawback of this nice feature is the problem of point evaluation: though we can evaluate as many points as close to the surface as we like, computation of just one point is not trivial.

One set of points on the surface is easy to identify: at every level of subdivision, the centroid of any face will be on the final surface. For a proof, we observe that any face will produce a sequence of F-faces converging to its centroid. In the limit, the centroid is thus on the surface.

21.3 Catmull–Clark Subdivision

The same issue of *Computer-Aided Design* that included the Doo–Sabin algorithm also contained a competing method, invented by E. Catmull and J. Clark; see [103]. Whereas Doo–Sabin surfaces are a generalization of biquadratic B-splines to arbitrary topology, Catmull–Clark surfaces generalize bicubic B-spline surfaces to arbitrary topology.

We start with a polygonal mesh $\mathcal{M}^0$ consisting of vertices $\mathbf{v}_k^0$. We iteratively refine the mesh, resulting in finer and finer meshes $\mathcal{M}^i$, consisting of vertices $\mathbf{v}_k^i$. Each refinement step can be described by explaining what happens locally for the points adjacent to a vertex $\mathbf{v}^i$:

1. Form face points $\mathbf{f}_j^{i+1}$: for each face in the mesh, find the centroid of its vertices.
2. Form edge points $\mathbf{e}_j^{i+1}$: for each edge in the mesh, average the edge's endpoints and the two face points on either side of the edge.
3. Form a new vertex point $\mathbf{v}^{i+1}$: assuming there are n faces around $\mathbf{v}^i$, it is computed as follows:

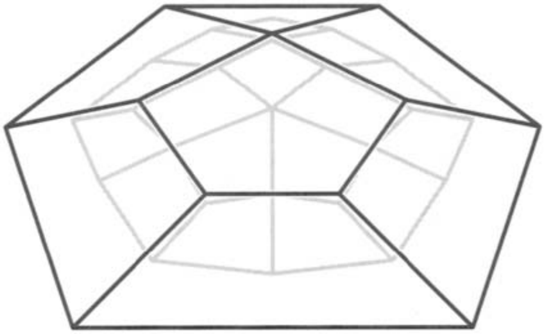


Figure 21.5 The Catmull–Clark algorithm: the original control net (black) and one level of subdivision (gray).

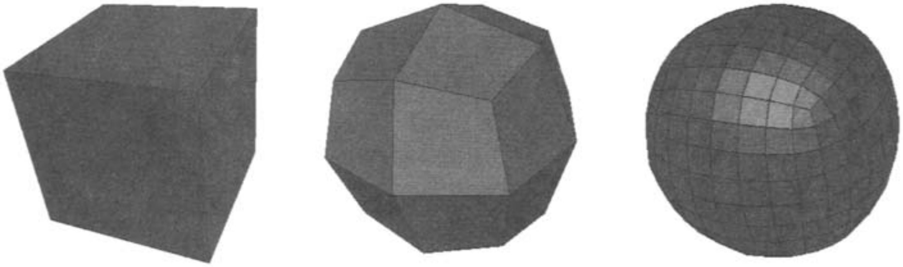


Figure 21.6 The Catmull–Clark algorithm: an example. Left, the original mesh (a cube). Middle, after one iteration. Right, after three iterations. Figure courtesy A. Nasri.

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \frac{1}{n}\sum_{j=1}^{n}(\mathbf{e}_j^i - \mathbf{v}^i) + \frac{1}{n}\sum_{j=1}^{n}(\mathbf{f}_j^{i+1} - \mathbf{v}^i). \tag{21.8}$$

4. Form new faces. Each new face consists of a loop of the form

$$\mathbf{f}^{i+1} - \mathbf{e}^{i+1} - \mathbf{v}^{i+1} - \mathbf{e}^{i+1} - \mathbf{f}^{i+1},$$

where the two $\mathbf{f}^{i+1}$'s refer to the same face point.

Note that all faces after the first level will be four sided. The basic principle is shown in Figure 21.5. Figure 21.6 gives an example.

For the example of a vertex with valency $n = 4$, the relationship between new and old vertices may be expressed as a matrix equation:

$$\begin{bmatrix} \mathbf{v}^{i+1} \\ \mathbf{e}_1^{i+1} \\ \mathbf{e}_2^{i+1} \\ \mathbf{e}_4^{i+1} \\ \mathbf{e}_4^{i+1} \\ \mathbf{v}_1^{i+1} \\ \mathbf{v}_2^{i+1} \\ \mathbf{v}_3^{i+1} \\ \mathbf{v}_4^{i+1} \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 9 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 6 & 6 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 6 & 1 & 6 & 1 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 6 & 1 & 0 & 1 & 1 & 0 \\ 6 & 1 & 0 & 1 & 6 & 0 & 0 & 1 & 1 \\ 4 & 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 & 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 4 & 4 & 0 & 0 & 4 & 0 \\ 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} \mathbf{v}^{i} \\ \mathbf{e}_1^{i} \\ \mathbf{e}_2^{i} \\ \mathbf{e}_4^{i} \\ \mathbf{e}_4^{i} \\ \mathbf{v}_1^{i} \\ \mathbf{v}_2^{i} \\ \mathbf{v}_3^{i} \\ \mathbf{v}_4^{i} \end{bmatrix}.$$

For valencies other than four, similar matrix equations hold; see [308]. Those vertices converge to *extraordinary vertices.*

We may abbreviate (21.3) as

$$\mathrm{V}^{i+1} = A\mathrm{V}^{i}. \tag{21.9}$$

An equivalent form of (21.9) is given by

$$\mathrm{V}^{i+1} = A^{i}\mathrm{V}^{1}. \tag{21.10}$$

We note that the matrix A must be *stochastic*; that is, the elements of each row must sum to one. This is a consequence of the fact that (21.8) represents a barycentric combination. We further note that our mesh refinement consists of taking *convex combinations* at each level: this implies that all meshes $\mathcal{M}^i$ lie in the convex hull of $\mathcal{M}^0$.

It follows that A has the dominant eigenvalue 1; the magnitudes of all other (possibly complex) eigenvalues are bounded by 1.

A careful eigenvalue analysis (see Ball and Storry [18]) can now be used to show that

$$\mathbf{v}^{\infty} = \frac{n^2\mathbf{v}^1 + 4\sum_j \mathbf{e}_j^1 + \sum_j \mathbf{f}_j^1}{n(n+5)}. \tag{21.11}$$

While Catmull–Clark surfaces are generalizations of uniform bicubic spline surfaces, a generalization of the nonuniform case also exists; it is described in [561].

21.4 Midpoint Subdivision

This algorithm is simpler than either Catmull–Clark or Doo–Sabin and goes back to Peters and Reif [469] who call it "the simplest scheme."

1. Form edge points $\mathbf{e}$: for each edge in the mesh, compute its midpoint.
2. Form faces of new level. There are two types: faces inscribed to the existing ones and faces whose vertices are the edge midpoints around old vertices. See Figure 21.7 for an illustration. In order to discuss convergence of the scheme, let $\mathbf{p}_1, \ldots, \mathbf{p}_n$ be the vertices of a face. This face generates vertices $\mathbf{p}_i^1$ as edge midpoints

$$\begin{bmatrix} \mathbf{p}_1^1 \\ \vdots \\ \mathbf{p}_{n-1}^1 \\ \mathbf{p}_n^1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & & & \\ & & \ddots & & \\ & & & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{p}_n \end{bmatrix}. \tag{21.12}$$

We write this as

$$\mathbf{P}^1 = M\mathbf{P}.$$

Matrices of the form (21.12) are called *circulant*.

After k iterations, this becomes

$$\mathbf{P}^k = M^k\mathbf{P}.$$

The matrix M may be decomposed into a product

$$M = E\Lambda E^{-1},$$

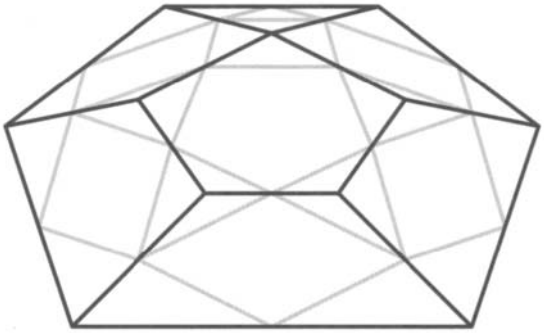


Figure 21.7 Midpoint subdivision: the original control net (black) and one level of subdivision (gray).

where E is the matrix containing M's eigenvectors and Λ is the diagonal matrix containing M's eigenvalues. We then have, following the reasoning in Section 21.1,

$$M^k = E\Lambda^k E^{-1}.$$

The analysis of the limit process is thus closely related to the eigenstructure of M. A result from Davis [134] asserts that for any initial polygon **P**, we obtain regular planar polygons in the limit. These will be the tangent planes to our surface, which therefore is G^1.

21.5 Loop Subdivision

A triangle-based subdivision scheme was discussed by C. Loop [397]. Its input is a triangular mesh as discussed in Section 21.9. It then successively refines this mesh, resulting in a smooth limit surface.

Loop subdivision proceeds as follows.

1. Form edge points $\mathbf{e}_j^{i+1}$: assuming that $\mathbf{v}_1^i$ and $\mathbf{v}_2^i$ are the endpoints of an edge in the mesh and that $\mathbf{v}_3^i$ and $\mathbf{v}_4^i$ are the remaining vertices of the two triangles sharing the edge, set

$$\mathbf{e}_j^{i+1} = \frac{3}{8}(\mathbf{v}_1^i + \mathbf{v}_2^i) + \frac{1}{8}(\mathbf{v}_3^i + \mathbf{v}_4^i). \tag{21.13}$$

   This process is easily visualized using a *mask*, shown in Figure 21.8. The coefficients shown are then multiplied by a factor of 1/8 in order to produce barycentric combinations.

2. For each vertex $\mathbf{v}^i$ in the mesh, form a new vertex point $\mathbf{v}^{i+1}$. Assuming $\mathbf{v}^i$ has n neighbors $\mathbf{v}_1^i, \ldots, \mathbf{v}_n^i$ it is computed as follows:

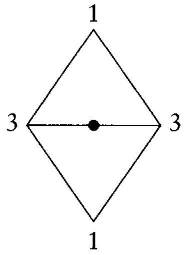


Figure 21.8 Loop subdivision: the edge point mask.

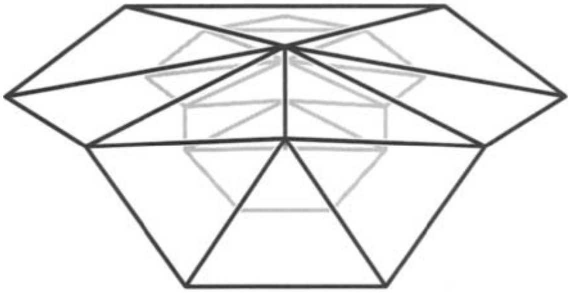


Figure 21.9 Loop subdivision: the original control net (black) and one level of subdivision (gray).

$$\mathbf{v}^{i+1} = (1 - n\alpha)\mathbf{v}^i + \alpha \sum_{j=1}^{n} \mathbf{v}_j^i \tag{21.14}$$

where

$$\alpha = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right)$$

for $n > 3$ and $\alpha = \frac{3}{16}$ if $n = 3$.

3. Form new triangles. Figure 21.9 illustrates.

Now consider the mesh that is obtained after some number k of iterations. Select any vertex $\mathbf{v}$ of it. This vertex will converge to a point $\mathbf{v}^\infty$ on the limit surface. Using an eigenvalue analysis as mentioned, we can show (see [584]) that this limit point is given by

$$\mathbf{v}^\infty = \frac{3 + 8\alpha(n-1)}{3 + 8n\alpha} + \frac{8\alpha}{3 + 8n\alpha} \sum_{j=1}^{n} \mathbf{v}_j, \tag{21.15}$$

where the $\mathbf{v}_j$ are the neighbors of $\mathbf{v}$ in the mesh obtained after k iterations. For the special case $k = 0$, (21.15) gives the limit points corresponding to the original mesh vertices.

If all vertices in the mesh have valency six, then the resulting limit surface will be a collection of quartic Bézier triangles that form a C^2 surface over an equilateral triangulation of a simply connected region of a (domain) plane. We note that closed surfaces cannot be formed using only points with valency six. Vertices with different valencies converge to *extraordinary vertices*, where the surface is only G^1.

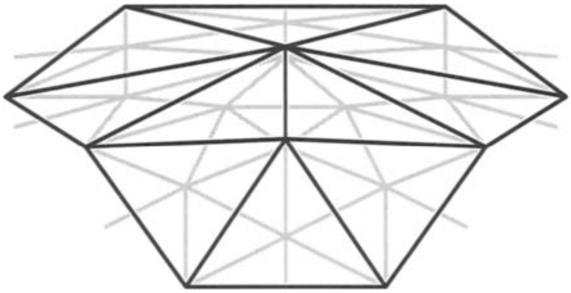


Figure 21.10 $\sqrt{3}$ subdivision: black, original mesh, gray: new mesh after one iteration.

21.6 $\sqrt{3}$ Subdivision

This scheme was developed by L. Kobbelt [362] and was also considered by Sabin [517]. We start with a triangular mesh and then subdivide each triangle into three triangles by splitting it at its centroid. Next, all edges of the initial mesh are flipped—instead of joining the initial vertices, they now join adjacent centroids. Finally, each initial vertex $\mathbf{p}$ (having valency n) is replaced by a barycentric combination of its neighbors:

$$\mathbf{p}^{(1)} = (1 - \alpha_n) + \alpha_n \mathbf{c}$$

where the neighbors of $\mathbf{p}$ are averaged to obtain $\mathbf{c}$. The value for α_n is set at

$$\alpha_n = \frac{1}{9}\left[4 - 2\cos\left(\frac{2\pi}{n}\right)\right].$$

An example of one step of the $\sqrt{3}$ scheme is shown in Figure 21.10.

Each subdivision step rotates the directional structure of the triangular mesh; after two applications, the initial orientation is reestablished. Each original triangle has then given rise to nine new triangles. Following a rigorous study of the eigenstructure of the $\sqrt{3}$ operator, we can show that the resulting limit surface is G^2 except at extraordinary points, where it still is G^1.

21.7 Interpolating Subdivision Surfaces

When dealing with B-splines, we could start from a control polygon and design a curve, or we could start with data points and find an interpolating curve.

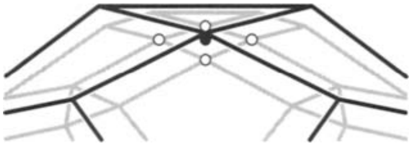


Figure 21.11 Interpolating Doo–Sabin surfaces: given points $\mathbf{p}_i$: solid. Intermediate points $\mathbf{q}_{i,k}$: hollow. Desired vertices $\mathbf{v}_i$: vertices of black mesh.

We can also use recursive subdivision surfaces for interpolation. The idea goes back to Nasri [440] and to M. Lounsbery, S. Mann, and T. DeRose [402]. The latter reference constructs interpolating Catmull–Clark surfaces, whereas Nasri constructs interpolating Doo–Sabin surfaces—we will start with them.

Given is a polyhedron with vertices $\mathbf{p}_i$, and we wish to find another polyhedron with vertices $\mathbf{v}_i$ such that the resulting Doo–Sabin surfaces pass through the $\mathbf{p}_i$. Each of the (unknown) $\mathbf{v}_i$ will generate a V-face with vertices $\mathbf{q}_{i,k}$; $k = 1, \ldots, n_i$, where n_i is the valency of $\mathbf{v}_i$. We know that the centroids of these V-faces are on the surface, and we simply require them to be the given data points:

$$\mathbf{p}_i = \frac{1}{n_i}(\mathbf{q}_{i,1} + \ldots + \mathbf{q}_{i,n_i}).$$

Note that the given points are not on the faces of the desired polyhedron, but rather on the V-faces obtained from it after one level of subdivision; see Figure 21.11 for an illustration.

Since the relationship between the $\mathbf{q}_{i,k}$ and the unknowns $\mathbf{v}_i$ is known, we have a set of linear equations relating the given $\mathbf{p}_i$ to the unknown $\mathbf{v}_i$. For closed polyhedra, the number of equations equals the number of unknowns, leading to a sparse linear system.

This method lends itself to a hybrid usage: some control mesh vertices $\mathbf{v}_i$ may be given as with freeform design, whereas others are replaced by interpolation points $\mathbf{p}_i$. Each of the interpolation points gives rise to one linear equation, and the resulting system is easily solved. See Figure 21.12 for an example.

For open polyhedra, the situation is more complicated; it is dealt with by Nasri [439], [440], [441].

Catmull–Clark subdivision surfaces may also be employed to generate surfaces passing through a prescribed set of data points. We can use (21.11) to generate a control mesh that interpolates to a set of known data points $\mathbf{v}^{\infty}$. All (21.11) form a linear system for the unknowns $\mathbf{v}^1$. This is a sparse system and thus quickly solved even if there are many (> 1000) data points.

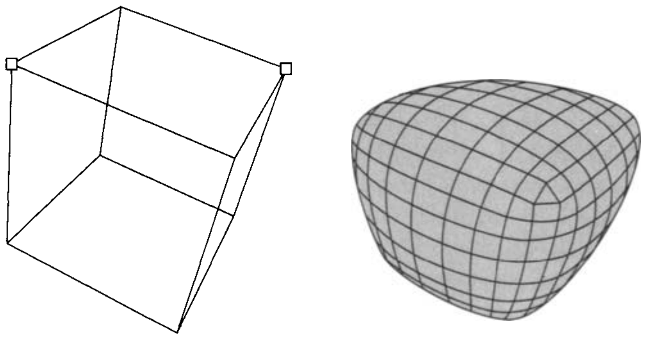


Figure 21.12 Interpolating Doo–Sabin surfaces: two vertices of the initial control net are marked as interpolation points. Resulting surface: right. (Courtesy A. Nasri.)

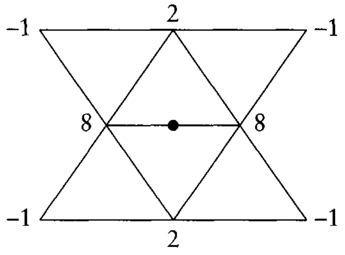


Figure 21.13 Butterfly subdivision: the edge point mask.

It is mentioned in [308] that simply solving the least squares equations for the control points will result in “wiggly” surfaces. Adding shape equations as in Section 15.6 will overcome this problem.

In an similar fashion, (21.15) may be used to find a set of vertices $\mathbf{v}_i$ that will generate a Loop surface through the data points $\mathbf{p}_i$. All these equations form a sparse linear system for the $\mathbf{v}_i$. It may be solved using a sparse solver or by an iterative method.

A more direct way to interpolation, that is, without the need to “invert” existing methods, is given by the *butterfly scheme*, due to Dyn, Gregory, and Levin [176]. Its structure is identical to Loop subdivision, but different weights are applied. At a given level in the refinement process, the vertex points are kept, and only new edge points are generated. Each new edge point is generated from the surrounding vertex points by a mask as shown in Figure 21.13. The (solid) edge point is obtained as a barycentric combination of nearby points; the coefficients in that combination are indicated in the figure. They have to be multiplied by a factor of 1/16.

Just as in the four-point curve interpolation scheme of Section 21.1, interpolation is assured since vertex points are kept.

21.8 Surface Splines

As we iterate through the Doo–Sabin algorithm, more and more of the surface is covered by biquadratic patches, just leaving the extraordinary regions. After two iterations, these are already nicely separated—they correspond to s-sided regions. J. Peters had the idea of deviating from the Doo–Sabin procedure after two steps and to fill in these s-sided regions with a collection of bicubics, such that the overall surface is G^1; see [467] and also [466] or [463].[3] It is not equivalent to the Doo–Sabin surface, but it has the advantage of being a collection of standard patches without singularities.

The situation after two (or more) steps of the Doo–Sabin algorithm is shown in Figure 21.14. We have so far created the points marked by squares. The solid squares mark control points surrounding an s-sided region, whereas the open ones are control points of the network of biquadratic patches. We are going to cover the s-sided region by a collection of s bicubic patches, all having a center point $\mathbf{c}$ in common. This center point is simply the average of all solid control points surrounding the s—sided region, only partially shown in Figure 21.14.

Next, we have to degree elevate each quadratic boundary curve of the s-sided region and to subdivide it at its (parametric) midpoint. This gives two boundary curves of each bicubic patch.

The "outer" two layers of each bicubic patch lie on bilinear patches as shown in Figure 21.14. Their computation[4] is illustrated for the four top left points in that figure:

$$\begin{bmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{11} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{6} & \frac{5}{6} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & \frac{5}{6} \end{bmatrix}$$

The remaining eight Bézier points along the outer patch boundary are found analogously.

The three remaining Bézier points $\mathbf{b}_{22}, \mathbf{b}_{23}, \mathbf{b}_{32}$ are determined as follows:

$$\mathbf{b}_{32}^{(i)} = \mathbf{b}_{23}^{(i+1)} = \frac{4}{3s} \sum_{j=1}^{s} \cos\left(j\frac{2\pi}{s}\right) \frac{\mathbf{d}^{(i+j)} + \mathbf{d}^{(i+j+1)}}{2},$$

[3] For a similar treatment of Catmull-Clark meshes, see [468].

[4] We give a slightly simplified version of Peters's original development [467] here.

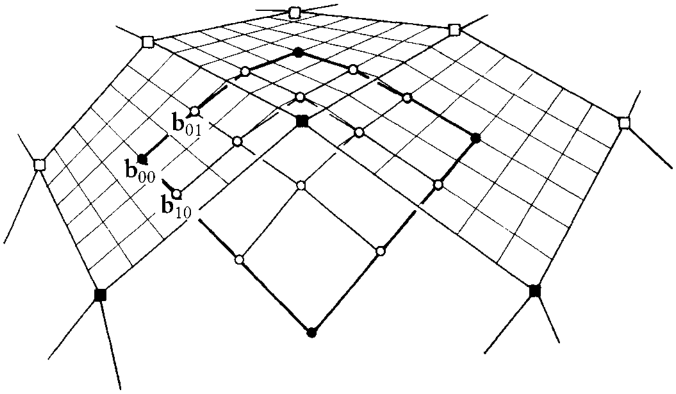


Figure 21.14 Surface splines: the control points determining the Bézier points $\mathbf{b}_{ij}$ of one bicubic patch.

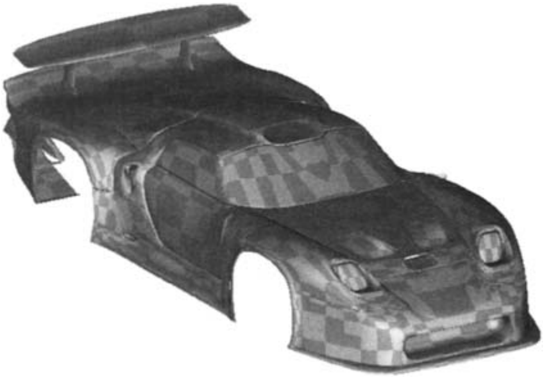


Figure 21.15 Surface splines: an example. (Courtesy J. Peters.)

where the superscript (i) refers to the ith bicubic patch of the patch collection covering the s-sided region. Now all angles $\angle(\mathbf{b}_{33}, \mathbf{b}_{32}, \mathbf{b}_{23})$ are equal. The points $\mathbf{b}_{22}^{(i)}$ must be determined such that G^1 continuity is ensured around $\mathbf{b}_{33}$. Setting $c = \cos(2\pi/s)$ and $\mathbf{e}_i = (1-c)\mathbf{b}_{32}^{(i)} + c\mathbf{b}_{31}^{(i)}$, they are

$$\mathbf{b}_{22}^{(i)} = \begin{cases} -\sum_{j=1}^{s}(-1)^j \mathbf{e}_{i+j} & \text{if } n \text{ is odd,} \\ \frac{-2}{n}\sum_{j=1}^{s}(s-j)(-1)^j \mathbf{e}_{i+j} & \text{if } n \text{ is even.} \end{cases}$$

A surface spline is shown in Figure 21.15.

Surface splines may also be used to interpolate to a mesh of data points in the same way as Doo–Sabin surfaces did: after two steps of the Doo–Sabin algorithm, move those control points that surround given data points such that their average equals that data point. Then proceed as before, and interpolation is ensured.

21.9 Triangular Meshes

We encountered triangular meshes in Section 3.7—there, we were dealing with 2D meshes. Keeping the same data structure, we may generalize 2D triangulations to 3D triangle meshes: these are surfaces consisting of a collection of triangles; see Figure 21.16. The vertices $\mathbf{p}_i$ of the triangles are now 3D points. Triangle meshes are piecewise linear and thus are not smooth—although this defect may be overcome by using very many triangles. One example is provided by the "Digital Michelangelo" project, carried out by M. Levoy. The aim of the project is to digitally record sculptures by Renaissance artist Michelangelo. Each sculpture was digitized using a laser digitizer; the resulting "point cloud" was then triangulated. For some sculptures, around two billion triangles were needed; see [386].

Boundary edges are edges in the mesh that belong to one triangle only; all other edges, being part of two triangles, are called *interior edges*. A vertex is called *boundary vertex* if it is on a boundary edge; otherwise, it is called an *interior vertex*. The solid vertex in Figure 21.17 is an interior vertex.

A significant difference between 2D and 3D triangulations is their *topology*. Every 2D triangulation must have boundary vertices, but 3D triangulations do not have to have any. Consider the example of a tetrahedron, the simplest 3D

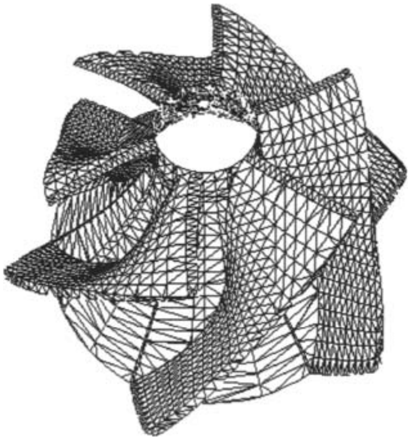


Figure 21.16 Triangular meshes: an example of a triangulated turbine. (Courtesy 3D Compression Technologies.)

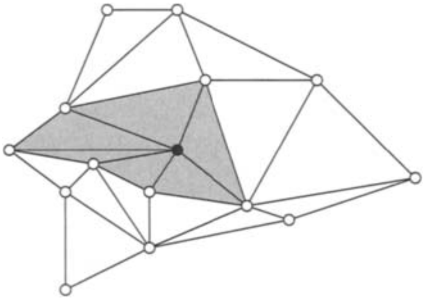


Figure 21.17 Star of a point: the shaded triangles form the star of the solid point.

triangle mesh. It has four vertices and four triangles, all of them interior vertices. Triangular meshes without boundaries are called *closed*; those that do have boundaries are called *open*.

Two additional examples of closed meshes are a triangulated sphere and a triangulated torus. A major difference between these is their *genus*, a number describing the topology of a mesh. The genus of a closed mesh is the number of holes it has. Thus a sphere has genus zero, a torus has genus one, a double torus has genus two, and so forth. If we denote the number of faces, edges, and vertices by f, e, v, respectively, then the following equation may be used to define the genus G:

$$G = \frac{1}{2}(v - e + f - 2). \tag{21.16}$$

This is known as the *Euler–Poincaré* formula and is valid even for meshes with polygonal faces, not just triangular ones. For example, a cube consisting of six square faces, twelve edges, and eight vertices, has genus 0. If we split each square into two triangles, we have $(f, e, v) = (12, 18, 8)$, again resulting in genus zero. For more details, see Hoffmann [327].

21.10 Decimation

In many cases, triangulations are far too dense for an efficient representation of an object. What is called for then is a reduction in size, also known as *decimation*.[5]

[5] The origin of the word can be traced to Roman times. When a legion fled during battle, it was punished by having every tenth soldier killed (decimated).

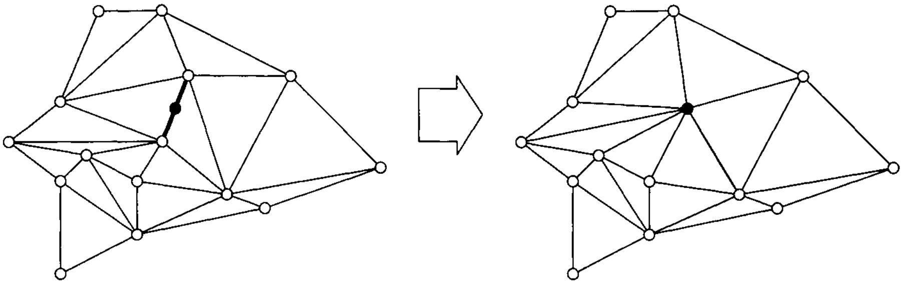


Figure 21.18 Edge collapse: an edge is marked for collapsing, left. It is then collapsed into a single point, right.

The goal is to remove as many points as possible, while still staying close to the initial triangulation.

A decimation algorithm will thus perform a check on every point and determine if it can be removed. If so, the point (and sometimes its neighbors) are removed and the resulting gap will be retriangulated. The process continues until no more data can be removed.

An important ingredient in many mesh algorithms is a test for determining if a mesh is flat in the vicinity of a vertex $\mathbf{p}$. For the following, we will need the concept of the *star* $\mathbf{p}_i^\star$ of a vertex $\mathbf{p}_i$: this is the set of all triangles having $\mathbf{p}_i$ as a vertex; see Figure 21.17. Thus the star of a point is itself a triangle mesh.

In order to check if the mesh is flat around $\mathbf{p}$, we compute the angles between all pairs of neighboring triangles in $\mathbf{p}^\star$. If the largest of these angles is less than a given tolerance, then we label $\mathbf{p}$ as flat. The tolerance is naturally application dependent, but 0.1 to 1 degrees should work well. This flatness test is independent of scalings of the mesh since angles are not affected by scales.

A different flatness criterion is due to M. Garland and P. Heckbert [254]. It locally constructs a quadric that approximates the neighborhood of a vertex. If the vertex is within a prescribed tolerance to this quadric, then it is labeled removable. This test is scale dependent: if the mesh is scaled, then the tolerance has to be scaled as well.

We now discuss a decimation method that is due to M. Lounsbery [400], [401]. It checks if an edge in a triangulation can be safely removed. If so, the edge is replaced by a point on it, thereby replacing two points by one. Retriangulation is fairly trivial: Figure 21.18 gives an example. A simple choice for the edge replacement point is the midpoint of the edge; more elaborate methods place it such that the shape of the resulting triangles is optimized.

When can an edge be removed? We simply perform a flatness test on the two edge vertices. If both meet the criterion, then the edge may be removed. Before that happens, it is advisable to check if the new triangles are within a specified

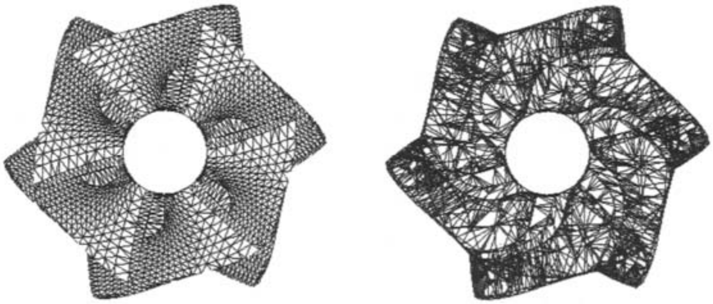


Figure 21.19 Decimation: the data set of Figure 21.16 (left), and after decimation (right). (Courtesy 3D Compression Technologies.)

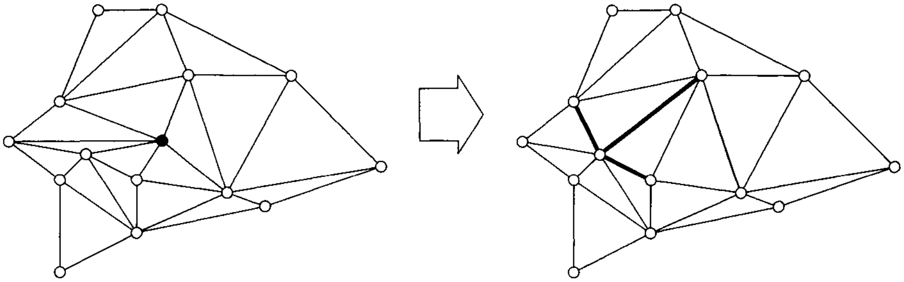


Figure 21.20 Vertex removal: a vertex is labeled removable (left), and its star is retriangulated (right).

tolerance to the old ones. An example of a decimated mesh is shown in Figure 21.19. This decimation method lends itself well to a *multiresolution* analysis of a triangle mesh. If we keep track of each collapsed edge, we create a hierarchy of triangle meshes. Each vertex in that hierarchy "knows" if it was generated by an edge collapse. The inverse process—recreating an edge from a vertex—is known as *vertex split*. Thus a heavily decimated mesh may be refined by successively adding in more edges using vertex splits. This successive refinement lends itself to streaming data transfer. More literature on multiresolution methods: [184], [183], [331], [330], [355].

A different decimation strategy is to remove a vertex, not an edge; this is due to Schroeder, Zarge, and Lorensen [545]. If a vertex is labeled removable, it is deleted and its star is retriangulated, see Figure 21.20.

For more literature on mesh decimation, see [116], [404], [544].

21.11 Problems

1 Consider the following subdivision method: starting with a closed polygonal mesh, recursively subdivide by alternating between the Doo–Sabin and Catmull–Clark schemes. What can you say about the number of extraordinary vertices?

2 The Euler–Poincaré formula (21.16) always produces a genus that is an integer. Why?

3 Take a cube with square faces and place three square holes through it, each hole connecting opposite faces. What is the genus of the resulting object?

4 Sketch the effect of two levels of the $\sqrt{3}$ scheme.

***5** The Doo–Sabin recursion generates a sequence of F-faces for every face, in the limit converging to the centroid of the considered face. Show that the limiting F-faces are planar.

***6** Each of the triangles in $\mathbf{p}^\star$ forms an angle at $\mathbf{p}$. Call the sum of all these angles Σ_p. If $\Sigma_\mathrm{p} = 360$, then all triangles are in one plane. Thus the quantity $360 - |\Sigma_\mathrm{p}|$ measures the nonplanarity of $\mathbf{p}^\star$—yet it is not a good flatness indicator. Why?

P1 Write a program to find an interpolating Doo–Sabin surface to the eight vertices of a cube.

Coons Patches 22

We have already encountered design tools that originated in car companies; Bézier curves and surfaces were developed by Citroën and Renault in Paris. Two other major concepts also emerged from the automotive field: Coons patches (S. Coons consulted for Ford, Detroit) and Gordon surfaces (W. Gordon worked for General Motors, Detroit).[1] These methods have a different flavor than Bézier or B-spline methods: instead of being described by control nets, they "fill in" curve networks in order to generate surfaces.

A designer does not think in terms of surfaces but rather in terms of "feature curves"; these are lines on a car between which the actual surfaces fit "naturally." In Color Plate III, we can see some of these lines as boundary curves of B-spline surfaces. Once a designer has produced the feature lines, a filling-in process follows that generates a surface from a network of curves. The techniques used in this process are known by the names of their inventors, Coons and Gordon.

Additional literature on Coons patches includes Coons's "little red book" [125] (also available in a French translation [128]) and Barnhill [21], [22]. In the area of numerical grid generation for computational fluid dynamics, Coons patches are also frequently employed; here, they are known as *transfinite interpolants*; see [588].

[1] Just for the record, in the late 1960s, Chrysler began to develop a curve and surface scheme that was based on Chebychev polynomials.

22.1 Coons Patches: Bilinearly Blended

We encountered Coons patches in the context of "filling in" control nets in Section 15.2. The underlying principle is applicable to more general surfaces. Here, the boundary curves are not piecewise linear control polygons, but arbitrary parametric curves. This simplest instance of Coons patches was also developed first by Coons [124].

To be more precise: given are four arbitrary curves $\mathbf{c}_1(u), \mathbf{c}_2(u)$ and $\mathbf{d}_1(v), \mathbf{d}_2(v)$, defined over $u \in [0, 1]$ and $v \in [0, 1]$, respectively, find a surface $\mathbf{x}$ that has these four curves as boundary curves:

$$\mathbf{x}(u, 0) = \mathbf{c}_1(u), \quad \mathbf{x}(u, 1) = \mathbf{c}_2(u), \tag{22.1}$$

$$\mathbf{x}(0, v) = \mathbf{d}_1(v), \quad \mathbf{x}(1, v) = \mathbf{d}_2(v). \tag{22.2}$$

The four boundary curves define two ruled surfaces:

$$\mathbf{r}_c(u, v) = (1 - v)\mathbf{x}(u, 0) + v\mathbf{x}(u, 1)$$

and

$$\mathbf{r}_d(u, v) = (1 - u)\mathbf{x}(0, v) + u\mathbf{x}(1, v).$$

Both interpolants are shown in Figure 22.1, and we see that $\mathbf{r}_c$ interpolates to the c-curves, yet fails to reproduce the $\mathbf{d}$-curves. The situation for $\mathbf{r}_d$ is similar, and therefore equally unsatisfactory. Both $\mathbf{r}_c$ and $\mathbf{r}_d$ do well on two sides, yet fail on the other two, where they are *linear*. Our strategy is therefore as follows: let us try to retain what each ruled surface interpolates to, and let us try to eliminate what each fails to interpolate to. A little thought reveals that the "interpolation failures" are captured by one surface: the bilinear interpolant $\mathbf{r}_{cd}$ to the four corners (see also Section 14.1):

$$\mathbf{r}_{cd}(u, v) = [\, 1 - u \quad u \,] \begin{bmatrix} \mathbf{x}(0,0) & \mathbf{x}(0,1) \\ \mathbf{x}(1,0) & \mathbf{x}(1,1) \end{bmatrix} \begin{bmatrix} 1 - v \\ v \end{bmatrix}.$$

We are now ready to create a Coons patch $\mathbf{x}$. It is given by

$$\mathbf{x} = \mathbf{r}_c + \mathbf{r}_d - \mathbf{r}_{cd}, \tag{22.3}$$

or, in the form of a recipe: "loft$_u$ + loft$_v$ − bilinear." The involved surfaces and the solution are illustrated in Figure 22.1. Writing (22.3) in full detail gives

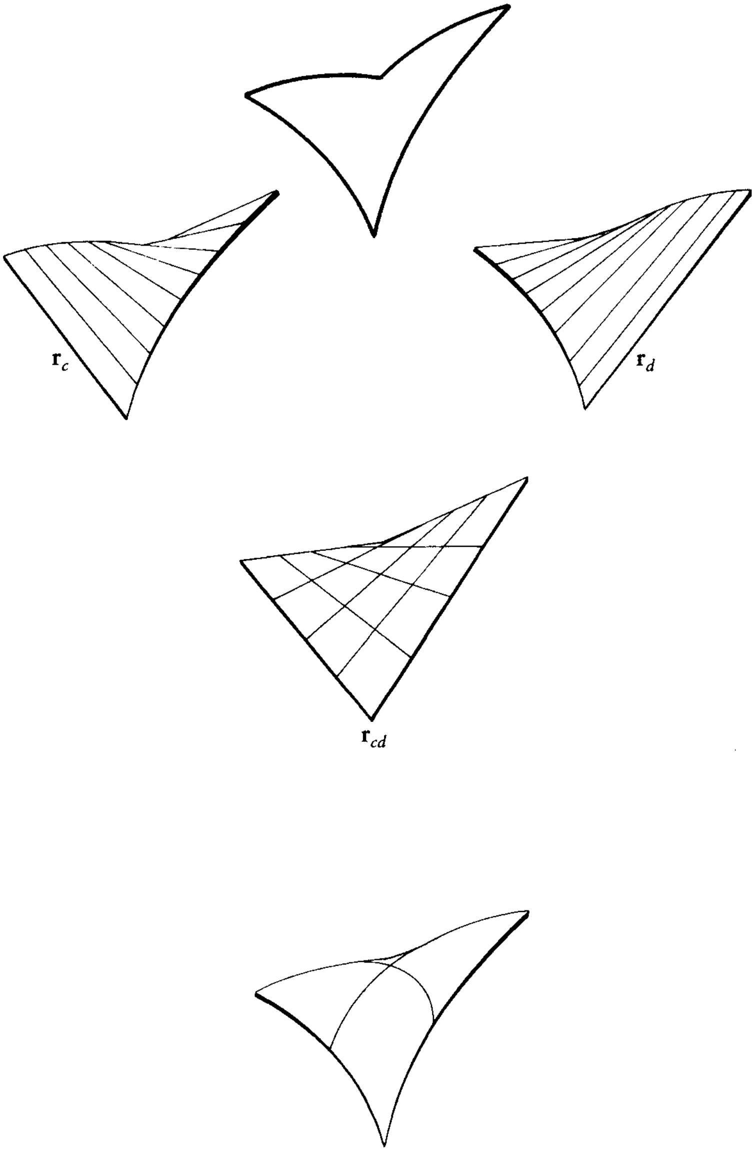


Figure 22.1 Coons patches: a bilinearly blended Coons patch comprises two lofted surfaces and a bilinear surface.

$$\begin{aligned}\mathbf{x}(u,v) = {} & \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{x}(0,v) \\ \mathbf{x}(1,v) \end{bmatrix} \\ & + \begin{bmatrix} \mathbf{x}(u,0) & \mathbf{x}(u,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} \\ & - \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{x}(0,0) & \mathbf{x}(0,1) \\ \mathbf{x}(1,0) & \mathbf{x}(1,1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}. \end{aligned} \tag{22.4}$$

It is left as an exercise to verify that (22.4) does indeed interpolate to all four boundary curves.

We can now justify the name *bilinearly blended* for the preceding Coons patch: a ruled surface "blends" together the two defining boundary curves; this blending takes place in both directions. However, the Coons patch is *not* generally itself a bilinear surface—the name refers purely to the method of construction.

The functions 1—u, u and 1—v, v are called *blending functions.* A close inspection of (22.4) reveals that many other pairs of blending functions, say, $f_1(u), f_2(u)$ and $g_1(v), g_2(v)$, could also be used to construct a generalized Coons patch. It would then be of the general form

$$\begin{aligned}\mathbf{x}(u,v) = {} & \begin{bmatrix} f_1(u) & f_2(u) \end{bmatrix} \begin{bmatrix} \mathbf{x}(0,v) \\ \mathbf{x}(1,v) \end{bmatrix} \\ & + \begin{bmatrix} \mathbf{x}(u,0) & \mathbf{x}(u,1) \end{bmatrix} \begin{bmatrix} g_1(v) \\ g_2(v) \end{bmatrix} \\ & - \begin{bmatrix} f_1(u) & f_2(u) \end{bmatrix} \begin{bmatrix} \mathbf{x}(0,0) & \mathbf{x}(0,1) \\ \mathbf{x}(1,0) & \mathbf{x}(1,1) \end{bmatrix} \begin{bmatrix} g_1(v) \\ g_2(v) \end{bmatrix}. \end{aligned} \tag{22.5}$$

There are only two restrictions on the f_i and g_i: each pair must sum to one identically: otherwise, we would generate nonbarycentric combinations of points (see Section 2.1). Further, we must have $f_1(0) = g_1(0) = 1, f_1(1) = g_1(1) = 0$ in order to actually interpolate. The shape of the blending functions has a predictable effect on the shape of the resulting Coons patch. Typically, we require f_1 and g_1 to be monotonically decreasing; this produces surfaces of predictable shape, but is not necessary for theoretical reasons.

22.2 Coons Patches: Partially Bicubically Blended

The bilinearly blended Coons patch solves a problem of considerable importance with very little effort, but we pay for that by an annoying drawback. Consider Figure 22.2: it shows two bilinearly blended Coons patches, defined over $u \in$

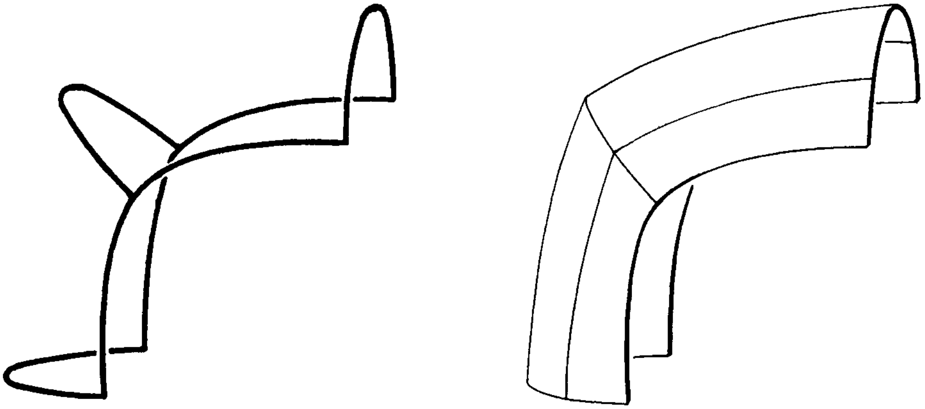


Figure 22.2 Coons patches: the input curves for two neighboring patches may have C^1 boundary curves (left), yet the two Coons patches determined by them do not form a smooth surface (right).

$[0,2]$, $v \in [0,1]$. The boundary curves $v = 0$ and $v = 1$, both composite curves, are differentiable. However, the cross-boundary derivative is clearly discontinuous along $u = 1$; also see Problems.[2]

Analyzing this problem, we see that it can be blamed on the fact that cross-boundary tangents along one boundary depend on *data not pertaining to that boundary*. For example, for any given bilinearly blended Coons patch, a change in the boundary curve $\mathbf{x}(1,v)$ will affect the derivatives across the boundary $\mathbf{x}(0,v)$.

How can we separate the derivatives across one boundary from information along the opposite boundary? The answer: use different blending functions, namely, some that have zero slopes at the endpoints. Striving for simplicity, as usual, we find two obvious candidates for such blending functions: the cubic Hermite polynomials H_0^3 and H_3^3 from Section 7.5, as defined by (7.14).

Let us investigate the effect of this choice of blending functions: we have set $f_1 = g_1 = H_0^3$ and $f_2 = g_2 = H_3^3$ in (22.5). The cross-boundary derivative along, say, $u = 0$, now becomes:

$$\mathbf{x}_u(0,v) = [\,\mathbf{x}_u(0,0) \quad \mathbf{x}_u(0,1)\,] \begin{bmatrix} H_0^3(v) \\ H_3^3(v) \end{bmatrix}; \tag{22.6}$$

[2] We also see that bilinearly blended Coons patches suffer from a shape defect: each of the two patches is too "flat." This effect of Coons patches has been studied by Nachman [436].

all other terms vanish since $\mathrm{d}/\mathrm{d}uH_{3i}^3(0) = \mathrm{d}/\mathrm{d}uH_{3i}^3(1) = 0$ for $i = 0$ and $i = 1$. Thus, the only data that influence $\mathbf{x}_u$ along $u = 0$ are the two tangents $\mathbf{x}_u(0, 0)$ and $\mathbf{x}_u(0, 1)$—we have achieved our goal of making the cross-boundary derivative along one boundary depend only on information pertaining to that boundary. With our new blending functions, the two patches from Figure 22.2 would now be C^1.

Unfortunately, we have also created a new problem. At the patch corners, these patches often have "flat spots." The reason: partially bicubically blended Coons patches,[3] constructed as before, suffer from *zero corner twists*:

$$\mathbf{x}_{uv}(i, j) = 0; \quad i, j \in \{0, 1\}.$$

This is easily verified by simply taking the uv-partial of (22.5) and evaluating at the patch corners.

The reason for this poor performance lies in the fact that we use only two functions, H_0^3 and H_3^3 from the full set of four Hermite polynomials. Both have zero derivatives at the interval endpoints, and pass that property on to the surface interpolant.

We will now modify the partially bicubically blended Coons patch in order to avoid the flat spots at the corners.

22.3 Coons Patches: Bicubically Blended

Cubic Hermite interpolation needs more input than positional data—first derivative information is needed. Since our positional input consists of whole curves, not just points, the obvious data to supply are derivatives along those input curves. Our given data now consist of

$$\mathbf{x}(u, 0), \ \mathbf{x}(u, 1), \ \mathbf{x}(0, v), \ \mathbf{x}(1, v)$$

and

$$\mathbf{x}_v(u, 0), \quad \mathbf{x}_v(u, 1), \quad \mathbf{x}_u(0, v), \quad \mathbf{x}_u(1, v).$$

We can think of the now prescribed cross-boundary derivatives as "tangent ribbons," illustrated in Figure 22.3 (only two of the four "ribbons" are shown there).

[3] We use the term *partially* bicubically blended since only a part of all cubic Hermite basis functions is used.

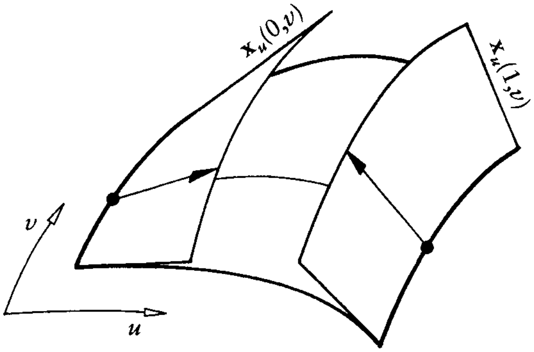


Figure 22.3 Coons patches: for the bicubically blended case, the concept of the lofted surface is generalized. In addition to the given boundary curves, cross-boundary derivatives are supplied.

The derivation of the bicubically blended Coons patch is analogous to the one in Section 22.1: we must simply generalize the concept of a ruled surface appropriately. This is almost trivial; we obtain

$$\mathbf{h}_c(u,v) = H_0^3(u)\mathbf{x}(0,v) + H_1^3(u)\mathbf{x}_u(0,v) + H_2^3(u)\mathbf{x}_u(1,v) + H_3^3(u)\mathbf{x}(1,v)$$

for the u-direction (this surface is shown in Figure 22.3) and

$$\mathbf{h}_d(u,v) = H_0^3(v)\mathbf{x}(u,0) + H_1^3(v)\mathbf{x}_v(u,0) + H_2^3(v)\mathbf{x}_v(u,1) + H_3^3(v)\mathbf{x}(u,1).$$

Proceeding as in the bilinearly blended case, we define the interpolant to the corner data. This gives the tensor product bicubic Hermite interpolant $\mathbf{h}_{cd}$ from Section 15.5:

$$\mathbf{h}_{cd}(u,v) = \begin{array}{l} [\,H_0^3(u) \quad H_1^3(u) \quad H_2^3(u) \quad H_3^3(u)\,] \\ \times \begin{bmatrix} \mathbf{x}(0,0) & \mathbf{x}_v(0,0) & \mathbf{x}_v(0,1) & \mathbf{x}(0,1) \\ \mathbf{x}_u(0,0) & \mathbf{x}_{uv}(0,0) & \mathbf{x}_{uv}(0,1) & \mathbf{x}_u(0,1) \\ \mathbf{x}_u(1,0) & \mathbf{x}_{uv}(1,0) & \mathbf{x}_{uv}(1,1) & \mathbf{x}_u(1,1) \\ \mathbf{x}(1,0) & \mathbf{x}_v(1,0) & \mathbf{x}_v(1,1) & \mathbf{x}(1,1) \end{bmatrix} \begin{bmatrix} H_0^3(v) \\ H_1^3(v) \\ H_2^3(v) \\ H_3^3(v) \end{bmatrix}. \end{array} \qquad (22.7)$$

The bicubically blended Coons patch now becomes

$$\mathbf{x} = \mathbf{h}_c + \mathbf{h}_d - \mathbf{h}_{cd}. \qquad (22.8)$$

Before closing this section, we need to take a closer look at the $\mathbf{h}_{cd}$ part of (22.8). On closer inspection, we find that it wants data that we were not willing (or able) to provide in our initial problem description, namely, the central "twist partition" of the 4×4 matrix in (22.7). The bicubically blended Coons patch needs these quantities as input, and this has caused CAD software developers many headaches since Coons proposed his surface scheme in 1964. The most popular "solution" seems to be simply to define each of the four corner twists to be the zero vector. The drawbacks of that choice were already discussed in Section 14.10, but alternatives are pointed out in that section, too.

22.4 Piecewise Coons Surfaces

We will now apply the bicubically blended patch to the situation for which it was intended: we assume that we are given a network of curves as shown in Figure 22.6 and that we want to fill in this curve network with bicubically blended Coons patches. The resulting surface will be C^1.

To apply (22.8), we must create twist vectors and cross-boundary derivatives (tangent ribbons) from the given curve network. As a preprocessing step, we estimate a twist vector $\mathbf{x}_{uv}(u_i, v_j)$ at each patch corner. This can be done by using any of the twist vector estimators discussed in Section 16.3. In that section, we assumed that the boundary curves of each patch were cubics; that assumption does not affect the computation of the twist vectors at all, however.

Having found a twist vector for each data point, we now need to create cross-boundary derivatives for each boundary curve. Let us focus our attention on one patch of our network, and let us assume for simplicity (but without loss of generality!) that the parameters u and v vary between 0 and 1. We shall now construct the tangent ribbon $\mathbf{x}_v(u, 0)$. We have four pieces of information about $\mathbf{x}_v(u, 0)$: the values of $\mathbf{x}_v(u, 0)$ at $u = 0$ and at $u = 1$, and the derivatives with respect to u there—these are the twists that we made up earlier, namely, $\mathbf{x}_{uv}(0, 0)$ and $\mathbf{x}_{uv}(1, 0)$.

We therefore have the input data for a univariate cubic Hermite interpolant, and the desired tangent ribbon assumes the form

$$\begin{aligned}\mathbf{x}_v(u, 0) = {} & \mathbf{x}_v(0, 0)H_0^3(u) + \mathbf{x}_{uv}(0, 0)H_1^3(u) \\ & + \mathbf{x}_{uv}(1, 0)H_2^3(u) + \mathbf{x}_v(1, 0)H_3^3(u). \end{aligned} \tag{22.9}$$

The remaining three tangent ribbons are computed analogously.

We have thus found a way to pass a C^1 surface through a C^1 network of curves. All we needed was the ability to estimate the twists at the data points.

22.5 Two Properties

Coons patches are *twist minimizing* in the sense that

$$\int_S \mathbf{x}_{uv}^2 \mathrm{d}S \tag{22.10}$$

is minimal exactly for the Coons patch (S being the domain).

If we apply the minimum principle to the discrete Coons patch, we have that

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[\Delta^{1,1}\mathbf{b}_{i,j}]^2$$

is minimal if the $\mathbf{b}_{i,j}$ form a discrete Coons patch.

Coons patches satisfy a *permanence principle*: let two points (u_0, v_0) and (u_1, v_1) define a rectangle R in the domain of a Coons patch. The four boundaries of this subpatch will map to four curves on the Coons patch. You may ask what the Coons patch to those four boundary curves is. The answer: the original Coons patch, restricted to the rectangle R.[4] In this sense, Coons patches are self-similar!

We can apply this principle to a 3 × 3 grid of a discrete Coons patch. Such a grid is given by

$$\begin{matrix} \mathbf{b}_{i-1,j+1} & \mathbf{b}_{i,j+1} & \mathbf{b}_{i+1,j+1} \\ \mathbf{b}_{i-1,j} & \mathbf{b}_{i,j} & \mathbf{b}_{i+1,j} \\ \mathbf{b}_{i-1,j-1} & \mathbf{b}_{i,j-1} & \mathbf{b}_{i+1,j-1} \end{matrix}$$

As a consequence of the permanence principle,

$$\begin{aligned} \mathbf{b}_{i,j} &= -\frac{1}{4}(\mathbf{b}_{i-1,j+1} + \mathbf{b}_{i+1,j+1} + \mathbf{b}_{i-1,j-1} + \mathbf{b}_{i+1,j-1}) \\ &+ \\ &\frac{1}{2}(\mathbf{b}_{i,j+1} + \mathbf{b}_{i-1,j} + \mathbf{b}_{i+1,j} + \mathbf{b}_{i,j-1}). \end{aligned} \tag{22.11}$$

We could thus obtain the discrete Coons patch as the solution to a linear system of equations. This is not practical for the construction of a rectangular control net, but offers a basic construction principle for more complex surfaces. For more details, see [207].

[4] The term *permanence principle* for Coons patches was coined by R. Barnhill in a slightly different context around 1980.

22.6 Compatibility

It is an obvious requirement for the bilinearly blended Coons patch that the four prescribed boundary curves meet at the corners; in other words, we must exclude data configurations as shown in Figure 22.4. This condition on the prescribed data is known as a *compatibility condition*. An incompatibility of that form can usually be overcome by adjusting boundary curves so that they meet at the patch corners.

The bicubically blended Coons patch suffers from a more difficult compatibility problem. It results from the appearance of the twist terms in the tensor product term $\mathbf{h}_{cd}$ in (22.7). The problem was not recognized by Coons, and only later did R. Barnhill and J. Gregory discover it; see Gregory [288].

From calculus, we know that we can usually interchange the order of differentiation when taking mixed partials: we can set $\mathbf{x}_{uv} = \mathbf{x}_{vu}$ if $\mathbf{x}(u, v)$ is twice continuously differentiable. Unfortunately, this simplification does not apply to our situation. Let us examine why: at $\mathbf{x}(0, 0)$, two given "tangent ribbons" meet. We can obtain the twist at $\mathbf{x}(0, 0)$ by differentiating the "ribbon" $\mathbf{x}_v(u, 0)$ with respect to u:

$$\mathbf{x}_{vu}(0, 0) = \lim_{u \to 0} \frac{\partial}{\partial u} \mathbf{x}_v(u, 0),$$

or the other way around:

$$\mathbf{x}_{uv}(0, 0) = \lim_{v \to 0} \frac{\partial}{\partial v} \mathbf{x}_u(0, v).$$

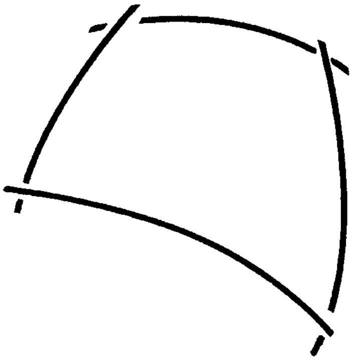


Figure 22.4 Compatibility problems: in the case of a bilinearly blended Coons patch, compatible boundary curves must be prescribed. Data as shown lead to ill-defined interpolants.

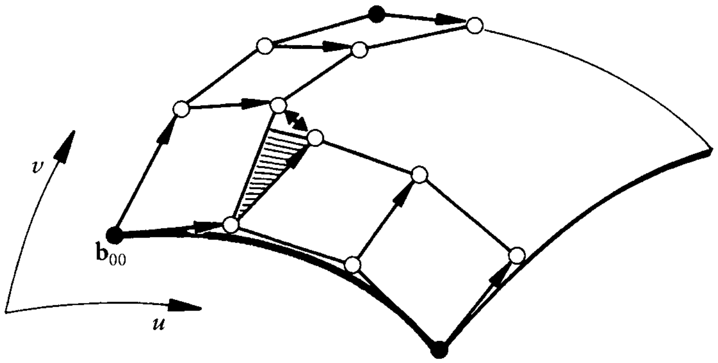


Figure 22.5 Compatibility problems: we show the example of tangent ribbons that are represented in cubic Bézier form. Note how we obtain two different interior Bézier points, and thus two different corner twists.

If the two twists $\mathbf{x}_{uv}(0,0)$ and $\mathbf{x}_{vu}(0,0)$ are equal, there are no problems: enter this twist term into the matrix in (22.7), and the bicubically blended Coons patch is well defined.

However, as Figure 22.5 illustrates, these two terms need not be equal. Now we have a serious dilemma: entering either one of the two values yields a surface that only partially interpolates to the given data. Entering zero twist vectors only aggravates matters, since they will in general not agree with even one of the two twists.

There are two ways out of this dilemma. One is to try to adjust the given data so that the incompatibilities disappear. Or, if the data cannot be changed, we can use a method known as *Gregory's square*. This method replaces the constant twist terms in the matrix in (22.7) by *variable twists*. The variable twists are computed from the tangent ribbons:

$$\mathbf{x}_{uv}(0,0) = \frac{u\frac{\partial}{\partial v}\mathbf{x}_u(0,0) + v\frac{\partial}{\partial u}\mathbf{x}_v(0,0)}{u+v},$$

$$\mathbf{x}_{uv}(0,1) = \frac{-u\frac{\partial}{\partial v}\mathbf{x}_u(0,1) + (v-1)\frac{\partial}{\partial u}\mathbf{x}_v(0,1)}{-u+v-1},$$

$$\mathbf{x}_{uv}(1,0) = \frac{(1-u)\frac{\partial}{\partial v}\mathbf{x}_u(1,0) + v\frac{\partial}{\partial u}\mathbf{x}_v(1,0)}{1-u+v},$$

$$\mathbf{x}_{uv}(1,1) = \frac{(u-1)\frac{\partial}{\partial v}\mathbf{x}_u(1,1) + (v-1)\frac{\partial}{\partial u}\mathbf{x}_v(1,1)}{u-1+v-1}.$$

The resulting surface does not have a continuous twist at the corners. In fact, it is *designed* to be discontinuous: it assumes two different values, depending on from where the corner is approached. If we approach $\mathbf{x}(0,0)$, say, along the isoparametric line $u=0$, we should get the u-partial of the given tangent ribbon $\mathbf{x}_v(u,0)$ as the twist $\mathbf{x}_{uv}(0,0)$. If we approach the same corner along $v=0$, we should get the v-partial of the given ribbon $\mathbf{x}_u(0,v)$ to be $\mathbf{x}_{uv}(0,0)$.

An interesting application of Gregory's square was developed by Chiyokura and Kimura [112]: suppose we are given four boundary curves of a patch in cubic Bézier form, and suppose that the cross-boundary derivatives also vary cubically. Let us consider the corner $\mathbf{x}(0,0)$ and the two boundary curves that meet there. These curves define the Bézier points $\mathbf{b}_{0j}$ and $\mathbf{b}_{i0}$. The cross-boundary derivatives determine $\mathbf{b}_{1j}$ and $\mathbf{b}_{i1}$. Note that $\mathbf{b}_{11}$ is defined twice! This situation is illustrated in Figure 22.5. Chiyokura and Kimura made $\mathbf{b}_{11}$ a function of u and v:

$$\mathbf{b}_{11}=\mathbf{b}_{11}(u,v)=\frac{u\mathbf{b}_{11}(v)+v\mathbf{b}_{11}(u)}{u+v},$$

where $\mathbf{b}_{11}(u)$ denotes the point $\mathbf{b}_{11}$ that would be obtained from the cross-boundary derivative $\mathbf{x}_u(0,v)$, and so on. Similar expressions hold for the remaining three interior Bézier points, all following the pattern of Gregory's square.

Although a solution to the posed problem, we should note that Gregory's square (or the Chiyokura and Kimura application) is not free of problems. Even with polynomial input data, it will produce a rational patch. Written in rational Bézier form, its degree is seven in both u and v and the corner weights are zero (see [202]). The resulting singularities are removable, but require special attention. In situations where we are not forced to use incompatible cross-boundary derivatives, it is therefore advisable first to estimate corner twists and then to use (22.9) as a cross-boundary derivative generator.

22.7 Gordon Surfaces

Gordon surfaces are a generalization of Coons patches. They were developed in the late 1960s by W. Gordon [281], [283], [280], [282], who was then working for the General Motors Research labs. He coined the term ***transfinite interpolation*** for this kind of surface.

It is often not sufficient to model a surface from only four boundary curves. A more complicated (and realistic) situation arises when a *network* of curves is prescribed, as shown in Figure 22.6. We will construct a surface $\mathbf{g}$ that interpolates to all these curves—they will then be isoparametric curves $\mathbf{g}(u_i,v); i=0,\ldots,m$ and $\mathbf{g}(u,v_j); j=0,\ldots,n$. We shall therefore refer to these input curves in terms of the final surface $\mathbf{g}$. The idea behind the construction of this *Gordon surface*

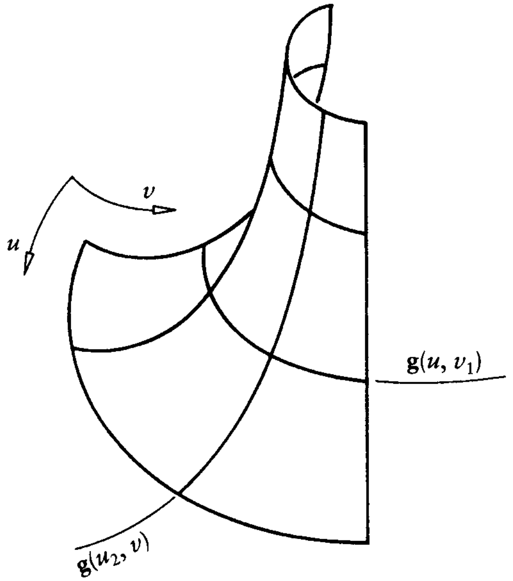


Figure 22.6 Gordon surfaces: a rectilinear network of curves is given and an interpolating surface is sought.

$\mathbf{g}$ is the same as for the Coons patch: find a surface $\mathbf{g}_1$ that interpolates to one family of isoparametric curves, for instance to the $\mathbf{g}(u_i, v)$. Next, find a surface $\mathbf{g}_2$ that interpolates to the $\mathbf{g}(u, v_j)$. Finally, add both together and subtract a surface $\mathbf{g}_{12}$.

Let us start with the task of finding the surface $\mathbf{g}_1$. If there are only two curves $\mathbf{g}(u_0, v)$ and $\mathbf{g}(u_1, v)$, the surface $\mathbf{g}_1$ reduces to the lofted surface $\mathbf{g}_1(u, v) = L_0^1(u)\mathbf{g}(u_0, v) + L_1^1(u)\mathbf{g}(u_1, v)$, where the L_i^1 are the linear Lagrange polynomials from Section 7.2. If we have more than two input curves, we might want to try higher-degree Lagrange polynomials:

$$\mathbf{g}_1(u, v) = \sum_{i=0}^{m} \mathbf{g}(u_i, v) L_i^m(u). \qquad (22.12)$$

Simple algebra verifies that we have successfully generalized the concept of a lofted surface.

Let us return to the construction of the Gordon surface, for which $\mathbf{g}_1$ will only be a building block. The second building block, $\mathbf{g}_2$, is obtained by analogy:

$$\mathbf{g}_2(u,v)=\sum_{j=0}^{n}\mathbf{g}(u,v_j)L_j^n(v).$$

The third building block, $\mathbf{g}_{12}$, is simply the interpolating tensor product surface

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

The Gordon surface $\mathbf{g}$ now becomes

$$\mathbf{g}=\mathbf{g}_1+\mathbf{g}_2-\mathbf{g}_{12}. \tag{22.13}$$

It is left as an exercise to verify that (22.13) in fact interpolates to all given curves. Note that for the actual computation of $\mathbf{g}$, we do not have to use the Lagrange polynomials. We only have to be able to solve the univariate polynomial interpolation problem, for example, by using the Vandermonde approach.

We have derived Gordon surfaces as based on polynomial interpolation. Much more generality is available. Equation (22.13) is also true if we use interpolation methods other than polynomial interpolation. The essence of (22.13) may be stated as follows: take a univariate interpolation scheme, apply it to all curves $\mathbf{g}(u,v_j)$ and to all curves $\mathbf{g}(u_i,v)$, add the resulting two surfaces, and subtract the tensor product interpolant that is defined by the univariate scheme. We may replace polynomial interpolation by spline interpolation, in which case we speak of *spline-blended* Gordon surfaces. The basis functions of the univariate interpolation scheme are called *blending functions.*

22.8 Boolean Sums

Our development of Coons patches was quite straightforward, yet it is slightly flawed from a geometric viewpoint. When we derived (22.3), we *added* the two surfaces $\mathbf{r}_c$ and $\mathbf{r}_d$ as an intermediate step. This is illegal—the sum of two surfaces would depend on the choice of a coordinate system (see the discussion in Section 2.1). Although the situation is straightened out by subtracting the bilinear surface $\mathbf{r}_{cd}$, one might ask for a cleaner development. It is provided by the use of *Boolean sums.*

Let us define an operator $\mathcal{P}_1$ that, when applied to a rectangular surface $\mathbf{x}$, returns the ruled surface through $\mathbf{x}(u,0)$ and $\mathbf{x}(u,1)$:

$$[\mathcal{P}_1\mathbf{x}](u,v)=(1-v)\mathbf{x}(u,0)+v\mathbf{x}(u,1).$$

Similarly, we define $\mathcal{P}_2$ to return the ruled surface through $\mathbf{x}(0, v)$ and $\mathbf{x}(1, v)$:

$$[\mathcal{P}_2\mathbf{x}](u, v) = (1 - u)\mathbf{x}(0, v) + u\mathbf{x}(1, v).$$

In terms of Section 22.1, $\mathcal{P}_1$ and $\mathcal{P}_2$ yield the surfaces $\mathbf{r}_c$ and $\mathbf{r}_d$.

We would like to formulate the bilinearly blended Coons patch—which we now call $\mathcal{P}\mathbf{x}$—in terms of $\mathcal{P}_1$ and $\mathcal{P}_2$.

Let us take $\mathcal{P}_1\mathbf{x}$ as a first building block for the Coons patch. Since $\mathcal{P}_1\mathbf{x}$ interpolates only on two boundaries, we will try to add another surface to it, such that the final result will interpolate to all four boundaries. Such a *correction surface* must interpolate to all four boundaries of the error surface $\mathbf{x} - \mathcal{P}_1\mathbf{x}$.[5] It may be obtained by applying $\mathcal{P}_2$ to the error surface. We then obtain

$$\mathcal{P}\mathbf{x} = \mathcal{P}_1\mathbf{x} + \mathcal{P}_2(\mathbf{x} - \mathcal{P}_1\mathbf{x}).$$

This expression for the bilinearly blended Coons patch may be shortened by showing only the involved operators:

$$\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2(\mathcal{I} - \mathcal{P}_1), \tag{22.14}$$

where $\mathcal{I}$ is the identity operator. This means of obtaining one operator $\mathcal{P}$ as a combination of two operators $\mathcal{P}_1, \mathcal{P}_2$ is called a Boolean sum and is often written

$$\mathcal{P} = \mathcal{P}_1 \oplus \mathcal{P}_2. \tag{22.15}$$

Of course, we may also multiply out the terms in (22.14). We then obtain

$$\mathcal{P} = \mathcal{P}_1 \oplus \mathcal{P}_2 = \mathcal{P}_1 + \mathcal{P}_2 - \mathcal{P}_1\mathcal{P}_2.$$

We now see that, even with the use of an operator calculus, we still have the same old Coons patch as defined by (22.3): the term $\mathcal{P}_1\mathcal{P}_2$ is simply the bilinear interpolant to the patch corners.

Let us summarize the essence of the Boolean sum approach: an interpolant to the given data is built by applying $\mathcal{P}_1$. A second operator $\mathcal{P}_2$ is then applied to the "failures" of $\mathcal{P}_1$, and the result is added back to the output from $\mathcal{P}_1$. The interpolant $\mathcal{P}_2$ may actually be of a simpler nature than $\mathcal{P}_1$, since it has to act on only zero data where $\mathcal{P}_1$ was "successful." We can illustrate this for the example of univariate cubic Hermite interpolation: we define $\mathcal{P}_1$ to be the (point-valued) linear interpolant between two points $\mathbf{x}_0$ and $\mathbf{x}_1$ and $\mathcal{P}_2$ to be the (vector-valued)

[5] Note that this error surface is *vector* valued, since both $\mathbf{x}$ and $\mathcal{P}_1\mathbf{x}$ are *point* valued.

cubic Hermite interpolant to a data set $\mathbf{0}, \mathbf{m}_0, \mathbf{m}_1, \mathbf{0}$. Then $\mathcal{P}_1 \oplus \mathcal{P}_2$ is the standard cubic Hermite interpolant.

A note on the notation used in this section: the letter $\mathcal{P}$ that we used to denote our building-block interpolants is due to the term *projector*. A projector is an operator, which, if applied to its own output, will not change the result.[6] For example, $\mathcal{P}_1\mathbf{x}$ is a ruled surface, and $\mathcal{P}_1\mathcal{P}_1\mathbf{x}$ is the same ruled surface. Operators with the property of being projectors are also called *idempotent*.

It was W. Gordon who realized the underlying algebraic structure of Coons patches. That discovery then led him to the generalization that now bears his name (Section 22.7). Boolean sums may be used in the development of many surface interpolation schemes, for an excellent survey, see Barnhill [21].

22.9 Triangular Coons Patches

Just as triangular Bézier patches provide an alternative to the rectangular variety, we may devise a triangular version of Coons patches. Several solutions have been proposed through the years; we will briefly explain the ones by Barnhill, Birkhoff, and Gordon [26] and by Nielson [443].

The C^0 Barnhill, Birkhoff, and Gordon (BBG) approach can be explained as follows. Suppose we are given three boundary curves, as shown in Figure 22.7. We seek a surface that interpolates to all three of them, that is, a *transfinite triangular interpolant*. The construction follows the standard Coons patch development in that it consists of several building blocks, which are then combined in a clever way.

Let us denote[7] the three boundary curves by $\mathbf{x}(0, v, w)$, $\mathbf{x}(u, 0, w)$, $\mathbf{x}(u, v, 0)$. We define three building blocks, each being a ruled surface that interpolates to two boundary curves:

$$\begin{aligned}
\mathcal{P}_1\mathbf{x}(\mathbf{u}) &= (1-r)\mathbf{x}(u,0,w) + r\mathbf{x}(u,v,0); & r &= \tfrac{v}{v+w}, \\
\mathcal{P}_2\mathbf{x}(\mathbf{u}) &= (1-s)\mathbf{x}(u,v,0) + s\mathbf{x}(0,v,w); & s &= \tfrac{w}{w+u}, \\
\mathcal{P}_3\mathbf{x}(\mathbf{u}) &= (1-t)\mathbf{x}(u,0,w) + t\mathbf{x}(0,v,w); & t &= \tfrac{v}{v+u}.
\end{aligned} \tag{22.16}$$

Several combinations of these surfaces yield an interpolant $\mathcal{P}\mathbf{x}$ to all three boundaries: the Boolean sum of any two—for example, $\mathcal{P} = \mathcal{P}_1 \oplus \mathcal{P}_3$—will have that property.

[6] The term comes from geometry: if a 3D object is projected into a plane, we may then repeat that projection, yet it will not change the image.

[7] We use the concept of barycentric coordinates as outlined in Section 3.5.

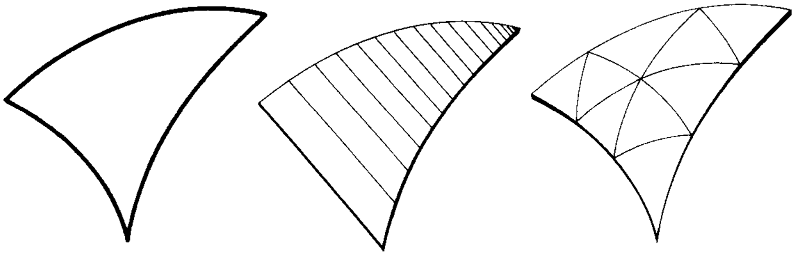


Figure 22.7 BBG interpolation: three boundary curves are given, left. Three ruled surfaces are constructed from them (only one shown, middle). They are combined to yield the final surface, right.

Another possibility is to define $\mathcal{P}$ as a convex combination of the three!$\mathcal{P}_i$:

$$\mathcal{P}\mathbf{x} = u\mathcal{P}_1\mathbf{x} + v\mathcal{P}_2\mathbf{x} + w\mathcal{P}_3\mathbf{x}. \tag{22.17}$$

The building blocks that are used in (22.16) are rational in u, v, w, but they are linear in r, s, t. If we were to incorporate cross-boundary derivative data, that is, to build a C^1 BBG interpolant, we would define $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ to be cubic Hermite interpolants in r, s, t:

$$\begin{aligned}
\mathcal{P}_1\mathbf{x}(\mathbf{u}) &= H_0^3(r)\mathbf{x}(u,0,w) + H_1^3(r)\mathbf{x}_1(u,0,w)\\
&\quad + H_2^3(r)\mathbf{x}_1(u,v,0) + H_3^3(r)\mathbf{x}(u,v,0),\\
\mathcal{P}_2\mathbf{x}(\mathbf{u}) &= H_0^3(s)\mathbf{x}(u,v,0) + H_1^3(s)\mathbf{x}_2(u,v,0)\\
&\quad + H_2^3(s)\mathbf{x}_2(0,v,w) + H_3^3(s)\mathbf{x}(0,v,w),\\
\mathcal{P}_3\mathbf{x}(\mathbf{u}) &= H_0^3(t)\mathbf{x}(0,v,w) + H_1^3(t)\mathbf{x}_3(0,v,w)\\
&\quad + H_2^3(t)\mathbf{x}_3(u,0,w) + H_3^3(t)\mathbf{x}(u,0,w).
\end{aligned} \tag{22.18}$$

The terms $\mathbf{x}_i$ are shorthand for directional derivatives of $\mathbf{x}$ taken in a direction parallel to edge i; more precisely:

$$\begin{aligned}
\mathbf{x}_1(\mathbf{u}) &= (v+w)D_{\mathrm{e}2-\mathrm{e}3}\mathbf{x}(\mathbf{u}),\\
\mathbf{x}_2(\mathbf{u}) &= (u+w)D_{\mathrm{e}3-\mathrm{e}1}\mathbf{x}(\mathbf{u}),\\
\mathbf{x}_3(\mathbf{u}) &= (u+v)D_{\mathrm{e}2-\mathrm{e}1}\mathbf{x}(\mathbf{u}).
\end{aligned}$$

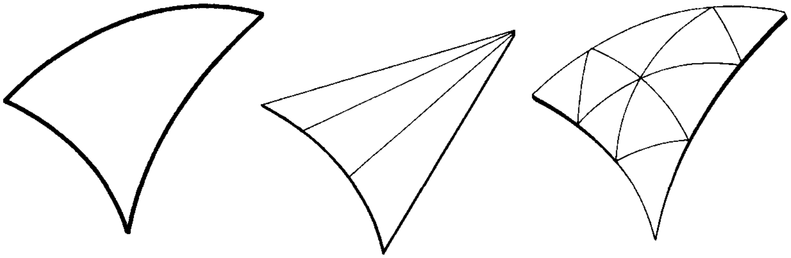


Figure 22.8 Nielson's side vertex method: three boundary curves are given, left. Three ruled surfaces are constructed from them (only one shown, middle). They are combined to yield the final surface, right.

The factors $(v + w)$, and so on, appear because cubic Hermite interpolation is sensitive to interval lengths. A reminder: the terms **e1, e2, e3** refer to points, *not* to edges!

Again, a Boolean sum of any two of the preceding operators will provide a solution—provided the cross-boundary derivatives are compatible (which typically they won't be!).

A different approach is due to G. Nielson [443]. He considers—for a C^0 interpolant—*radial* curves, connecting a patch vertex with a point on the opposite edge, as shown in Figure 22.8. We have

$$\begin{aligned}
\mathcal{P}_1\mathbf{x}(\mathbf{u}) &= u\mathbf{x}(1,0,0) + (1-u)\mathbf{x}(0,r,1-r); \quad r = \frac{v}{v+w},\\
\mathcal{P}_2\mathbf{x}(\mathbf{u}) &= v\mathbf{x}(0,1,0) + (1-v)\mathbf{x}(1-s,0,s); \quad s = \frac{u}{u+w},\\
\mathcal{P}_3\mathbf{x}(\mathbf{u}) &= w\mathbf{x}(0,0,1) + (1-w)\mathbf{x}(t,1-t,0); \quad t = \frac{u}{u+v}.
\end{aligned} \tag{22.19}$$

The final interpolant may then be written as a *triple Boolean sum:*

$$\begin{aligned}
\mathcal{P} &= \mathcal{P}_1 \oplus \mathcal{P}_2 \oplus \mathcal{P}_3\\
&= \mathcal{P}_1 + \mathcal{P}_2 + \mathcal{P}_3\\
&\quad - \mathcal{P}_1\mathcal{P}_2 - \mathcal{P}_1\mathcal{P}_3 - \mathcal{P}_2\mathcal{P}_3\\
&\quad + \mathcal{P}_1\mathcal{P}_2\mathcal{P}_3.
\end{aligned}$$

To make this scheme C^1, we again replace the linear interpolants in (22.19) by cubic Hermite interpolants, now with directional derivatives supplied in the radial directions.

For more literature on triangular Coons-type interpolants, consult the following: Barnhill [20], Barnhill and Gregory [33], [32], Gregory and Charrot [292], Marshall and Mitchell [420], Lacombe and Bédard [367], and Nielson [444].

22.10 Problems

1 Show that the bilinearly blended Coons patch is not in the convex hull of its boundary curves. Is this a good or a bad property?

2 Verify the caption to Figure 22.2 algebraically.

3 Equation (22.9) generates tangent ribbons from the given boundary curve network. Verify that the resulting surface does not suffer from twist incompatibilities.

***4** Translational surfaces have zero twists. Show that the inverse statement is also true: every surface with identically vanishing twists is a translational surface.

***5** Show that the bilinearly blended Coons patch, when applied to cubic boundary curves, yields a bicubic patch.

***6** Show that Adini's twist from Section 16.3 is the twist of the bilinearly blended Coons patch for the four boundary cubics.

***7** As we have seen, two adjacent bilinearly blended Coons patches are not C^1 in general. What are the conditions for the boundary curves of the two patches such that the Coons patches *are* C^1?

P1 Use the data set `car.dat`. Interpolate all four boundaries using uniform B-spline interpolation. Then compute the bilinearly blended Coons patch. Next, experiment with different blending functions and discuss how they change the shape of the surface.

This Page Intentionally Left Blank

Shape 23

We discussed many methods for curve and surface generation. In this chapter, we shall discuss some ways to inspect the geometric quality of those curves and surfaces, and develop a few ideas on how to remove shape imperfections. There is a growing interest in this area; see for example, the collection of Sapidis [532].

23.1 Use of Curvature Plots

A spline curve is typically obtained in one of two ways: as a curve that interpolates to given data points, or as the result of interactive manipulation of a B-spline polygon. In both cases, it is hard to tell from the display on the screen if the shape of the curve is acceptable or not: two curves may look identical on the screen, yet reveal significant shape differences when plotted to full scale. Such plots are both expensive and time consuming—one needs a tool to analyze curve shape at the CAD terminal.

The most commonly used tool for this task is provided by the *curvature plot* of the curve. For a given curve, we can plot curvature versus arc length or versus the parameter. The resulting graph is the curvature plot. We have already used curvature plots in Chapter 9. All three curves from Figures 9.8, 9.10, and 9.12 look very similar, yet their curvature plots reveal substantial differences. The same is true for Figures 9.16 and 9.18. What actually constitutes a "substantial" difference depends on the application at hand, of course.

The curvature of a space curve is nonnegative by definition (10.7).[1] Very often, we are interested in the detection of inflection points of the current planar projection, that is, the points of inflection of the curve as it appears on the screen. If we introduce signed curvature by

$$\kappa(u) = \frac{\ddot{x}(u)\dot{y}(u) - \ddot{y}(u)\dot{x}(u)}{[(\dot{x}(u))^2 + (\dot{y}(u))^2]^{3/2}}, \tag{23.1}$$

where x, y are the two components of the curve, it is easy to point to changes in the sign of curvature, which indicate inflection points. (Those sign changes can be marked by special symbols on the plot.) Signed curvature is used in all examples in this book.

We now go one step further and use curvature plots for the definition of fair curves: *a curve is fair if its curvature plot is continuous and consists of only a few monotone pieces.*[2] Regions of monotone curvature are separated by points of extreme curvature. The number of curvature extrema of a fair curve should thus be small—curvature extrema should occur only where explicitly desired by the designer, and nowhere else!

This definition of fairness (also suggested by Dill [169], Birkhoff [64], and Su and Liu [583] in similar form) is certainly subjective; however, it has proved to be a practical concept. Once a designer has experienced that "flat spots" on the curve correspond to "almost zero" curvature values and that points of inflection correspond to crossings of the zero curvature line, he or she will use curvature plots as an everyday tool.

An interesting alternative to plain curvature plots are plots of the logarithm of curvature; see [99]. Such plots highlight "flat" areas on a curve. Tiny changes in curvature have a significant effect in these regions, and they are magnified by the use of logarithms.

Figure 23.1 shows several curvature plots. They are obtained from the curves in Figures 9.1 through 9.4. Note how "bumpy" the cubic case with only nine intervals is.

[1] See also the Problems at the end of Chapter 13.

[2] M. Sabin has suggested that "a frequency analysis of the radius of curvature plotted against arc length might give some measure of fairness—the lower the dominant frequency, the fairer the curve." Quoted from Forrest [240].

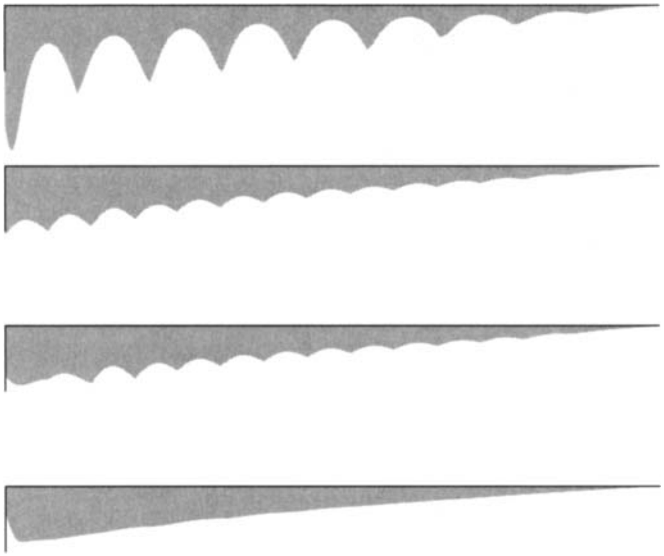


Figure 23.1 Curvature plots: the curvatures of four spiral-shaped curves. (Courtesy M. Jeffries.)

23.2 Curve and Surface Smoothing

A typical problem in the design process of many objects is that of *digitizing errors*: data points have been obtained from some digitizing device (a tablet being the simplest), and a fair curve is sought through them. In many cases, however, the digitized data are inaccurate, and this presence of digitizing error manifests itself in a "rough" curvature plot of an interpolating spline curve.[3]

For a given curvature plot of a C^2 cubic spline, we may now search for the largest slope discontinuity of $\kappa(s)$ (s being arc length) and try to "fair" the curve there. Let this largest slope discontinuity occur at $u = u_j$. The slope κ' is given by

$$\frac{\mathrm{d}\kappa}{\mathrm{d}s} = \frac{\det[\dot{\mathbf{x}}, \dddot{\mathbf{x}}]}{||\dot{\mathbf{x}}||^4} - 3\dot{\mathbf{x}}\ddot{\mathbf{x}}\frac{\det[\dot{\mathbf{x}}, \ddot{\mathbf{x}}]}{||\dot{\mathbf{x}}||^6}, \tag{23.2}$$

where, as usual, dots denote derivatives with respect to the given parameter u (see Pottmann [487]). Note that this formula applies for 2D curves only.

[3] Typically, splines that are obtained from interactive adjustment of control polygons exhibit rough curvature plots as well.

The data point $\mathbf{x}(u_j)$ is potentially in error; so why not move $\mathbf{x}(u_j)$ to a more favorable position? It seems that a more favorable position should be such that the spline curve through the new data no longer exhibits a slope discontinuity.

We now make the following observation: if a spline curve is three (instead of just two) times differentiable at a point $\mathbf{x}(u_j)$, then certainly its curvature is differentiable at u_j; that is, it does not have a slope discontinuity there (assuming that the tangent vector does not vanish at u_j). Moreover, the two cubic segments corresponding to the intervals (u_{j-1}, u_j) and (u_j, u_{j+1}) are now part of *one* cubic segment: the knot u_j is only a pseudoknot, which could be removed from the knot sequence without changing the curve.

We will thus try to *remove* the "offending knot" u_j from the knot sequence, thereby fairing the curve, and then reinsert the knot in order to keep a spline curve with the same number of intervals as the initial one. We discussed knot insertion in Chapter 8. The inverse process, *knot removal*, has no unique solution. Several possibilities are discussed in Eck and Hadenfeld [185], Farin et al. [214], and Farin and Sapidis [215], [533]. We present here a simple yet effective solution to the knot removal problem. Let the offending knot u_j be associated with the vertex $\mathbf{d}_j$.[4] We now formulate our knot removal problem: to what position $\hat{\mathbf{d}}_j$ must we move $\mathbf{d}_j$ such that the resulting new curve becomes three times differentiable at u_j? After some calculation (equating the left and the right third derivative of the new spline curve), we verify that the new vertex $\hat{\mathbf{d}}_j$ is given by

$$\hat{\mathbf{d}}_j = \frac{(u_{j+2} - u_j)\mathbf{l}_j + (u_j - u_{j-2})\mathbf{r}_j}{u_{j+2} - u_{j-2}},$$

where the auxiliary points $\mathbf{l}_j, \mathbf{r}_j$ are given by

$$\mathbf{l}_j = \frac{(u_{j+1} - u_{j-3})\mathbf{d}_{j-1} - (u_{j+1} - u_j)\mathbf{d}_{j-2}}{u_j - u_{j-3}}$$

and

$$\mathbf{r}_j = \frac{(u_{j+3} - u_{j-1})\mathbf{d}_{j+1} - (u_j - u_{j-1})\mathbf{d}_{j+2}}{u_{j+3} - u_j}.$$

The geometry underlying these equations is illustrated in Figure 23.2.

The faired curve now differs from the old curve between $\mathbf{x}(u_{j-2})$ and $\mathbf{x}(u_{j+2})$— thus this fairing procedure is *local*.

[4] This is a slight deviation from the notation of Chapter 8.

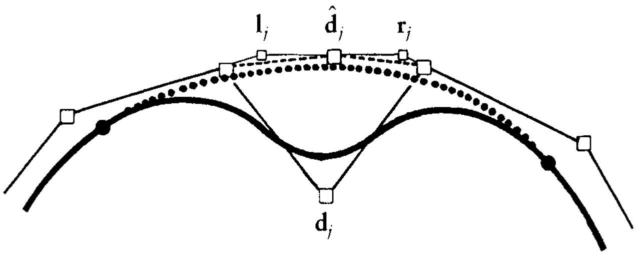


Figure 23.2 Knot removal: if $\mathbf{d}_j$ is moved to $\hat{\mathbf{d}}_j$, the new curve is three times differentiable at u_j.

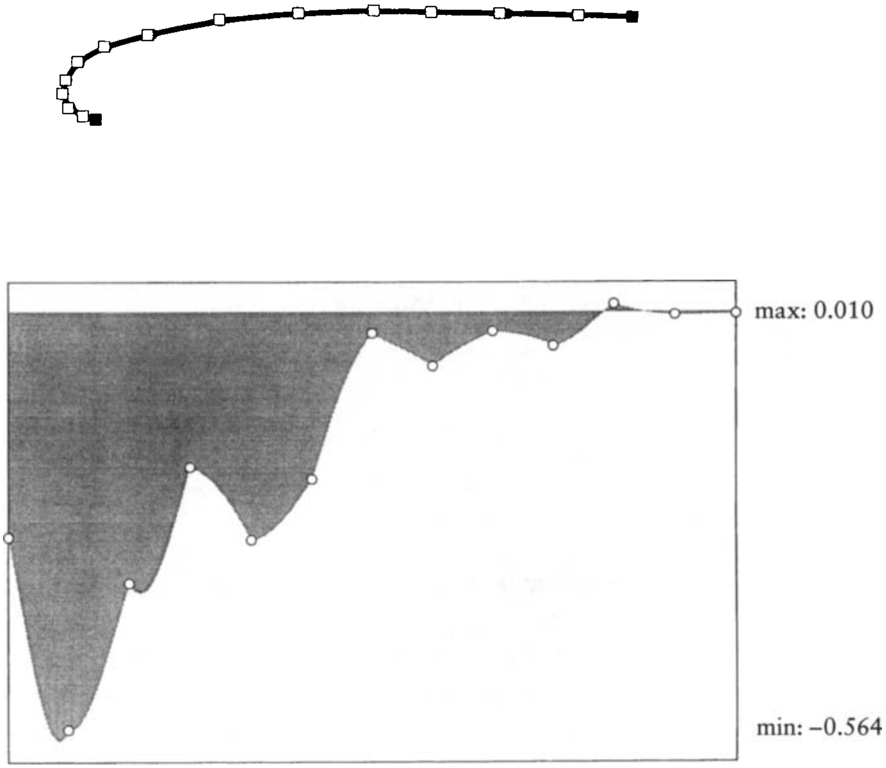


Figure 23.3 Curve smoothing: an initial curve and B-spline polygon with its curvature plot.

Figures 23.3 and 23.4 illustrate an application of this algorithm, although it is not used locally, but for all knots. Note that the initial and the smoothed curves look almost identical, and only their curvature plots reveal significant shape differences. The initial curve has an inflection point, which is not visible

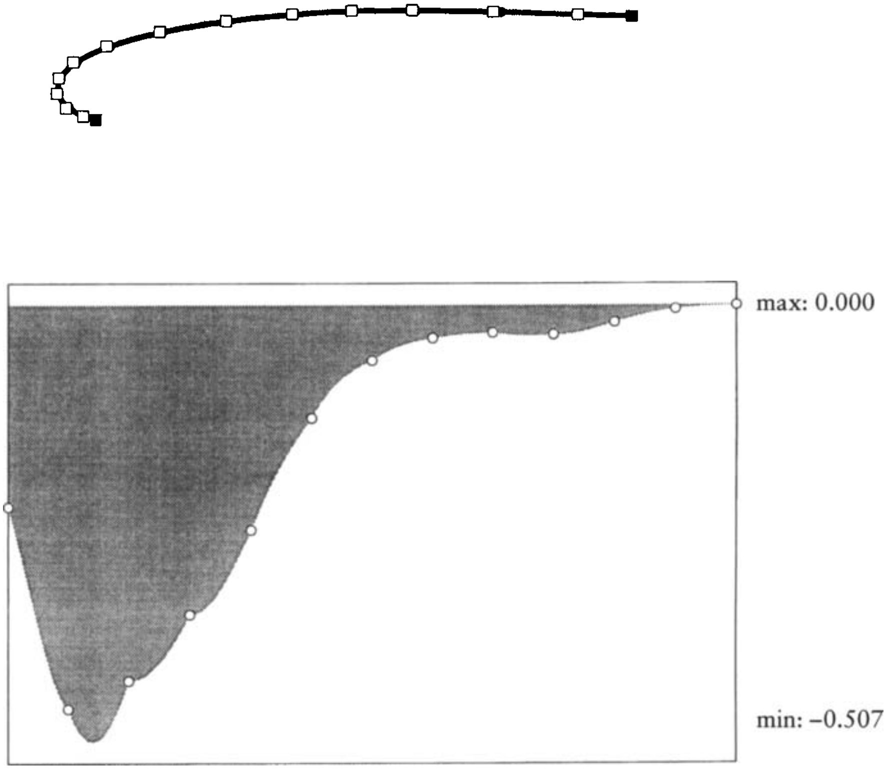


Figure 23.4 Curve smoothing: the smoothed curve and its curvature plot.

without the use of curvature plots. The faired curve does not have this shape defect any more.

In practice, the improved vertex $\hat{\mathbf{d}}_j$ may be farther away from the original vertex $\mathbf{d}_j$ than a prescribed tolerance allows. In that case, we restrict a realistic $\hat{\mathbf{d}}_j$ to be in the direction toward the optimal $\hat{\mathbf{d}}_j$, but within tolerance to the old $\mathbf{d}_j$.

Other methods for curve fairing exist. We mention Kjellander's method [358], which moves a data point to a more favorable location and then interpolates the changed data set with a C^2 cubic spline. This method is global. A method that fairs only data points, not spline curves, is presented by Renz [507]. This method computes second divided differences, smoothes them, and "integrates" back up. Methods that aim at the smoothing of single Bézier curves are discussed by Hoschek [333], [335]. Variations on the described method are given in [215].

A method that tries to reduce the degrees of each cubic segment to quadratic is given in [200].

23.3 Surface Interrogation

Curvature plots are useful for curves; it is reasonable, therefore, to investigate the analogous concepts for surfaces. Several authors have done this, including Beck et al. [49], Farouki [219], Dill [169], Munchmeyer [435], [434], and Forrest [242]. An interesting early example is on page 197 of Hilbert and Cohn-Vossen [323]. Surfaces have two major kinds of curvature: Gaussian and mean; see Section 19.6. Both kinds can be used for the detection of surface imperfections. Another type of curvature can be useful, too: *absolute curvature* κ_{abs}. It is defined by

$$\kappa_{\text{abs}} = |\kappa_1| + |\kappa_2|,$$

where κ_1 and κ_2 are the maximal and minimal normal curvatures at the point under consideration.

Gaussian curvature does not offer much information about generalized cylinders of the form

$$\mathbf{c}(u, v) = (1 - u)\mathbf{x}(v) + u[\mathbf{x}(v) + \mathbf{v}].$$

Even if the generating curve $\mathbf{x}(v)$ is highly curved, we still have $K \equiv 0$ for these surfaces. A similar statement can be made about the mean curvature H, which is always zero for minimal surfaces, no matter how complicated.

Color Plates IV and V illustrate the use of curvatures in nonengineering applications. Plate IV shows the digitized model of a bone (digitized as a mesh and locally fitted with Bézier patches) and a color coding of the absolute curvature: where the curvature is high, the surface is "painted" red, and where it is low, it is "painted" blue. This process is referred to as *texture mapping* in computer graphics.

Color Plate V shows an application in archeology: a digitized vessel (Native American) is represented as a triangle mesh (left, with original texture). A cross section is computed, fitted with a B-spline curve, and its curvature is displayed (middle). Finally, Gaussian curvature is used as a texture map (right).

Another method for surface interrogation is the use of *reflection lines*, first described in Klass [360]. Poeschl [486] introduced a simplified method, *isophotes*. Reflection lines are a standard surface interrogation tool in the styling shop of a car manufacturer. They are the pattern that is formed on the polished car surface by the mirror images of a number of parallel fluorescent strip lights. If the

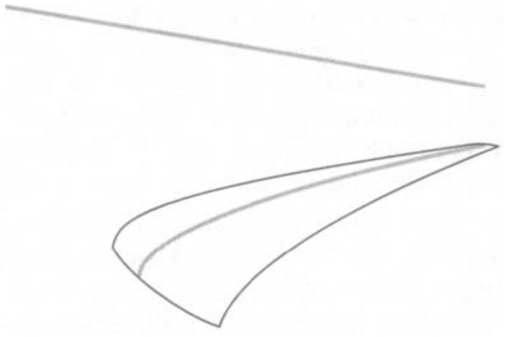


Figure 23.5 Isophotes: a line light source (top) is reflected by a surface.

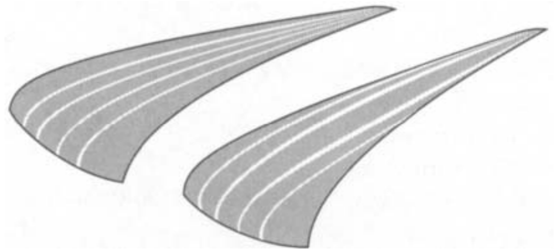


Figure 23.6 Isophotes: left, a surface with "perfect" isophotes; right, after a perturbation was applied to the surface.

mirror images are "nice," then the corresponding surface is deemed acceptable. Whereas reflection lines depend on the position of an observer, isophotes consider only the angle formed between surface normals and light source. The principle is illustrated in Figure 23.5.

Reflection lines and isophotes can easily be simulated on a raster graphics device (mark points whose normal points to one of the light sources). With some more effort, they can also be computed on a line drawing device (see Klass [360]).

Figure 23.6 shows a surface with several isophotes and the effect that a small perturbation can have on them.

Reflection lines and curvature "paintings" have different usages: reflection lines are not as fine-tuned as curvatures; they are prone to miss local shape defects of a surface.[5] On the other hand, curvatures of a surface may look perfect, yet it might not have a "pleasant" overall shape—reflection lines have a better chance of flagging global imperfections.

[5] This is because reflection lines may be viewed as a first-order interrogation tool (involving only first derivatives), whereas curvature plots are second-order interrogation tools.

Once imperfections are detected in a tensor product B-spline surface, we would want methods to remove them without time-consuming interactive adjustment of control polygons. One such method is to apply the curve-smoothing method from Section 23.2 in a tensor product way: smooth all control net rows, then all control net columns. The resulting surface is usually smoother than the original surface. Figure 23.6 is an example of this method: the right figure is a B-spline surface; four iterations of the program fair_surf produced the figure on the left.

More involved methods for surface fairing exist; they aim for the enforcement of convexity constraints in tensor product spline surfaces. We mention Andersson et al. [8], Jones [348], and Kaufmann and Klass [354].

23.4 Implementation

The routine curvatures may be used to generate curvature values of a rational Bézier curve. It writes the values into a file that might be read by another program that generates a curvature plot.

To compute the curvature at the parameter value t, the curve is subdivided using the (rational) de Casteljau algorithm. Of the two subpolygons that are generated, the larger one is selected, and its beginning curvature is computed. Since the subdivision routine rat_subdiv orders both subpolygons beginning at the subdivision point, only one curvature routine curvature_0 is needed.

```
void curvatures(coeffx,coeffy,weight,degree,dense)
/*    writes  signed curvatures of  a rational  Bezier curve into
      a file.
input:
     coeffx, coeffy:  2D Bezier polygon
     weight:          the weights
     degree:          the degree
     dense:           how many curvature values to compute
output:
                      written into file outfile
*/
```

The routine curvature_0 is a simple application of (10.10):

```
float curvature_0(bez_x,bez_y,weight,degree)
/* computes curvature of rational  Bezier curve at t=0
   Input: bez_x, bez_y, weight: control polygon and weights
          degree:               degree of curve
*/
```

The following area routine is included for completeness:

```
float area(p1,p2,p3)
/* find area of triangle p1,p2,p3 */
```

Note, however, that area returns a negative value if the input points are ordered clockwise!

The following routine generates the "raw data" that are needed to create the curvature plot of a rational B-spline curve. Of course, we may simply set all weights to unity for the polynomial case.

```
void bspl_kappas(bspl_x,bspl_y,bspl_w,knot,l,dense)
/*      writes curvatures of cubic rational  B-spline curve into
        a file.
  input:
        bspl_x,bspl_y:  2D rat. B-spline polygon
        bspl_w:         the  B-spline weights
        knot:           the knot sequence
        dense:          how many curvature values to compute per interval
        l:              no. of intervals
  output:
                        written into file outfile
*/
```

The preceding programs are used by the main program plot_b_kappa.c in order to produce Postscript output for a curvature plot.

Now the programs to fair curves and surfaces. First, the curve case:

```
void fair_bspline(bspl,knot,l,from,to)
/* Fairs a cubic rational B-spline curve by knot removal/reinsertion.

Input:  bspl:       cubic B-spline control polygon (one coord.)
        knot:       knot sequence
        l:          no. of intervals
        from, to:   from where to where to fair
Output: same as input, but hopefully fairer.
*/
```

Second, the surface case:

```
void fair_surf(bspl,lu,lv,knot_u, knot_v)
/*      Fairs B-spline control net.
```

```
Input:  bspl:   B-spline control net (one coordinate only)
        lu,lv:  no. of intervals in u- and v-direction
        knots_u, knots_v:   knot vectors in u- and v-direction
Output: as input

Note:   Has to be called once for each x-,y-,z- coordinate.
```

23.5 Problems

1 Show that a planar cubic curve may have two points of inflection, that is, points where curvature changes sign.

2 Show that a true space cubic cannot have any points with zero curvature.

P1 The routine `curvatures` produces a file that contains pairs t_i, κ_i; that is, it can be used to plot curvature versus parameter. Modify the program so it can be used to produce plots of curvature versus arc length.

P2 Write a program to compute the torsion of a Bézier or a spline curve. Then produce torsion plots as an additional interrogation tool for space curves.

P3 Compute the curvatures of isoparametric curves of a spline surface, color code them, and use them as an interrogation tool.

P4 Produce a uniform B-spline surface that interpolates the four boundary data sets from `car.dat`. Test if that surface is fair (using P3 if you want); if not, improve its shape by using `fair_surf`.

This Page Intentionally Left Blank

Evaluation of Some Methods 24

In this chapter, we will examine some of the many methods that have been presented. We will try to point out the relative strengths and weaknesses of each, a task that is necessarily influenced by personal experience and opinion.

24.1 Bézier Curves or B-Spline Curves?

Taken at face value, this is a meaningless question: Bézier curves are a special case of B-spline curves. Any system that contains B-splines in their full generality, allowing for multiple knots, is capable of representing a Bézier curve or, for that matter, a piecewise Bézier curve.

In fact, several systems use both concepts simultaneously. A curve may be stored as an array holding B-spline control vertices, knots, and knot multiplicities. For evaluation purposes, the curve may then be converted to the piecewise Bézier form.

24.2 Spline Curves or B-Spline Curves?

This question is often asked, yet it does not make much sense. B-splines form a basis for all splines, so any spline curve can be written as a B-spline curve. What is often meant is the following: if we want to design a curve, should we pass an interpolating spline curve through data points, or should we design a curve by interactively manipulating a B-spline polygon? Now the question has become one concerning curve *generation* methods rather than curve *representation* methods.

A flexible system should have both: interpolation and interactive manipulation. The interpolation process may of course be formulated in terms of B-splines. Since many designers do not favor interactive manipulation of control polygons, we should allow them to generate curves using interpolation. Subsequent curve modification may also take place without display of a control polygon: for instance, the designer might move one (interpolation) point to a new position. The system could then compute the B-spline polygon modification that would produce exactly that effect. So a user might actually work with a B-spline package, but a system that is adapted to his or her needs might hide that fact. See Section 9.3 for details.

We finally note that every C^2 B-spline curve may be generated as an interpolating spline curve: read off junction points, end tangent vectors, and knot sequence from the B-spline curve. Feed these data into a C^2 cubic spline interpolator, and the original curve will be regenerated.

24.3 The Monomial or the Bézier Form?

We have made the point in this book that the monomial form[1] is *less geometric* than the Bézier form for a polynomial curve. A software developer, however, might not care much about the beauty of geometric ideas—in the workplace, the main priority is performance. Since the fundamental work by Farouki and Rajan [224], [225], [220], one important performance issue has been resolved: the Bézier form is *numerically more stable* than the monomial form. Farouki and Rajan observed that numerical inaccuracies, unavoidable with the use of finite precision computers, affect curves in the monomial form significantly more than those in Bézier form. More precisely, they show that the condition number of simple roots of a polynomial[2] is smaller in the Bernstein basis than in the monomial basis. If one decides to use the Bézier form for stability reasons, then it is essential that no conversions be made to other representations; these will destroy the accuracy gained by the use of the Bézier form. For example, it is not advisable from a stability viewpoint to store data in the monomial form and to convert to Bézier form to perform certain operations. More details are given in Daniel and Daubisse [132].

Figure 24.1 shows a numerical example, carried out using the routine bez_to_pow with single precision arithmetic. A degree 18 Bézier curve (top) was converted

[1] This form is also called the *power basis form.*

[2] This number indicates by how much the location of a root is perturbed as a result of a perturbation of the coefficients of the given polynomial.

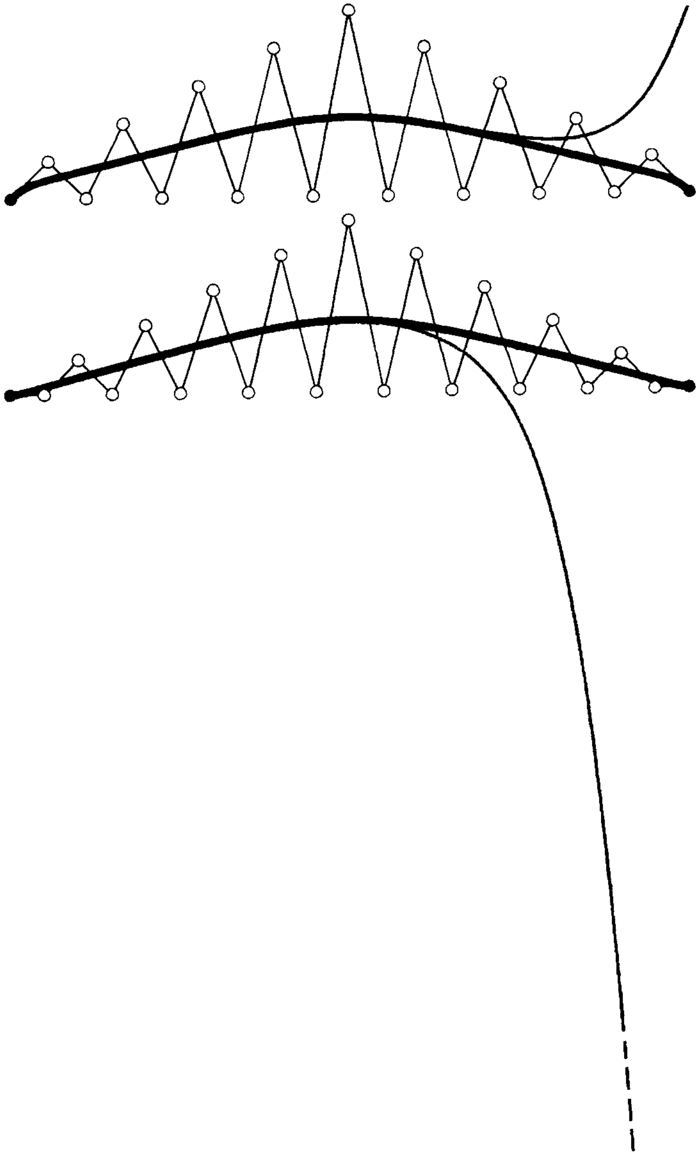


Figure 24.1 Stability of the monomial form: slight perturbations in the coefficients affect the monomial form (gray) much more than the Bézier form (black). Top: a degree 18 curve; bottom: a degree 20 curve.

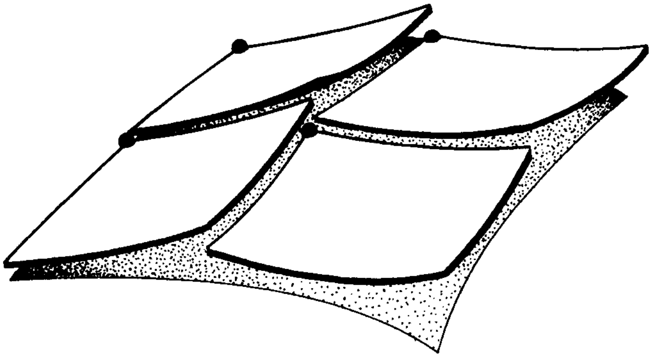


Figure 24.2 A piecewise monomial surface: the patches miss the points (1,1) due to roundoff.

to the monomial form. Then, the coefficients of the Bézier and of the monomial form were perturbed by a random error, less than 0.001 in each coordinate. (The x-values of the control polygon extend from 0 to 20.) The Bézier curve shows no visual sign of perturbation, whereas the monomial form is not very reliable near $t = 1$. The experiment was then repeated for a degree 20 curve (bottom) with even more disastrous results. Although degrees like 18 or 20 look high, we should not forget that such degrees already appear in harmless-looking tensor product surfaces: for example, the diagonal $u = v$ of a patch with $n = m = 9$ is of degree 18!

As a consequence of its numerical instability, the monomial form is not very reliable for the representation of curves or surfaces. For the case of surfaces, Figure 24.2 gives an illustration (in somewhat exaggerated form). Since the monomial form is essentially a Taylor expansion around the local coordinate (0,0) of each patch, it is quite close to the intended surface there. Farther away from (0,0), however, roundoff takes its toll. The point (1,1) is *computed* and therefore missed. For an adjacent patch, the actual point is *stored* as a patch corner, thus giving rise to the discontinuities shown in Figure 24.2. The significance of this phenomenon increases dramatically when curves or surfaces of high degrees are used.

Let's not forget to mention the main attraction of the monomial form: speed. Horner's method is faster than the de Casteljau algorithm; it is also faster than the routine hornbez. There is a tradeoff therefore between stability and speed. (Given modern hardware, things are not quite that clear-cut, however: T. DeRose and T. Holman [166] have developed a multiprocessor architecture that hardwires the de Casteljau algorithm into a network of processors and now outperforms Horner's method.)

24.4 The B-Spline or the Hermite Form?

Cubic B-spline curves are numerically more stable than curves in the piecewise cubic Hermite form. This comes as no surprise, since some of the Hermite basis functions are negative, giving rise to numerically unstable nonconvex combinations. However, there is an argument in favor of the piecewise Hermite form: it stores interpolation points explicitly. In the B-spline form, they must be computed. Even if this computation is stable, it may produce roundoff.

A significant argument against the use of the Hermite form points to its lack of invariance under affine parameter transformations. Everyone who has programmed the Hermite form has probably experienced the trauma resulting from miscalculated tangent lengths. A programmer should not be burdened with the subtleties of the interplay between tangent lengths and parameter interval lengths.

An important advantage of the B-spline form is *storage*. For B-spline curves, one needs a control point array of length $L+2$ plus a knot vector array of length $L+1$ for a curve with L data points, resulting in an overall storage requirement of $4L+7$ reals.[3] For the piecewise Hermite form, one needs a data array of length $2L$ (position and tangent at each data point) plus a knot vector of length $L+1$, resulting in a storage requirement of $7L+1$ reals. For surfaces (with same degrees in u and v for simplicity), the discrepancy becomes even larger: $3(L+2)^2+2(L+1)$ versus $12L^2+2(L+1)$ reals. (For the Hermite form, we have to store position, u- and v-tangents, and twist for each data point.)

When both forms are used for tensor product interpolation, the Hermite form must solve three sets of linear equations (see Section 15.5) while the B-spline form must solve only two sets (see Section 15.4).

24.5 Triangular or Rectangular Patches?

Most of the early CAD efforts were developed in the car industry, and this is perhaps the main reason for the predominance of rectangular patches in most CAD systems; the first applications of CAD methods to car body design were to the outer panels such as roof, doors, and hood. These parts basically have a rectangular geometry, hence it is natural to break them down into smaller rectangles. These smaller rectangles were then represented by rectangular patches.

Once a CAD system had been successfully applied to a design problem, it seemed natural to extend its use to other tasks: the design of the interior car

[3] We are not storing knot multiplicities. We would then be able to represent curves that are only C^0, which the cubic Hermite form is not capable of.

body panels, for instance. Such structures do not possess a notably rectangular structure, and rectangular patches are therefore not a natural choice for modeling these complicated geometries. However, rectangle-based schemes already existed, and the obvious approach was to make them work in "unnatural" situations also. They do the job, although with some difficulties, which arise mainly in the case of degenerate rectangular patches.

Triangular patches do not suffer from such degeneracies and are thus better suited to describe complex geometries than are rectangular patches. It seems obvious, therefore, to advise any CAD system developer to add triangular patches to the system.

The French car company Citroën used triangular patches in the early 1960s, but later abandoned their use. In the middle 1980s, Evans and Sutherland used triangular patches as the backbone of their CDDRS system—they were subsequently replaced by B-spline surfaces. More recently, NVIDIA uses triangular patches to fill in holes in rectangular topologies.

Quick Reference of Curve and Surface Terms

ab initio design Latin: from the beginning. Used to describe design processes in which the designer inputs his or her ideas directly into the computer, without constraints such as interpolatory constraints.

Affine combination Same as a barycentric combination.

Affine invariance A property of a curve or surface generation scheme: the same result is obtained if computation of a point on a curve or surface occurs before or after an affine map is applied to the input data.

Affine map Any map comprising translations, rotations, scalings, and shears. Maps parallels to parallels. Leaves ratios of collinear points unchanged.

Approximation Fitting a curve or surface to given data. As opposed to interpolation, the curve or surface approximation only has to be close to the data.

Barycentric combination A weighted average where the sum of the weights equals one.

Barycentric coordinates A point in $\mathbb{E}^2$ may be written as a unique barycentric combination of three points. The coefficients in this combination are its barycentric coordinates.

Basis function Functions form linear spaces, which have bases. The elements of these bases are the basis functions.

Bernstein polynomial The basis functions for Bézier curves.

Beta-spline curve A G^2 piecewise cubic curve that is defined over a uniform knot sequence.

Bézier curve A polynomial curve that is expressed in terms of Bernstein polynomials.

Bézier polygon The coefficients in the expansion of a Bézier curve in terms of Bernstein polynomials are points. Connected according to their natural numbering, they form the Bézier polygon.

Bilinear patch A patch that is ruled in two directions. Or: a hyperbolic paraboloid.

Blossom A multivariate polynomial that is associated with a given polynomial through the process of blossoming.

Blossoming The procedure of applying n (the polynomial degree) de Casteljau algorithm steps or n de Boor steps to a polynomial (or to a segment of a spline curve), but each one for a different parameter value.

B-spline A piecewise polynomial function. It is defined over a knot partition, has local support, and is nonnegative. If a spline curve is expressed in terms of B-splines, it is called a B-spline curve.

B-spline polygon The coefficients in the expansion of a B-spline curve in terms of B-splines are points. Connected according to their natural numbering, they form the B-spline polygon. Also called de Boor polygon.

Breakpoint Same as a knot.

Butterfly scheme An interpolatory recursive subdivision scheme that creates smooth surfaces from a given set of triangulated points.

CAGD Computer aided geometric design.

Catmull–Clark Surfaces Subdivision surfaces that are bicubic spline surfaces when the input mesh is rectangular.

C^r A smoothness property of curves or surfaces: being r times differentiable with respect to the given parametrization.

Chord length parameters In many curve interpolation problems, data points need to be assigned parameter values. If these are spaced relative to the spacing of the data points, we have chord length parameters.

Collinear Being on a straight line.

Compatibility For some interpolation problems, the input data may not be arbitrary but must satisfy some consistency constraints, called compatibility conditions.

Conic section The intersection curve between a cone and a plane. Or: the projective image of a parabola. A nondegenerate conic is an ellipse, a parabola, or a hyperbola.

CONS Curve on surface.

Control polygon See Bézier polygon or B-spline polygon.

Control vector For rational curves, a Bézier or B-spline control point that has degenerated to a vector, implying a zero weight.

Convex curve A planar curve that is a subset of the boundary of its convex hull.

Convex hull The smallest convex set that contains a given set.

Convex set A set such that the straight line segment connecting any two points of the set is completely contained within the set.

Coons patch A patch that is fitted between four arbitrary boundary curves.

Coplanar Being on the same plane.

Cross plot Breaking down the plot of a parametric curve into the plots of each coordinate function.

Cross ratio A quantity computed from four collinear points, invariant under projective maps. A generalization of affine ratios.

Curvature At a point on a curve, curvature is the inverse of the radius of the osculating circle. Also: curvature measures by how much a curve deviates from a straight line at a given point.

Curve The path of a point moving through space. Or: the image of the real line under a continuous map.

de Boor algorithm The algorithm that recursively computes a point on a B-spline curve.

de Casteljau algorithm The algorithm that recursively computes a point on a Bézier curve.

Decimation Reducing the number of triangles in a triangulation while staying close to the initial geometry.

Delaunay triangulation A triangulation that maximizes the minimal angle of all triangles. Or: the dual of the Dirichlet tessellation.

Developable surface A ruled surface whose Gaussian curvature vanishes everywhere.

Direct G^2 splines G^2 piecewise cubics that are generated by specifying a control polygon and some Bézier points.

Dirichlet tessellation A partition of $\mathbb{E}^2$ or $\mathbb{E}^3$ into convex tiles. Each tile is associated with a given data point such that all of its points are closer to "its" data point than to any other data point.

Domain The preimage of a curve or surface.

Doo–Sabin surfaces Subdivision surfaces that are biquadratic spline surfaces when the input mesh is rectangular.

End condition In cubic spline curve interpolation, one has to supply an extra condition at each of the two endpoints. Examples of such end conditions: prescribed tangents or zero curvature.

Four-point scheme An interpolatory curve subdivision scheme.

Frenet frame At each point of a (nondegenerate) curve, the first, second, and third derivative vectors are linearly independent. Applying Gram–Schmidt orthonormalization to them yields the Frenet frame of the curve at the given point.

Functional curve or surface A curve of the form $y = f(x)$ or a surface of the form $z = f(x, y)$.

G^2 spline curve A C^1 piecewise cubic curve that is twice differentiable with respect to arc length.

γ-spline A G^2 spline that is C^1 over a given knot sequence.

Geometric continuity Smoothness properties of a curve or a surface that are more general than its order of differentiability.

Gordon surface A generalization of Coons patches. Interpolates to a rectilinear network of curves.

Hermite interpolation Generating a curve or surface from data that consist of points and first and/or higher derivatives.

Hodograph The first derivative curve of a parametric curve.

Homogeneous coordinates A coordinate system that is used to describe rational curves and surfaces in terms of projective geometry, where they are just polynomial.

Horner's scheme An efficient method to evaluate a polynomial in monomial form by nested multiplication.

IGES Initial Graphics Exchange Specification. A popular data specification format, aiming at unifying geometry descriptions.

Infinite control point Same as control vector.

Inflection point A point on a curve where the tangent intersects the curve. Often corresponds to points with zero curvature.

Interior Bézier points For curves, those Bézier points that are not junction points. For surfaces, those Bézier points that are not boundary points.

Interpolation Finding a curve or surface that satisfies some imposed constraints exactly. The most common constraint is the requirement of passing through a set of given points.

Junction point A spline curve comprises segments. The common point shared by two segments is called the junction point. See also knot.

Knot A spline curve is defined over a partition of an interval of the real line. The points that define the partition are called knots. If evaluated at a knot, the spline curve passes through a junction point.

Knot insertion Adding a new knot to the knot sequence of a B-spline curve without changing the graph of the curve.

Lagrange interpolation Finding a polynomial curve through a given set of data points.

Least squares An approximation process that aims at minimizing the deviations of given data points from a desired curve or surface.

Linear precision A property of many curve schemes: if the curve generation scheme is applied to data read off from a straight line, that straight line is reproduced.

Local control A curve or surface scheme has the local control property if a change in the input data only changes the curve or surface in a region near the changed data.

Lofting Creating a ruled surface between two given curves.

Minmax box Smallest 2D or 3D box with edges parallel to the coordinate axes that completely contains a given object.

Monomial form A polynomial is in monomial form if it is expressed in terms of the monomials $1, t, t^2, \ldots$.

Multiresolution Breaking down an object into a sequence of approximations with increasing accuracy.

Node A term that is used inconsistently in the literature: it sometimes refers to a knot, sometimes to a control point.

NURB Nonuniform rational B-spline curve or surface.

ν-spline An interpolating G^2 spline curve that is C^1 over a given knot sequence.

Osculating circle At a given point, the osculating circle approximates the curve "better" than any other circle.

Osculating plane The plane that contains the osculating circle of a curve at a given point. This plane is spanned by the given point and the curve's first and second derivative vectors.

Oslo algorithm The process of simultaneously inserting several knots into a B-spline curve.

Parametrization Assigning parameter values to junction points in spline curves. Also used with a different meaning: the function that describes the speed of a point traversing a curve.

Patch Complicated surfaces are usually broken down into smaller units, called patches. For example, a bicubic spline surface consists of a collection of bicubic patches.

Point A location in space. If one uses coordinate systems to describe space, a point is represented as an n-tuple of real numbers.

Point cloud A (typically large) set of 3D points without any ordering or structure.

Precision A curve or surface generation scheme has nth order precision if it reproduces polynomials of degree n.

Projective map A map comprising affine maps and central projections. Leaves cross ratios of collinear points unchanged. Does (in general) not map parallels to parallels.

Quad Short for quadrilateral.

Quadric A surface with the implicit representation $f(x, y, z) = 0$, where f is a quadratic polynomial. Or: the projective image of an elliptic paraboloid, a hyperbolic paraboloid, or a parabolic cylinder.

Ratio A quantity computed from three collinear points. Invariant under affine maps, but not under projective maps.

Rational curves and surfaces Projections of nonrational (integral) curves or surfaces from four-space into three-space.

Recursive subdivision Curves or surfaces that are defined as the limit of a polygon or polyhedron refinement process.

Ruled surface A surface containing a family of straight lines. Obtained as linear interpolation between two given curves.

S-patch A surface patch with an arbitrary number of boundary curves, constructed by mapping a multidimensional simplex onto a 2D polygon, the domain of the patch.

Segment An individual polynomial (or rational polynomial) curve in an assembly of such curves to form a spline curve. The bivariate analog of a segment is a patch.

Shape parameter A degree of freedom (usually a real number) in a curve or surface representation that can be used to fine-tune the shape of that curve or surface.

Solid modeling The description of closed objects that are bounded by a collection of surfaces.

Space The collection of all points.

Spline curve A continuous curve that comprises several polynomial segments. Spline curves are often represented in terms of B-splines. They may be the result of an interpolation process or of an ab initio design process. If the segments are rational polynomials, we have a rational spline curve.

Standard form The property of a rational curve of having its end weights equal to unity.

Star In a triangular mesh: the set of all triangles having a given point as a vertex.

Stereo lithography The process of producing a physical (usually plastic) model of a part, involving building layers of material hardened by laser rays aimed inside a tank of liquid resin.

Subdivision Breaking a curve or surface down into smaller pieces of the same type as the original curve or surface.

Subdivision surface A surface that is the result of iteratively refining a given control mesh.

Support The region over which a nonnegative function is actually positive.

Surface The locus of all points of a moving and deforming curve. Or: the 3D image of a region in two-space under a continuous map. A surface is often broken down into patches.

Surface spline A piecewise bicubic approximation to a Doo–Sabin surface.

Surface triangulation A collection of triangular facets that covers a smooth surface, obeying the structure of a triangulation.

Tangent The straight line that best approximates a smooth curve at a point on it. This straight line is parallel to the tangent vector.

Tangent vector The first derivative of a differentiable curve at a point on it. The length of the tangent vector depends on the parametrization of the curve.

Tensor product A method to generate rectangular surfaces using curve methods.

Tile The interior of a convex closed 2D polygon.

Torsion A measure of how much a curve "curves away" from the osculating plane at a given point.

Transfinite interpolation Interpolating to curves, with infinitely, that is, transfinitely many points on it, as opposed to discrete interpolation, which interpolates only to finitely many points.

Translational surface A surface that is obtained by translating one curve along another one.

Triangular patch A patch whose domain is a triangle.

Triangulation A collection of triangles, covering a region in $\mathbb{E}^2$, such that the triangles do not overlap, and that any two triangles either have no points in common, or they have one edge in common, or they have one vertex in common. See also surface triangulation.

Trimmed surface If the domain of a parametric surface is divided into "valid" and "invalid" regions, the image of the valid regions is called a trimmed surface.

Twist vector The mixed second partial of a surface at a point. Note: not a geometric property of the surface, but parametrization dependent.

Valency A vertex in a polygonal mesh has valency n if n edges emanate from the vertex. Also: valence.

Variation diminution Intuitively: a curve or surface scheme has this property if its output "wiggles less" than the data from which it is constructed.

Vector A direction. Usually the difference of two points.

Volume deformation A surface or a collection of surfaces may be embedded in a cube. That cube may then be deformed using some trivariate Bézier or B-spline method—this is the volume distortion—in order to change the shape of the initial surface(s).

Voronoi diagram Same as Dirichlet tessellation.

Weight Rational curves and surfaces are often defined in terms of homogeneous coordinates. The last component of the homogeneous coordinate is called weight.

Weight point The point formed by a weighted average of two control points of a rational curve; the weights in the average are the weights of the control points.

B

List of Programs

The following list contains all programs that are contained in the text.

aitken:	Section 7.12
area:	Section 23.4
bessel_ends:	Section 9.9
bez_to_points:	Section 4.5
bspl_to_points:	Section 8.10
curvature_0:	Section 23.4
curvatures:	Section 23.4
deboor:	Section 8.10
deboor_blossom:	Section 8.10
decas:	Section 4.5
degree_elevate:	Section 6.11
direct_gspline:	Section 11.7
hornbez:	Section 5.8
l_u_system:	Section 9.9
netcoons:	Section 15.10
parameters:	Section 9.9
plot_bez_surfaces:	Section 16.10
plot_surf:	Section 15.10
ratbez:	Section 13.10
ratbspline_to_bezier:	Section 13.10
ratbspl_to_bez_surf:	Section 16.10

rat_subdiv:	Section 13.10
set_up_system:	Section 9.9
solve_system:	Section 9.9
spline_surf_int:	Section 16.10
subdiv_rat:	Section 13.2

C

Notation

Here is the notation used in this book.

$\wedge$	cross product
$\dot{}\ \ddot{}$	curve derivatives with respect to the current parameter
$'\ ''$	curve derivatives with respect to arc length
a, b, α, β	real numbers or real-valued functions
$\mathbf{0}$	zero vector, in 3D: short for (0,0,0)
$\mathbf{a}, \mathbf{b}$	points or vectors (possibly in terms of barycentric coordinates)
A, B	matrices
$\mathbf{A}, \mathbf{B}$	matrices whose elements are points (hypermatrices)
$\mathbf{b}_i^r$	intermediate points in the de Casteljau algorithm
B_i^n	univariate Bernstein polynomials of degree n
B_{i}^n	bivariate Bernstein polynomials of degree n
$\mathbf{e1}, \mathbf{e2}, \mathbf{e3}$	short for (1,0,0), (0,1,0), and (0,0,1), respectively
$\mathbf{b}[*, \ldots, *]$	blossom
$\mathbb{E}^d$	d-dimensional euclidean space
$D_{\mathbf{d}}f$	directional derivative of f in the direction $\mathbf{d}$
Δ_i	difference in parameter intervals (i.e., $\Delta_i = u_{i+1} - u_i$)
Δ^r	iterated forward difference operator
H_i^3	cubic Hermite polynomials
I	the identity matrix
$\mathbf{P}$	control polygon
Φ	an affine map

$\mathcal{P}_i$	operators
$t^{<n>}$	n-fold repetition of an argument t
$\|\|\mathbf{v}\|\|$	(euclidean) length of the vector $\mathbf{v}$
$\mathbf{x}_u$	u-partial of $\mathbf{x}(u, v)$

References

[1] S. Abi-Ezzi. *The graphical processing of B-splines in a highly dynamic environment*. PhD thesis, RPI, 1989. Rensselaer Design Research Center.

[2] T. Ackland. On osculatory interpolation, where the given values of the function are at unequal intervals. *J Inst. Actuar.*, 49:369–375, 1915.

[3] H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J ACM*, 17(4):589–602, 1970.

[4] G. Albrecht. A remark on Farin points for Bézier triangles. *Computer Aided Geometric Design*, 12(5):507–512, 1995.

[5] P. Alfeld. A bivariate C^2 Clough-Tocher scheme. *Computer Aided Geometric Design*, 1(3):257–267, 1984.

[6] P. Alfeld. A case study of multivariate piecewise polynomials. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 149–160. SIAM, Philadelphia, 1987.

[7] P. Alfeld and L. Schumaker. The dimension of bivariate spline spaces of smoothness r and degree $d \geq 4r + 1$. *Constructive Approximation*, 3:189–197, 1987.

[8] R. Andersson, E. Andersson, M. Boman, B. Dahlberg, T. Elmroth, and B. Johansson. The automatic generation of convex surfaces. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 427–446. Oxford University Press, 1987.

[9] E. Angel. *Interactive Computer Graphics*. Addison-Wesley, 2000. Second edition.

[10] M. Vigo Anglada, N. Pla Garcia, and P. Brunet Crosa. Directional adaptive surface triangulation. *Computer Aided Geometric Design*, 16(2):107–126, 1999.

[11] G. Aumann. Interpolation with developable Bézier patches. *Computer Aided Geometric Design*, 8(5):409–420, 1991.

[12] G. Baer. Parametrische Interpolation empirischer Raumkurven. *ZAMM*, 57:305–314, 1977.

[13] A. Ball. Consurf I: introduction of the conic lofting tile. *Computer Aided Design*, 6(4):243–249, 1974.

[14] A. Ball. Consurf II: description of the algorithms. *Computer Aided Design*, 7(4):237–242, 1975.

[15] A. Ball. Consurf III: how the program is used. *Computer Aided Design*, 9(1):9–12, 1977.

[16] A. Ball. Reparametrization and its application in computer-aided geometric design. *Int. J for Numer. Methods in Eng.*, 20:197–216, 1984.

[17] A. Ball. The parametric representation of curves and surfaces using rational polynomial functions. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 39–62. Oxford University Press, 1987.

[18] A. Ball and D. Storry. A matrix approach to the analysis of recursively generated B-spline surfaces. *Computer Aided Design*, 18(8):437–442, 1986.

[19] A. Ball and D. Storry. Conditions for tangent plane continuity of recursively generated B-spline surfaces. *ACM Transactions on Graphics*, 7(2):83–102, 1988.

[20] R. Barnhill. Smooth interpolation over triangles. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 45–70. Academic Press, 1974.

[21] R. Barnhill. Representation and approximation of surfaces. In J. R. Rice, editor, *Mathematical Software III*, pages 69–120. Academic Press, 1977.

[22] R. Barnhill. Coons' patches. *Computers in Industry*, 3:37–43, 1982.

[23] R. Barnhill. A survey of the representation and design of surfaces. *IEEE Computer Graphics and Applications*, 3:9–16, 1983.

[24] R. Barnhill. Surfaces in computer aided geometric design: a survey with new results. *Computer Aided Geometric Design*, 2(1–3):1–17, 1985.

[25] R. Barnhill, editor. *Geometry Processing*. SIAM, Philadelphia, 1992.

[26] R. Barnhill, G. Birkhoff, and W. Gordon. Smooth interpolation in triangles. *J Approx. Theory*, 8(2):114–128, 1973.

[27] R. Barnhill, W. Boehm, and J. Hoschek, editors. *Surfaces in CAGD '89*. North Holland, Amsterdam, 1990.

[28] R. Barnhill, J. Brown, and I. Klucewicz. A new twist in CAGD. *Computer Graphics and Image Processing*, 8(1):78–91, 1978.

[29] R. Barnhill and G. Farin. C^1 quintic interpolation over triangles: two explicit representations. *Int. J Numer. Methods in Eng.*, 17:1763–1778, 1981.

[30] R. Barnhill, G. Farin, L. Fayard, and H. Hagen. Twists, curvatures, and surface interrogation. *Computer Aided Design*, 20(6):341–346, 1988.

[31] R. Barnhill, G. Farin, M. Jordan, and B. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(1–2):3–16, 1987.

[32] R. Barnhill and J. Gregory. Compatible smooth interpolation in triangles. *J of Approx. Theory*, 15(3):214–225, 1975.

[33] R. Barnhill and J. Gregory. Polynomial interpolation to boundary data on triangles. *Math. of Computation*, 29(131):726–735, 1975.

[34] R. Barnhill and R. F. Riesenfeld, editors. *Computer Aided Geometric Design*. Academic Press, 1974.

[35] M. Barnsley. *Fractals everywhere*. Academic Press, 1988.

[36] P. Barron. The general formula of the quadratically precise nine-parameter interpolant. Technical report, University of Utah, 1988. Manuscript.

[37] P. Barry. de Boor-Fix functionals and polar forms. *Computer Aided Geometric Design*, 7(5):425–430, 1990.

[38] P. Barry and R. Goldman. De Casteljau-type subdivision is peculiar to Bèzier curves. *Computer Aided Design*, 20(3):114–116, 1988.

[39] P. Barry and R. Goldman. A recursive proof of a B-spline identity for degree elevation. *Computer Aided Geometric Design*, 5(2):173–175, 1988.

[40] B. Barsky. *The Beta-spline: a local representation based on shape parameters and fundamental geometric measures*. PhD thesis, Dept. of Computer Science, U. of Utah, 1981.

[41] B. Barsky. Exponential and polynomial methods for applying tension to an interpolating spline curve. *Computer Vision, Graphics and Image Processing*, 27:1–18, 1984.

[42] B. Barsky and J. Beatty. Varying the Betas in Beta-splines. Technical Report CS-82-49, U. of Waterloo, Waterloo, Ontario, Canada N31 3G1, December 1982.

[43] B. Barsky and T. DeRose. Geometric continuity of parametric curves. Technical Report UCB/CSB 84/205, Computer Science Division, U. of California, Berkley, 1984.

[44] B. Barsky and T. DeRose. The beta2-spline: a special case of the beta-spline curve and surface representation. *IEEE Computer Graphics and Applications*, 5(9):46–58, 1985.

[45] B. Barsky and D. Greenberg. Determining a set of B-spline control vertices to generate an interpolating surface. *Computer Graphics and Image Processing*, 14(3):203–222, 1980.

[46] R. Bartels and J. Beatty. Beta-splines with a difference. Technical Report CS-83-40, Computer Science Department, U. of Waterloo, Ontario, Canada, 1984.

[47] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[48] R. Bartels and D. Forsey. Modifying B-spline curves. *IEEE Computer Graphics and Applications*, 12(3):179–208, 1995.

[49] J. Beck, R. Farouki, and J. Hinds. Surface analysis methods. *IEEE Computer Graphics and Applications*, 6(12):18–36, 1986.

[50] E. Beeker. Smoothing of shapes designed with free-form surfaces. *Computer Aided Design*, 18(4):224–232, 1986.

[51] G. Behforooz and N. Papamichael. End conditions for interpolatory cubic splines with unequally spaced knots. *J of Comp. Applied Math.*, 6(1), 1980.

[52] M. Berger. *Geometry I*. Springer-Verlag, 1987.

[53] S. Bernstein. Démonstration du théoreme de Weierstrass fondeé sur le calcul des probabilités. *Harkov Soobs. Matem ob-va*, 13:1–2, 1912.

[54] H. Bez. On invariant curve forms. *Computer Aided Geometric Design*, 3(3):193–204, 1986.

[55] H. Bez and J. Edwards. Distributed algorithm for the planar convex hull algorithm. *Computer Aided Design*, 22(2):81–86, 1990.

[56] P. Bézier. Définition numérique des courbes et surfâces I. *Automatisme*, XI:625–632, 1966.

[57] P. Bézier. Définition numérique des courbes et surfâces II. *Automatisme*, XII:17–21, 1967.

[58] P. Bézier. Procédé de définition numérique des courbes et surfâces non mathématiques. *Automatisme*, XIII(5):189–196, 1968.

[59] P. Bézier. *Numerical Control: Mathematics and Applications*. Wiley, 1972. translated from the French by A. R. Forrest.

[60] P. Bézier. Mathematical and practical possibilities of UNISURF. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 127–152. Academic Press, 1974.

[61] P. Bézier. *Essay de définition numérique des courbes et des surfaces expérimentales*. PhD thesis, University of Paris VI, 1977.

[62] P. Bézier. General distortion of an ensemble of biparametric patches. *Computer Aided Design*, 10(2):116–120, 1978.

[63] P. Bézier. *The Mathematical Basis of the UNISURF CAD System*. Butterworths, London, 1986.

[64] G. Birkhoff. *Aesthetic Measure*. Harvard University Press, 1933.

[65] W. Blaschke. *Differentialgeometrie*. Chelsea, 1953. Reprint of the original 1923 edition.

[66] W. Boehm. Parameterdarstellung kubischer und bikubischer Splines. *Computing*, 17:87–92, 1976.

[67] W. Boehm. Cubic B-spline curves and surfaces in computer aided geometric design. *Computing*, 19(1):29–34, 1977.

[68] W. Boehm. Inserting new knots into B-spline curves. *Computer Aided Design*, 12(4):199–201, 1980.

[69] W. Boehm. Generating the Bézier points of B-spline curves and surfaces. *Computer Aided Design*, 13(6):365–366, 1981.

[70] W. Boehm. On cubics: a survey. *Computer Graphics and Image Processing*, 19:201–226, 1982.

[71] W. Boehm. Curvature continuous curves and surfaces. *Computer Aided Geometric Design*, 2(4):313–323, 1985.

[72] W. Boehm. Multivariate spline methods in CAGD. *Computer Aided Design*, 18(2):102–104, 1986.

[73] W. Boehm. Rational geometric splines. *Computer Aided Geometric Design*, 4(1–2):67–77, 1987.

[74] W. Boehm. Smooth curves and surfaces. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 175–184. SIAM, Philadelphia, 1987.

[75] W. Boehm. On de Boor-like algorithms and blossoming. *Computer Aided Geometric Design*, 5(1):71–80, 1988.

[76] W. Boehm. On the definition of geometric continuity. *Computer Aided Design*, 20(7):370–372, 1988. Letter to the editor.

[77] W. Boehm. Visual continuity. *Computer Aided Design*, 20(6):307–311, 1988.

[78] W. Boehm. On cyclides in geometric modeling. *Computer Aided Geometric Design*, 7(1-4):243–256, 1990.

[79] W. Boehm. Smooth rational curves. *Computer Aided Design*, 22(1):70, 1990. Letter to the editor.

[80] W. Boehm. Circles of curvature for curves in space. *Computer Aided Geometric Design*, 16(7):633–638, 2000.

[81] W. Boehm and G. Farin. Letter to the editor. *Computer Aided Design*, 15(5):260–261, 1983. Concerning subdivison of Bézier triangles.

[82] W. Boehm, G. Farin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1(1):1–60, 1984.

[83] W. Boehm and D. Hansford. Bézier patches on quadrics. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 1–14. SIAM, 1991.

[84] W. Boehm and A. Müller. On de Casteljau's algorithm. *Computer Aided Geometric Design*, 16(7):583–586, 2000.

[85] W. Boehm and H. Prautzsch. *Geometric Foundations of Geometric Design*. AK Peters, Boston, 1992.

[86] W. Boehm and H. Prautzsch. *Numerical Methods*. Vieweg, 1992.

[87] W. Boehm and H. Prautzsch. *B-Methods*. Springer-Verlag, Heidelberg, 2002.

[88] G. Bol. *Projective Differential Geometry, Vol. 1.* Vandenhoeck and Ruprecht, Göttingen, 1950. Vol. 2 in 1954, Vol. 3 in 1967. In German.

[89] G. Bonneau. Weight estimation of rational Bézier curves and surfaces. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modeling*, pages 79–86. Springer, Vienna, 1995.

[90] F. Bookstein. Fitting conic sections to scattered data. *CGIP*, 9:56–71, 1979.

[91] J. Brewer and D. Anderson. Visual interaction with Overhauser curves and surfaces. *Computer Graphics*, 11(2):132–137, 1977.

[92] J. Brown. Vertex based data dependent triangulations. *Computer Aided Geometric Design*, 8(3):239–251, 1991.

[93] I. Brueckner. Construction of Bézier points of quadrilaterals from those of triangles. *Computer Aided Design*, 12(1):21–24, 1980.

[94] P. Brunet. Increasing the smoothness of bicubic spline surfaces. *Computer Aided Geometric Design*, 2(1–3):157–164, 1985.

[95] G. Brunnett. Geometric design with trimmed surfaces. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modeling*, pages 101–116. Springer, Vienna, 1995.

[96] G. Brunnett, H. Bieri, and G. Farin, editors. *Geometric Modeling '99*. Springer, 2001.

[97] G. Brunnett, T. Schreiber, and J. Braun. The geometry of optimal degree reduction of Bézier curves. *Computer Aided Geometric Design*, 13(8):773–788, 1996.

[98] B. Buchberger. Applications of Groebner bases in non-linear computational geometry. In J. Rice, editor, *Mathematical Aspects of Scientific Software*. Springer-Verlag, 1988.

[99] H. Burchardt, J. Ayers, W. Frey, and N. Sapidis. Approximation with aesthetic constraints. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 3–28. SIAM, Philadelphia, 1994.

[100] C. Calladine. Gaussian curvature and shell structures. In J. Gregory, editor, *The Mathematics of Surfaces*, pages 179–196. Clarendon Press, 1986.

[101] Y. Cao and X. Hua. The convexity of quadratic parametric triangular Bernstein-Bézier surfaces. *Computer Aided Geometric Design*, 8(1):1–6, 1991.

[102] M. Casale and J. Bobrow. A set operation algorithm for sculptured solids modeled with trimmed patches. *Computer Aided Geometric Design*, 6(3):235–248, 1989.

[103] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[104] E. Catmull and R. Rom. A class of local interpolating splines. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974.

[105] G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[106] G. Chang. Matrix formulation of Bézier technique. *Computer Aided Design*, 14(6):345–350, 1982.

[107] G. Chang and P. Davis. The convexity of Bernstein polynomials over triangles. *J Approx Theory*, 40:11–28, 1984.

[108] G. Chang and Y. Feng. An improved condition for the convexity of Bernstein-Bézier surfaces over triangles. *Computer Aided Geometric Design*, 1(3):279–283, 1985.

[109] S. Chang, M. Shantz, and R. Rochetti. Rendering cubic curves and surfaces with integer adaptive forward differencing. *Computer Graphics*, 23(3):157–166, 1989. SIGGRAPH '89 Proceedings.

[110] P. Charrot and J. Gregory. A pentagonal surface patch for computer aided geometric design. *Computer Aided Geometric Design*, 1(1):87–94, 1984.

[111] F. Cheng and B. Barsky. Interproximation: interpolation and approximation using cubic spline curves. *Computer Aided Design*, 23(10):700–706, 1991.

[112] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17(3):289–298, 1983.

[113] H. Chiyokura, T. Takamura, K. Konno, and T. Harada. G^1 surface interpolation over irregular meshes with rational curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 15–34. SIAM, 1991.

[114] J. Chou. Higher order Bézier circles. *Computer Aided Design*, 27(4):303–309, 1995.

[115] C. Chui. *Multivariate Splines*. SIAM, Philadelphia, 1988.

[116] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[117] R. Clough and J. Tocher. Finite element stiffness matrices for analysis of plates in blending. In *Proceedings of Conference on Matrix Methods in Structural Analysis*, 1965.

[118] J. Cobb. A rational bicubic representation of the sphere. Technical report, Computer science, U. of Utah, 1988.

[119] J. Cobb. Letter to the editor. *Computer Aided Geometric Design*, 6(1):85, 1989. Concerning Piegl's sphere approximation.

[120] E. Cohen. A new local basis for designing with tensioned splines. *ACM Transactions on Graphics*, 6(2):81–122, 1987.

[121] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. *Comp. Graphics and Image Process.*, 14(2):87–111, 1980.

[122] E. Cohen and C. O'Dell. A data dependent parametrization for spline approximation. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 155–166. Academic Press, 1989.

[123] E. Cohen and L. Schumaker. Rates of convergence of control polygons. *Computer Aided Geometric Design*, 2(1–3):229–235, 1985.

[124] S. Coons. Surfaces for computer aided design. Technical report, MIT, 1964. Available as AD 663 504 from the National Technical Information service, Springfield, VA 22161.

[125] S. Coons. Surfaces for computer aided design of space forms. Technical report, MIT, 1967. Project MAC-TR 41.

[126] S. Coons. Rational bicubic surface patches. Technical report, MIT, 1968. Project MAC.

[127] S. Coons. Surface patches and B-spline curves. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 1–16. Academic Press, 1974.

[128] S. Coons. *Méthode Matricielle*. Hermes, Paris, 1987. Translation from English by P. Bézier and M. Moronval.

[129] M. Cox. The numerical evaluation of B-splines. *J Inst. Maths. Applics.*, 10:134–149, 1972.

[130] H. Coxeter. *Introduction to Geometry*. Wiley, 1961.

[131] W. Dahmen. Subdivision algorithms converge quadratically. *J. of Computational and Applied Mathematics*, 16:145–158, 1986.

[132] M. Daniel and J. Daubisse. The numerical problem of using Bézier curves and surfaces in the power basis. *Computer Aided Geometric Design*, 6(2):121–128, 1989.

[133] P. Davis. *Interpolation and Approximation*. Dover, New York, 1975. first edition 1963.

[134] P. Davis. *Circulant Matrices*. Wiley, New York, 1979.

[135] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000. Second edition.

[136] C. de Boor. Bicubic spline interpolation. *J. Math. Phys.*, 41:212–218, 1962.

[137] C. de Boor. On calculating with B-splines. *J Approx. Theory*, 6(1):50–62, 1972.

[138] C. de Boor. *A Practical Guide to Splines*. Springer, 1978.

[139] C. de Boor. B-form basics. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 131–148. SIAM, Philadelphia, 1987.

[140] C. de Boor. Cutting corners always works. *Computer Aided Geometric Design*, 4(1–2):125–131, 1987.

[141] C. de Boor. Local corner cutting and the smoothness of the limiting curve. *Computer Aided Geometric Design*, 7(5):389–398, 1990.

[142] C. de Boor and R. de Vore. A geometric proof of total positivity for spline interpolation. *Math. of Computation*, 45(172):497–504, 1985.

[143] C. de Boor and K. Höllig. B-splines without divided differences. In G. Farin, editor, *Geometric Modeling - Algorithms and New Trends*, pages 21–27. SIAM, Philadelphia, 1987.

[144] C. de Boor, K. Höllig, and M. Sabin. High accuracy geometric Hermite interpolation. *Computer Aided Geometric Design*, 4(4):269–278, 1987.

[145] P. de Casteljau. Outillages méthodes calcul. Technical report, A. Citroën, Paris, 1959.

[146] P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.

[147] P. de Casteljau. *Shape Mathematics and CAD*. Kogan Page, London, 1986.

[148] P. de Casteljau. *Le Lissage*. Hermes, Paris, 1990.

[149] P. de Casteljau. de Casteljau's autobiography: My time at Citroën. *Computer Aided Geometric Design*, 16(7):583–586, 2000.

[150] G. de Rham. Un peu de mathématique à propos d'une courbe plane. *Elem. Math.*, 2:73–76; 89–97, 1947. Also in Collected Works, 678–689.

[151] G. de Rham. Sur une courbe plane. *J Math. Pures Appl.*, 35:25–42, 1956. Also in Collected Works, 696–713.

[152] W. Degen. Some remarks on Bézier curves. *Computer Aided Geometric Design*, 5(3):259–268, 1988.

[153] W. Degen. Explicit continuity conditions for adjacent Bézier surface patches. *Computer Aided Geometric Design*, 7(1–4):181–190, 1990.

[154] W. Degen. The shape of the Overhauser spline. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modeling*, pages 117–128. Springer, Vienna, 1995.

[155] W. Degen and V. Milbrandt. The geometric meaning of Nielson's affine invariant norm. *Computer Aided Geometric Design*, 15(1):19–25, 1997.

[156] S. Demko. Interpolation by quadratic splines. *J. Approx. Theory*, 23:392–400, 1978.

[157] Y. DeMontaudouin. Resolution of $p(x, y) = 0$. *Computer Aided Design*, 23(9):653–654, 1991.

[158] T. DeRose. *Geometric continuity: a parametrization independent measure of continuity for computer aided geometric design*. PhD thesis, Dept. of Computer Science, U. Calif. at Berkeley, 1985. Also tech report UCB/CSD 86/255.

[159] T. DeRose. Composing Bézier simplices. *ACM Transactions on Graphics*, 7(3):198–221, 1988.

[160] T. DeRose. Geometric programming. In *SIGGRAPH '88 course notes*, 1988.

[161] T. DeRose. A coordinate-free approach to geometric programming. In W. Strasser and H.-P. Seidel, editors, *Theory and Practice of Geometric Modeling*, pages 291–306. Springer-Verlag, Berlin, 1989.

[162] T. DeRose. Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces. *Computer Aided Geometric Design*, 7(1–4):165–180, 1990.

[163] T. DeRose. Rational Bezier curves and surfaces on projective domains. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 35–46. SIAM, 1991.

[164] T. DeRose and B. Barsky. Geometric continuity, shape parameters, and geometric constructions for Catmull-Rom splines. *ACM Transactions on Graphics*, 7(1):1–41, 1988.

[165] T. DeRose and R. Goldman. A tutorial introduction to blossoming. In H. Hagen and D. Roller, editors, *Geometric Modeling*. Springer, 1991.

[166] T. DeRose and T. Holman. The triangle: a multiprocessor architecture for fast curve and surface generation. Technical Report 87-08-07, Computer Science Department, Univ. of Washington, 1987.

[167] T. DeRose and C. Loop. S-patches: a class of representations for multisided surface patches. Technical Report 88-05-02, Computer Science Department, Univ. of Washington, 1988.

[168] T. DeRose and C. Loop. The S-patch: a new multisided patch scheme. *ACM Transactions on Graphics*, 8(3):204–234, 1989.

[169] J. Dill. An application of color graphics to the display of surface curvature. *Computer Graphics*, 15:153–161, 1981.

[170] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Englewood Cliffs, 1976.

[171] T. Dokken. Finding intersections of B-spline represented geometries using recursive subdivision techniques. *Computer Aided Geometric Design*, 2(1–3):189–195, 1985.

[172] T. Dokken, M. Daehlen, T. Lyche, and K. Morken. Good approximation of circles by curvature-continuous Bézier curves. *Computer Aided Geometric Design*, 7(1–4):33–42, 1990.

[173] T. Dokken and A. Ytrehus. Recursive subdivision and iteration in intersections and related problems. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 207–214. Academic Press, 1989.

[174] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[175] W-H. Du and F. Schmitt. On the G^1 continuity of piecewise Bézier surfaces: a review with new results. *Computer Aided Design*, 22(9):556–573, 1990.

[176] N. Dyn, J. Gregory, and D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990.

[177] N. Dyn, J. Gregory, and D. Levin. Piecewise uniform subdivision schemes. In M. Daehlen, T. Lyche, and L. Schumaker, editors, *Mathematical Methods for Curve and Surface Design*, pages 111–120. Vanderbilt University Press, 1995.

[178] N. Dyn, D. Levin, and J. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4(4):257–268, 1987.

[179] N. Dyn, D. Levin, and C. Micchelli. Using parameters to increase smoothness of curves and surfaces generated by subdivision. *Computer Aided Geometric Design*, 7(1–4):129–140, 1990.

[180] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA J Numer. Analysis*, 10:137–154, 1990.

[181] N. Dyn and C. Micchelli. Piecewise polynomial spaces and geometric continuity of curves. Technical report, IBM report RCC11390, Yorktown Heights, 1985.

[182] M. Eck. Degree reduction of Bézier curves. *Computer Aided Geometric Design*, 10(3–4):237–252, 1993.

[183] M. Eck. Multiresolution analysis of arbitrary meshes. *Computer Graphics*, 29:173–182, 1995. Proceedings SIGGRAPH 1995.

[184] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95 Proc.*, pages 173–182. ACM, August 1995.

[185] M. Eck and J. Hadenfeld. A stepwise algorithm for converting B-splines. In P.-J. Laurent, A. Le Méhauté, and L. Schumaker, editors, *Curves and Surfaces in Geometric Design*, pages 131–138. AK Peters, Ltd., 1994.

[186] M. Epstein. On the influence of parametrization in parametric interpolation. *SIAM J Numer. Analysis*, 13(2):261–268, 1976.

[187] G. Farin. *Konstruktion und Eigenschaften von Bézier-Kurven und-Flächen.* Master's thesis, Technical University Braunschweig, Germany, 1977.

[188] G. Farin. *Subsplines ueber Dreiecken.* PhD thesis, Technical University Braunschweig, Germany, 1979.

[189] G. Farin. Bézier polynomials over triangles and the construction of piecewise C^r polynomials. Technical Report TR/91, Brunel University, Uxbridge, England, 1980.

[190] G. Farin. A construction for the visual C^1 continuity of polynomial surface patches. *Computer Graphics and Image Processing*, 20:272–282, 1982.

[191] G. Farin. Designing C^1 surfaces consisting of triangular cubic patches. *Computer Aided Design*, 14(5):253–256, 1982.

[192] G. Farin. Visually C^2 cubic splines. *Computer Aided Design*, 14(3):137–139, 1982.

[193] G. Farin. Algorithms for rational Bézier curves. *Computer Aided Design*, 15(2):73–77, 1983.

[194] G. Farin. A modified Clough-Tocher interpolant. *Computer Aided Geometric Design*, 2(1–3):19–27, 1985.

[195] G. Farin. Some remarks on V^2-splines. *Computer Aided Geometric Design*, 2(2):325–328, 1985.

[196] G. Farin. Piecewise triangular C^1 surface strips. *Computer Aided Design*, 18(1):45–47, 1986.

[197] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.

[198] G. Farin. Curvature continuity and offsets for piecewise conics. *ACM Transactions on Graphics*, 8(2):89–99, 1989.

[199] G. Farin, editor. *NURBS for Curve and Surface Design.* SIAM, Philadelphia, 1991.

[200] G. Farin. Degree reduction fairing of cubic B-spline curves. In R. Barnhill, editor, *Geometry Processing for Design and Manufacturing*, pages 87–99. SIAM, Philadelphia, 1992.

[201] G. Farin. Tighter convex hulls for rational Bézier curves. *Computer Aided Geometric Design*, 10(2):123–126, 1993.

[202] G. Farin. *NURB Curves and Surfaces.* AK Peters, Boston, 1995. Second edition 1999.

[203] G. Farin. Shape. In B. Engquist and W. Schmid, editors, *Mathematics Unlimited—2001 and Beyond*, pages 463–467. Springer, 2001.

[204] G. Farin. A history of curves and surfaces in CAGD. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of 3D Modeling and Graphics*, pages 1–22. Elsevier, 2002. In preparation.

[205] G. Farin and P. Barry. A link between Lagrange and Bézier curve and surface schemes. *Computer Aided Design*, 18:525–528, 1986.

[206] G. Farin and H. Hagen. Optimal twist estimation. In H. Hagen, editor, *Surface Design*. SIAM, Philadelphia, 1992.

[207] G. Farin and D. Hansford. Discrete Coons patches. *Computer Aided Geometric Design*, 16(7):691–700, 1999.

[208] G. Farin and D. Hansford. *The Essentials of CAGD*. AK Peters, 2000.

[209] G. Farin, D. Hansford, and A. Worsey. The singular cases for γ-spline interpolation. *Computer Aided Geometric Design*, 7(6):533–546, 1990.

[210] G. Farin, J. Hoschek, and M.-S. Kim, editors. *Handbook of 3D Modeling and Graphics*. Elsevier, 2002. In preparation.

[211] G. Farin and D. Jung. Linear precision of rational Bézier curves. *Computer Aided Geometric Design*, 12(4):431–433, 1995.

[212] G. Farin and P. Kashyap. An iterative Clough-Tocher interpolant. *Mathematical Modelling and Numerical Analysis*, 26(1):201–209, 1992.

[213] G. Farin, B. Piper, and A. Worsey. The octant of a sphere as a non-degenerate triangular Bézier patch. *Computer Aided Geometric Design*, 4(4):329–332, 1988.

[214] G. Farin, G. Rein, N. Sapidis, and A. Worsey. Fairing cubic B-spline curves. *Computer Aided Geometric Design*, 4(1–2):91–104, 1987.

[215] G. Farin and N. Sapidis. Curvature and the fairness of curves and surfaces. *IEEE Computer Graphics and Applications*, 9(2):52–57, 1989.

[216] G. Farin and A. Worsey. Reparametrization and degree elevation of rational Bézier curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 47–58. SIAM, 1991.

[217] R. Farouki. Exact offset procedures for simple solids. *Computer Aided Geometric Design*, 2(4):257–279, 1985.

[218] R. Farouki. The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design*, 3(1):15–43, 1986.

[219] R. Farouki. Direct surface section evaluation. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 319–334. SIAM, Philadelphia, 1987.

[220] R. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, 1991.

[221] R. Farouki. Pythagorean-hodograph curves in practical use. In R. E. Barnhill, editor, *Geometry Processing for Design and Manufacturing*, pages 3–33. SIAM, Philadelphia, 1992.

[222] R. Farouki and T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation*, 65:1553–1566, 1996.

[223] R. Farouki and J. Hinds. A hierarchy of geometric forms. *IEEE Computer Graphics and Applications*, 5(5):51–78, 1985.

[224] R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[225] R. Farouki and V. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, 1988.

[226] R. Farouki and T. Sakkalis. Real rational curves are not 'unit speed.' *Computer Aided Geometric Design*, 8(2):151–158, 1991.

[227] R. T. Farouki and H. Pottmann. Polynomial and rational pythagorian-hodograph curves reconciled. In Glen Mullineux, editor, *The Mathematics of Surfaces VI*, pages 355–378. Oxford University Press, 1996.

[228] I. Faux and M. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, 1979.

[229] L. Fayard. Surface interrogation using curvature plots. Master's thesis, Dept. of Computer Science, Arizona State Univ., 1988.

[230] D. Ferguson. Construction of curves and surfaces using numerical optimization techniques. *Computer Aided Design*, 18(1):15–21, 1986.

[231] J. Ferguson. Multivariable curve interpolation. *J ACM*, 11(2):221–228, 1964.

[232] D. Filip. Adaptive subdivision algorithms for a set of Bézier triangles. *Computer Aided Design*, 18(2):74–78, 1986.

[233] J. Fiorot and P. Jeannin. *Rational Curves and Surfaces*. Wiley, Chicester, 1992. Translated from the French by M. Harrison.

[234] J. Fiorot and P. Jeannin. Linear precision of BR-curves. *Computer Aided Geometric Design*, 12(4):435–438, 1995.

[235] M. Floater. Derivatives of rational Bézier curves. *Computer Aided Geometric Design*, 10, 1993.

[236] M. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–271, 1997.

[237] N. Fog. Creative definition and fairing of ship hulls using a B-spline surface. *Computer Aided Design*, 16(4):225–230, 1984.

[238] J. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[239] T. Foley. Interpolation with interval and point tension controls using cubic weighted ν-splines. *ACM Trans. on Math. Software*, 13(1):68–96, 1987.

[240] A. R. Forrest. *Curves and surfaces for computer-aided design*. PhD thesis, Cambridge, 1968.

[241] A. R. Forrest. On Coons' and other methods for the representation of curved surfaces. *Computer Graphics and Image Processing*, 1(4):341–359, 1972.

[242] A. R. Forrest. On the rendering of surfaces. *Computer Graphics*, 13(2): 253–259, 1979.

[243] A. R. Forrest. The twisted cubic curve: a computer-aided geometric design approach. *Computer Aided Design*, 12(4):165–172, 1980.

[244] A. R. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer J*, 15(1):71–79, 1972. reprinted in CAD 22(9):527–537, 1990.

[245] D. Forsey and R. Bartels. Hierarchical B-spline refinement. *Computer Graphics*, 22(4):205–212, 1988. SIGGRAPH Proceedings.

[246] R. Franke. Scattered data interpolation: tests of some methods. *Math. Computation*, 38(157):181–200, 1982.

[247] R. Franke. Recent advances in the approximation of surfaces from scattered data. *Topics in Multivariate Approx.*, pages 79–98, 1987.

[248] R. Franke and L. Schumaker. A bibliography of multivariate approximation. In C. Chui and L. Schumaker, editors, *Topics in Multivariate Approximation*. Academic Press, 1986.

[249] L. Frederickson. Triangular spline interpolation/generalized triangular splines. Technical Report no. 6/70 and 7/71, Dept. of Math., Lakehead University, Canada, 1971.

[250] F. Fritsch. Energy comparison of Wilson-Fowler splines with other interpolating splines. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 185–201. SIAM, Philadelphia, 1987.

[251] Q. Fu. The intersection of a bicubic patch and a plane. *Computer Aided Geometric Design*, 7(6):475–488, 1990.

[252] J. Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. Morgan Kaufmann, 1998.

[253] D. Gans. *Transformations and Geometries*. Appleton-Century-Crofts, 1969.

[254] M. Garland and P. Heckbert. Simplifying surfaces with color and texture using quadric error bounds. In *Proceedings of IEEE Visualization '98*. ACM Press, 1998.

[255] T. Garrity and J. Warren. Geometric continuity. *Computer Aided Geometric Design*, 8(1):51–66, 1991.

[256] G. Geise. Über berührende Kegelschnitte ebener Kurven. *ZAMM*, 42:297–304, 1962.

[257] G. Geise and U. Langbecker. Finite quadratic segments with four conic boundary curves. *Computer Aided Geometric Design*, 7(1–4):141–150, 1990.

[258] A. Geisow. *Surface interrogations*. PhD thesis, Univ. of East Anglia, 1983.

[259] A. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989.

[260] M. Goldapp. Approximation of circular arcs by cubic polynomials. *Computer Aided Geometric Design*, 8(3):227–238, 1991.

[261] R. Goldman. Using degenerate Bézier triangles and tetrahedra to subdivide Bézier curves. *Computer Aided Design*, 14(6):307–311, 1982.

[262] R. Goldman. Illicit expressions in vector algebra. *ACM Transactions on Graphics*, 4(3):223–243, 1985.

[263] R. Goldman. The method of resolvents: a technique for the implicitization, inversion, and intersection of non-planar, parametric, rational cubic curves. *Computer Aided Geometric Design*, 2(4):237–255, 1985.

[264] R. Goldman. Blossoming and knot insertion algorithms for B-spline curves. *Computer Aided Geometric Design*, 7(1–4):69–82, 1990.

[265] R. Goldman. Blossoming with cancellation. *Computer Aided Geometric Design*, 16(7):671–689, 2000.

[266] R. Goldman and T. DeRose. Recursive subdivision without the convex hull property. *Computer Aided Geometric Design*, 3(4):247–265, 1986.

[267] R. Goldman and C. Micchelli. Algebraic aspects of geometric continuity. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 313–332. Academic Press, 1989.

[268] J. Gomes, L. Darsa, B. Costa, and L. Velho, editors. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.

[269] H. Gonska and J. Meier. A bibliography on approximation of functions by Bernstein type operators. In L. Schumaker and K. Chui, editors, *Approximation Theory IV*. Academic Press, 1983.

[270] T. Goodman. Properties of Beta-splines. *J Approx. Theory*, 44(2):132–153, 1985.

[271] T. Goodman. Shape preserving interpolation by parametric rational cubic splines. Technical report, University of Dundee, 1988. Department of Mathematics and Computer Science.

[272] T. Goodman. Constructing piecewise rational curves with Frenet frame continuity. *Computer Aided Geometric Design*, 7(1–4):15–32, 1990.

[273] T. Goodman. Closed surfaces defined from biquadratic splines. *Constructive Approximation*, 7(2):149–160, 1991.

[274] T. Goodman. Convexity of Bézier nets on triangulations. *Computer Aided Geometric Design*, 8(2):175–180, 1991.

[275] T. Goodman. Inflections on curves in two and three dimensions. *Computer Aided Geometric Design*, 8(1):37–51, 1991.

[276] T. Goodman, B. Ong, and K. Unsworth. Constrained interpolation using rational cubic splines. In G. Farin, editor, *NURBS for Curve and Surface Design*. SIAM, 1991.

[277] T. Goodman and H. Said. Properties of generalized Ball curves and surfaces. *Computer Aided Design*, 23(8):554–560, 1991.

[278] T. Goodman and H. Said. Shape preserving properties of the generalised Ball basis. *Computer Aided Geometric Design*, 8(2):115–122, 1991.

[279] T. Goodman and K. Unsworth. Manipulating shape and producing geometric continuity in beta-spline surfaces. *IEEE Computer Graphics and Applications*, 6(2):50–56, 1986.

[280] W. Gordon. Blending-function methods of bivariate and multivariate interpolation and approximation. *SIAM J Numer. Analysis*, 8(1):158–177, 1969.

[281] W. Gordon. Distributive lattices and the approximation of multivariate functions. In I. Schoenberg, editor, *Approximation with Special Emphasis on Splines*. University of Wisconsin Press, Madison, 1969.

[282] W. Gordon. Free-form surface interpolation through curve networks. Technical Report GMR-921, General Motors Research Laboratories, 1969.

[283] W. Gordon. Spline-blended surface interpolation through curve networks. *J of Math. and Mechanics*, 18(10):931–952, 1969.

[284] W. Gordon and R. Riesenfeld. B-spline curves and surfaces. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 95–126. Academic Press, 1974.

[285] T. Gossing. Bulge, shear and squash: a representation for the general conic arc. *Computer Aided Design*, 13(2):81–84, 1981.

[286] J. Gourret, N. Magnenat-Thalmann, and D. Thalmann. Modeling of contact deformations between a synthetic human and its environment. *Computer Aided Design*, 23(7):514–520, 1991.

[287] T. Grandine. Computing zeros of spline functions. *Computer Aided Geometric Design*, 6(2):129–136, 1989.

[288] J. Gregory. Smooth interpolation without twist constraints. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 71–88. Academic Press, 1974.

[289] J. Gregory. C^1 rectangular and non-rectangular surface patches. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 25–34. North-Holland, 1983.

[290] J. Gregory. N-sided surface patches. In J. Gregory, editor, *The Mathematics of Surfaces*, pages 217–232. Clarendon Press, 1986.

[291] J. Gregory. Geometric continuity. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 353–372. Academic Press, 1989.

[292] J. Gregory and P. Charrot. A C^1 triangular interpolation patch for computer-aided geometric design. *Computer Graphics and Image Processing*, 13(1):80–87, 1980.

[293] J. Gregory and J. Hahn. Geometric continuity and convex combination patches. *Computer Aided Geometric Design*, 4(1–2):79–90, 1987.

[294] J. Gregory and M. Safraz. A rational cubic spline with tension. *Computer Aided Geometric Design*, 7(1–4):1–14, 1990.

[295] J. Gregory and J. Zhou. Convexity of Bézier on sub-triangles. *Computer Aided Geometric Design*, 8(3):207–213, 1991.

[296] E. Grosse. Tensor spline approximation. *Linear Algebra and Its Applications*, 34:29–41, 1980.

[297] H. Gursoy. *Shape interrogation by medial axis transform for automated analysis*. PhD thesis, MIT, Dept. of Ocean Engineering, 1989.

[298] H. Hagen. Geometric spline curves. *Computer Aided Geometric Design*, 2(1–3):223–228, 1985.

[299] H. Hagen. Bézier-curves with curvature and torsion continuity. *Rocky Mtn. J of Math.*, 16(3):629–638, 1986.

[300] H. Hagen. Geometric surface patches without twist constraints. *Computer Aided Geometric Design*, 3(3):179–184, 1986.

[301] H. Hagen and G. Bonneau. Variational design of smooth rational Bezier curves. *Computer Aided Geometric Design*, 8(5):393–400, 1991.

[302] H. Hagen and H. Pottmann. Curvature continuous triangular interpolants. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 373–384. Academic Press, 1989.

[303] H. Hagen and G. Schulze. Automatic smoothing with geometric surface patches. *Computer Aided Geometric Design*, 4(3):231–236, 1987.

[304] S. Hahmann. Shape improvement of surfaces. In G. Farin, H. Bieri, G. Brunnett, and T. DeRose, editors, *Geometric Modeling*, pages 135–152. Springer-Verlag, Vienna, 1998.

[305] S. Hahmann and S. Kontz. Knot-removal surface fairing using search strategies. *Computer Aided Design*, 30(2):131–138, 1998.

[306] J. Hahn. Geometric continuous patch complexes. *Computer Aided Geometric Design*, 6(1):55–67, 1989.

[307] R. Hall and G. Mullineux. The Zheng–Ball construction without twist constraints. *Computer Aided Geometric Design*, 16(3):165–175, 1999.

[308] M. Halstead, M. Kass, and T. DeRose. Efficient, fair interpolation using Catmull-Clark surfaces. *Computer Graphics*, pages 35–44, 1993. SIGGRAPH 1993 Proceedings.

[309] B. Hamann, G. Farin, and G. Nielson. G^1 surface interpolation based on degree elevated conics. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 75–86. SIAM, 1991.

[310] J. Hands. Reparametrisation of rational surfaces. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 87–100. Oxford University Press, 1987.

[311] S. Hanna, J. Abel, and D. Greenberg. Intersection of parametric surfaces by means of lookup tables. *IEEE Computer Graphics and Applications*, 3(7):39–48, 1983.

[312] D. Hansford. The neutral case for the min-max triangulation. *Computer Aided Geometric Design*, 7(5):431–438, 1990.

[313] D. Hansford. Bézier techniques. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of 3D Modeling and Graphics*. Elsevier, 2002. In preparation.

[314] P. Hartley and C. Judd. Parametrization of Bézier-type B-spline curves. *Computer Aided Design*, 10(2):130–134, 1978.

[315] P. Hartley and C. Judd. Parametrization and shape of B-spline curves. *Computer Aided Design*, 12(5):235–238, 1980.

[316] J. Hayes. New shapes from bicubic splines. Technical report, National Physics Laboratory, 1974.

[317] L. Hering. Closed C^2 and C^3 continuous Bézier and B-spline curves with given tangents. *Computer Aided Design*, 15(1):3–6, 1983.

[318] T. Hermann. On the smoothness of offset surfaces. *Computer Aided Geometric Design*, 15(5):529–533, 1998.

[319] T. Hermann. On the derivatives of second and third degree rational Bézier curves. *Computer Aided Geometric Design*, 16(3):157–163, 1999.

[320] T. Hermann and G. Renner. Subdivision of n-sided regions into four-sided patched. In D. C. Handscomb, editor, *The Mathematics of Surfaces III*, pages 347–358. Clarendon Press, 1989.

[321] G. Herron. Smooth closed surfaces with discrete triangular interpolants. *Computer Aided Geometric Design*, 2(4):297–306, 1985.

[322] G. Herron. Techniques for visual continuity. In G. Farin, editor, *Geometric Modeling*, pages 163–174. SIAM, Philadelphia, 1987.

[323] D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea, New York, 1952.

[324] B. Hinds, J. McCartney, and G. Woods. Pattern development for 3d surfaces. *Computer Aided Design*, 23(8):583–592, 1991.

[325] H. Hochfeld and M. Ahlers. Role of Bézier curves and surfaces in the Volkswagen CAD approach from 1967 to today. *Computer Aided Design*, 22(9):598–608, 1990.

[326] G. Hoelzle. Knot placement for piecewise polynomial approximation of curves. *Computer Aided Design*, 15(5):295–296, 1983.

[327] C. Hoffmann. *Geometric & Solid Modeling*. Morgan Kaufmann, 1989.

[328] D. Hoitsma and M. Lee. Generalized rational B-spline surfaces. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 87–102. SIAM, 1991.

[329] K. Höllig and H. Mogerle. G-splines. *Computer Aided Geometric Design*, 7(1-4):197–208, 1990.

[330] H. Hoppe. Progressive meshes. In *SIGGRAPH '96 Proc.*, pages 99–108, August 1996. *http://research.microsoft.com/hoppe/*.

[331] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proc.*, pages 19–26, August 1993. *http://research.microsoft.com/hoppe/*.

[332] M. Hosaka and F. Kimura. Non-four-sided patch expressions with control points. *Computer Aided Geometric Design*, 1(1):75–86, 1984.

[333] J. Hoschek. Detecting regions with undesirable curvature. *Computer Aided Geometric Design*, 1(2):183–192, 1984.

[334] J. Hoschek. Offset curves in the plane. *Computer Aided Design*, 17(2):77–82, 1985.

[335] J. Hoschek. Smoothing of curves and surfaces. *Computer Aided Geometric Design*, 2(1–3):97–105, 1985.

[336] J. Hoschek. Approximate conversion of spline curves. *Computer Aided Geometric Design*, 4(1–2):59–66, 1987.

[337] J. Hoschek. Intrinsic parametrization for approximation. *Computer Aided Geometric Design*, 5(1):27–31, 1988.

[338] J. Hoschek. Spline approximation of offset curves. *Computer Aided Geometric Design*, 5(1):33–40, 1988.

[339] J. Hoschek and D. Lasser. *Grundlagen der Geometrischen Datenverarbeitung*. B.G. Teubner, Stuttgart, 1989. English translation: Fundamentals of Computer Aided Geometric Design, AK Peters, 1993.

[340] J. Hoschek and F. Schneider. Spline conversion for trimmed rational Bézier- and B-spline surfaces. *Computer Aided Design*, 22(9):580–590, 1990.

[341] J. Hoschek and N. Wissel. Optimal approximate conversion of spline curves and spline approximation of offset curves. *Computer Aided Design*, 20(8):475–483, 1988.

[342] E. Houghton, E. Emnett, R. Factor, and L. Sabharwal. Implementation of a divide-and-conquer-method for the intersection of parametric surfaces. *Computer Aided Geometric Design*, 2(1–3):173–184, 1985.

[343] T. Jensen. Assembling triangular and rectangular patches and multivariate splines. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 203–220. SIAM, Philadelphia, 1987.

[344] T. Jensen, C. Petersen, and M. Watkins. Practical curves and surfaces for a geometric modeler. *Computer Aided Geometric Design*, 8(5):357–370, 1991.

[345] B. Joe. Knot insertion for beta-spline curves and surfaces. *ACM Transactions on Graphics*, 9(1):41–66, 1990.

[346] S. Jolles. *Die Theorie der Oskulanten und das Sehnensystem der Raumkurve 4. Ordnung, 2. Spezies.* PhD thesis, Technical Univ. Aachen, 1886.

[347] A. Jones. An algorithm for convex parametric splines. Technical Report ETA-TR-29, Boeing Computer Services, 1985.

[348] A. Jones. Shape control of curves and surfaces through constrained optimization. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 265–279. SIAM, Philadelphia, 1987.

[349] A. Jones. Nonrectangular surface patches with curvature continuity. *Computer Aided Design*, 20(6):325–335, 1988.

[350] J. Kahmann. Continuity of curvature between adjacent Bézier patches. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 65–76. North-Holland, 1983.

[351] M. Kallay and B. Ravani. Optimal twist vectors as a tools for interpolating a network of curves with a minimum surface energy. *Computer Aided Geometric Design*, 7(6):465–474, 1990.

[352] P. Kashyap. Geometric interpretation of continuity over triangular domains. *Computer Aided Geometric Design*, 15(9):773–786, 1998.

[353] K. Kato. Generation of *n*-sided surface patches with holes. *Computer Aided Design*, 23(10):676–683, 1991.

[354] E. Kaufmann and R. Klass. Smoothing surfaces using reflection lines for families of splines. *Computer Aided Design*, 20(6):312–316, 1988.

[355] A. Khodakovosky, P. Schroeder, and W. Sweldens. Progressive geometry compression. *Computer Graphics*, 34:271–278, 2000. Proceedings SIGGRAPH 2000.

[356] P. Kiciak. Constructions of G^1 continuous joins of rational Bézier patches. *Computer Aided Geometric Design*, 12(3):283–303, 1995.

[357] D. Kim. *Cones on Bézier curves and surfaces.* PhD thesis, Industrial and Operations Engineering Dept, U. of Michigan at Ann Arbor, 1990.

[358] J. Kjellander. Smoothing of bicubic parametric surfaces. *Computer Aided Design*, 15(5):288–293, 1983.

[359] J. Kjellander. Smoothing of cubic parametric splines. *Computer Aided Design*, 15(3):175–179, 1983.

[360] R. Klass. Correction of local surface irregularities using reflection lines. *Computer Aided Design*, 12(2):73–77, 1980.

[361] R. Klass. An offset spline approximation for plane cubics. *Computer Aided Design*, 15(5):296–299, 1983.

[362] L. Kobbelt. $\sqrt{3}$ subdivision. *Computer Graphics*, pages 103–112, 2000. SIGGRAPH 2000.

[363] L. Kocić. Modification of Bézier curves and surfaces by degree elevation technique. *Computer Aided Design*, 23(10):692–699, 1991.

[364] P. Korovkin. *Linear Operators and Approximation Theory*. Hindustan Publishing Co., Delhi, 1960.

[365] M. Kosters. Curvature-dependent parametrization of curves and surfaces. *Computer Aided Design*, 23(8):569–578, 1991.

[366] M. Lachance and A. Schwartz. Four point parabolic interpolation. *Computer Aided Geometric Design*, 8(2):143–150, 1991.

[367] C. Lacombe and C. Bédard. Interpolation function over a general triangular mid-edge finite element. *Comp. Math. Appl.*, 12A(3):362–373, 1986.

[368] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting*. Academic Press, 1986.

[369] J. Lane and R. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Analysis Machine Intell.*, 2(1):35–46, 1980.

[370] J. Lane and R. Riesenfeld. A geometric proof for the variation diminishing property of B-spline approximation. *J of Approx. Theory*, 37:1–4, 1983.

[371] D. Lasser. Bernstein-Bézier representation of volumes. *Computer Aided Geometric Design*, 2(1–3):145–150, 1985.

[372] D. Lasser. Intersection of parametric surfaces in the Bernstein-Bézier representation. *Computer Aided Design*, 18(4):186–192, 1986.

[373] D. Lasser and G. Bonneau. Bézier representation of trim curves. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modeling*, pages 227–242. Springer, Vienna, 1995.

[374] D. Lasser and A. Purucker. B-spline-Bézier representations of rational geometric spline curves: quartics and quintics. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 115–130. SIAM, 1991.

[375] C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1971.

[376] C. Lawson and R. Hanson. *Solving Least Squares Problems*. SIAM, 1995. Republication of same title by Prentice Hall 1974.

[377] E. Lee. The rational Bézier representation for conics. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 3–19. SIAM, Philadelphia, 1987.

[378] E. Lee. Choosing nodes in parametric curve interpolation. *Computer Aided Design*, 21(6), 1989. Presented at the SIAM Applied Geometry meeting, Albany, N.Y., 1987.

[379] E. Lee. A note on blossoming. *Computer Aided Geometric Design*, 6(4):359–362, 1989.

[380] E. Lee. Energy, fairness, and a counterexample. *Computer Aided Design*, 22(1):37–40, 1990.

[381] E. Lee and M. Lucian. Moebius reparametrizations of rational B-splines. *Computer Aided Geometric Design*, 8(3):213–216, 1991.

[382] I.-K. Lee. Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, 17(2):161–177, 2000.

[383] S. Leopoldseder and H. Pottmann. Approximation of developable surfaces with cone spline surfaces. *Computer Aided Design*, 30(7):571–582, 1998.

[384] J. Levin. Mathematical models for determining the intersection of quadric surfaces. *Computer Graphics and Image Processing*, 11:73–87, 1979.

[385] G. Levner, P. Tassinari, and D. Marini. A simple method for raytracing bicubic surfaces. In R. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 805–820. Springer Verlag, 1988.

[386] M. Levoy et al. The Digital Michelangelo Project: 3D scanning of large statues. *SIGGRAPH 2000*, pages 131–144, 2000.

[387] J. Li, J. Hoschek, and E. Hartmann. G^{n-1} functional splines for interpolation and approximation of curves, surfaces and solids. *Computer Aided Geometric Design*, 7(1–4):209–220, 1990.

[388] Y.-M. Li and X.-Y. Zhang. Basis conversion among Bézier, Tchebyshev and Legendre. *Computer Aided Geometric Design*, 15(6):637–642, 1998.

[389] S. Lien, M. Shantz, and V. Pratt. Adaptive forward differencing for rendering curves and surfaces. *Computer Graphics*, 21, 1987. SIGGRAPH '87 proceedings.

[390] R. Liming. *Practical analytical geometry with applications to aircraft*. Macmillan, 1944.

[391] R. Liming. *Mathematics for Computer Graphics*. Aero publishers, 1979.

[392] Q. Lin and J. Rokne. Disk Bézier curves. *Computer Aided Geometric Design*, 15(7):721–737, 1998.

[393] N. Litke, A. Levin, and P. Schroeder. Trimming for subdivision surfaces. *Computer Aided Geometric Design*, 18(5):463–482, 2001.

[394] D. Liu. GC^1 continuity conditions between two adjacent rational Bézier surface patches. *Computer Aided Geometric Design*, 7(1–4):151–164, 1990.

[395] D. Liu and J. Hoschek. GC^1 continuity conditions between adjacent rectangular and triangular Bézier surface patches. *Computer Aided Design*, 21(4):194–200, 1989.

[396] S. Lodha and J. Warren. Bézier representation for quadric surface patches. *Computer Aided Design*, 22(9):574–579, 1990.

[397] C. Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11(3):303–330, 1994.

[398] C. Loop and T. DeRose. Generalized B-spline surfaces of arbitrary topology. *Computer Graphics*, 24(4):347–356, 1990.

[399] G. Lorentz. *Bernstein Polynomials*. Toronto press, 1953. Second edition, Chelsea 1986.

[400] M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1994.

[401] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Trans. on Graphics*, 16(1):34–73, 1997.

[402] M. Lounsbery, S. Mann, and T. DeRose. Parametric surface interpolation. *IEEE Computer Graphics and Applications*, 12(5):45–52, 1992.

[403] M. Lucian. Linear fractional transformations of rational Bézier curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 131–139. SIAM, Philadelphia, 1991.

[404] D. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 20:24–35, 2001.

[405] G. Lukaćs. Differential geometry of G1 variable radius rolling ball blend surfaces. *Computer Aided Geometric Design*, 15(6):585–613, 1998.

[406] D. Lutterkort, J. Peters, and U. Reif. Polynomial degree reduction in the L2-norm equals best Euclidean approximation of Bézier coefficients. *Computer Aided Geometric Design*, 16(7):607–612, 2000.

[407] T. Lyche and V. Morken. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design*, 4(3):217–230, 1987.

[408] W. Ma and J.-P. Kruth. Mathematical modelling of free-form curves and surfaces from discrete points with NURBS. In P.-J. Laurent, A. Le Méhauté, and L. Schumaker, editors, *Curves and Surfaces in Geometric Design*, pages 319–326. A K Peters, Ltd., 1994.

[409] T. Maekawa and N. Patrikalakis. Computation of singularities and intersections of offsets and planar curves. *Computer Aided Geometric Design*, 10(5):407–429, 1993.

[410] T. Maekawa, F.-E. Wolter, and N. Patrikalakis. Umbilics and lines of curvature for shape interrogation. *Computer Aided Geometric Design*, 13(2):133–163, 1996.

[411] B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, San Francisco, 1983.

[412] S. Mann. Cubic precision Clough-Tocher interpolation. *Computer Aided Geometric Design*, 16(2):85–88, 1999.

[413] S. Mann and T. DeRose. Computing values and derivatives of Bézier and B-spline tensor products. *Computer Aided Geometric Design*, 12(1):107–109, 1995.

[414] J. Manning. Continuity conditions for spline curves. *The Computer J*, 17(2):181–186, 1974.

[415] D. Manocha and J. Canny. Rational curves with polynomial parametrization. *Computer Aided Design*, 23(9):645–652, 1991.

[416] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Md, 1988.

[417] R. Markot and R. Magedson. Solutions of tangential surface and curve intersections. *Computer Aided Design*, 21(7):421–429, 1989.

[418] R. Markot and R. Magedson. Procedural method for evaluating the intersection curves of two parametric surfaces. *Computer Aided Design*, 23(6):395–404, 1991.

[419] D. Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer-Verlag, 1999.

[420] J. Marshall and A. Mitchell. Blending interpolants in the finite element method. *Int. J Numer. Meth. Eng.*, 12:77–83, 1978.

[421] D. McConalogue. A quasi-intrinsic scheme for passing a smooth curve through a discrete set of points. *The Computer J*, 13:392–396, 1970.

[422] D. McConalogue. Algorithm 66—an automatic French-curve procedure for use with an incremental plotter. *The Computer J*, 14:207–209, 1971.

[423] H. McLaughlin. Shape preserving planar interpolation: an algorithm. *IEEE Computer Graphics and Applications*, 3(3):58–67, 1985.

[424] A. Meek and R. Thomas. A guided clothoid spline. *Computer Aided Geometric Design*, 8(2):163–174, 1991.

[425] E. Mehlum. Nonlinear splines. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 173–208. North-Holland, 1974.

[426] C. Micchelli and H. Prautzsch. Computing surfaces invariant under subdivision. *Computer Aided Geometric Design*, 4(4):321–328, 1987.

[427] J. Miller. Sculptured surfaces in solid models: issues and alternative approaches. *IEEE Computer Graphics and Applications*, 6(12):37–48, 1986.

[428] C. Millham and A. Meyer. Modified Hermite quintic curves and applications. *Computer Aided Design*, 23(10):707–712, 1991.

[429] F. Moebius. *August Ferdinand Moebius, Gesammelte Werke*. Verlag von S. Hirzel, 1885. also published by Dr. M. Saendig oHG, Wiesbaden, FRG, 1967.

[430] P. Montès. Kriging interpolation of a Bézier curve. *Computer Aided Design*, 23(10):713–716, 1991.

[431] H. Moreton and C. Sequin. Minimum variation curves and surfaces for CAGD. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 123–160. SIAM, Philadelphia, 1994.

[432] G. Morin, J. Warren, and H. Weimer. A subdivision scheme for surfaces of revolution. *Computer Aided Geometric Design*, 18(5):483–502, 2001.

[433] M. Mortenson. *Geometric Modeling*. Wiley, 1985.

[434] F. Munchmeyer. On surface imperfections. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 459–474. Oxford University Press, 1987.

[435] F. Munchmeyer. Shape interrogation: A case study. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 291–301. SIAM, Philadelphia, 1987.

[436] L. Nachman. Blended tensor product B-spline surface. *Computer Aided Design*, 20(6):336–340, 1988.

[437] L. Nachman. A note on control polygons and derivatives. *Computer Aided Geometric Design*, 8(3):223–226, 1991.

[438] D. Nairn, J. Peters, and D. Lutterkort. Sharp, quantitative bounds on the distance between a polynomial piece and its Bézier control polygon. *Computer Aided Geometric Design*, 16(7):613–631, 2000.

[439] A. Nasri. Boundary-corner control in recursive-subdivision surfaces. *Computer Aided Design*, 23(6):405–411, 1991.

[440] A. Nasri. Surface interpolation on irregular networks with normal conditions. *Computer Aided Geometric Design*, 8(1):89–96, 1991.

[441] A. Nasri. Recursive subdivision of polygonal complexes and its applications in CAGD. *Computer Aided Geometric Design*, 17:595–619, 2000.

[442] G. Nielson. Some piecewise polynomial alternatives to splines under tension. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 209–235. Academic Press, 1974.

[443] G. Nielson. The side-vertex method for interpolation in triangles. *J of Approx. Theory*, 25:318–336, 1979.

[444] G. Nielson. Minimum norm interpolation in triangles. *SIAM J Numer. Analysis*, 17(1):46–62, 1980.

[445] G. Nielson. A rectangular nu-spline for interactive surface design. *IEEE Computer Graphics and Applications*, 6(2):35–41, 1986.

[446] G. Nielson. Coordinate free scattered data interpolation. In L. Schumaker, editor, *Topics in Multivariate Approximation*. Academic Press, 1987.

[447] G. Nielson. A transfinite, visually continuous, triangular interpolant. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 235–246. SIAM, Philadelphia, 1987.

[448] G. Nielson. A characterization of an affine invariant triangulation. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling*, pages 191–210. Springer, Vienna, 1993.

[449] G. Nielson and T. Foley. A survey of applications of an affine invariant norm. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in CAGD*, pages 445–467. Academic Press, 1989.

[450] H. Nowacki. Splines in shipbuilding. *Proc. 21st Duisburg colloquium on marine technology*, May 2000.

[451] H. Nowacki, D. Liu, and X. Lu. Fairing Bézier curves with constraints. *Computer Aided Geometric Design*, 7(1–4):43–56, 1990.

[452] A. Overhauser. Analytic definition of curves and surfaces by parabolic blending. Technical report, Ford Motor Company, 1968.

[453] D. Paglieroni. The directional parameter plane transform of a height field. *ACM Transactions on Graphics*, 17(1):50–70, 1998.

[454] M. Paluszny and W. Boehm. General cyclides. *Computer Aided Geometric Design*, 15(7):699–710, 1998.

[455] D. Parkinson and D. Moreton. Optimal biarc-curve fitting. *Computer Aided Design*, 23(6):411–419, 1991.

[456] N. Patrikalakis. Shape interrogation. In C. Chrystossomidis, editor, *Automation in the Design and Manufacture of Large Marine Systems*, pages 83–104. Hemisphere, New York, 1990.

[457] R. Patterson. Projective transformations of the parameter of a rational Bernstein-Bézier curve. *ACM Transactions on Graphics*, 4:276–290, 1986.

[458] T. Pavlidis. Curve fitting with conic splines. *ACM Transactions on Graphics*, 2(1):1–31, 1983.

[459] J. Pegna and F. Wolter. Geometric criteria to guarantee curvature continuity of blend surfaces. *ASME Transactons, J. of Mech. Design*, 114, 1992.

[460] Q. Peng. An algorithm for finding the intersection lines between two B-spline surfaces. *Computer Aided Design*, 16(4):191–196, 1984.

[461] M. Penna and R. Patterson. *Projective Geometry and Its Applications to Computer Graphics.* Prentice Hall, 1986.

[462] G. Peters. Interactive computer graphics application of the parametric bicubic surface to engineering design problems. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 259–302. Academic Press, 1974.

[463] J. Peters. Local cubic and bicubic C^1 surface interpolation with linearly varying boundary normal. *Computer Aided Geometric Design*, 7(6):499–516, 1990.

[464] J. Peters. Local smooth surface interpolation: a classification. *Computer Aided Geometric Design*, 7(1–4):191–196, 1990.

[465] J. Peters. Smooth mesh interpolation with cubic patches. *Computer Aided Design*, 22(2):109–120, 1990.

[466] J. Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7(2):221–247, 1991.

[467] J. Peters. Constructiong C^1 surfaces of arbitrary topology using biquadratic and bicubic splines. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 277–294. SIAM, Philadelphia, 1994.

[468] J. Peters. Patching Catmull-Clark meshes. *Computer Graphics*, pages 255–258, 2000. Proceedings SIGGRAPH 2000.

[469] J. Peters and U. Reif. The 42 equivalence classes of quadratic surfaces in affine *n*-space. *Computer Aided Geometric Design*, 15(5):459–474, 1998.

[470] J. Peters and G. Umlauf. Computing curvature bounds for bounded curvature subdivision. *Computer Aided Geometric Design*, 18(5):455–462, 2001.

[471] C. Petersen. Adaptive contouring of three-dimensional surfaces. *Computer Aided Geometric Design*, 1(1):61–74, 1984.

[472] J. Peterson. Degree reduction of Bézier curves. *Computer Aided Design*, 23(6):460–461, 1991. Letter to the editor.

[473] H. N. Phien and N. Dejdumrong. Efficient algorithms for Bézier curves. *Computer Aided Geometric Design*, 17(3):247–250, 2000.

[474] L. Piegl. A geometric investigation of the rational Bézier scheme in computer aided geometric design. *Computers in Industry*, 7(5):401–410, 1986.

[475] L. Piegl. The sphere as a rational Bézier surface. *Computer Aided Geometric Design*, 3(1):45–52, 1986.

[476] L. Piegl. Interactive data interpolation by rational Bézier curves. *IEEE Computer Graphics and Applications*, 7(4):45–58, 1987.

[477] L. Piegl. On the use of infinite control points in CAGD. *Computer Aided Geometric Design*, 4(1–2):155–166, 1987.

[478] L. Piegl. Hermite- and Coons-like interpolants using rational Bézier approximation form with infinite control points. *Computer Aided Design*, 20(1):2–10, 1988.

[479] L. Piegl. On NURBS: a survey. *Computer Graphics and Applications*, 11(1):55–71, 1990.

[480] L. Piegl and W. Tiller. Curve and surface constructions using rational B-splines. *Computer Aided Design*, 19(9):485–498, 1987.

[481] L. Piegl and W. Tiller. Algorithm for degree reduction of B-spline curves. *Computer Aided Design*, 27(2):101–110, 1995.

[482] L. Piegl and W. Tiller. *The NURBS Book*. Springer Verlag, 1997. Second edition.

[483] B. Piper. Visually smooth interpolation with triangular Bézier patches. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 221–233. SIAM, Philadelphia, 1987.

[484] D. Plowman and P. Charrot. A practical implementation of vertex blend surfaces using an *n*-sided patch. In Glen Mullineux, editor, *The Mathematics of Surfaces VI*, pages 67–78. Oxford University Press, 1996.

[485] A. Pobegailo. Local interpolation with weight functions for variable-smoothness curve design. *Computer Aided Design*, 23(8):579–582, 1991.

[486] T. Poeschl. Detecting surface irregularities using isophotes. *Computer Aided Geometric Design*, 1(2):163–168, 1984.

[487] H. Pottmann. Curves and tensor product surfaces with third order geometric continuity. In S. Slaby and H. Stachel, editors, *Proceedings of the Third International Conference on Engineering Graphics and Descriptive Geometry*, pages 107–116, 1988.

[488] H. Pottmann. Projectively invariant classes of geometric continuity for CAGD. *Computer Aided Geometric Design*, 6(4):307–322, 1989.

[489] H. Pottmann. A projectively invariant characterization of G^2 continuity for rational curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 141–148. SIAM, Philadelphia, 1991.

[490] M. Powell and M. Sabin. Piecewise quadratic approximation on triangles. *ACM Trans. Math. Software*, 3(4):316–325, 1977.

[491] M. Pratt. Cyclides in computer aided geometric design. *Computer Aided Geometric Design*, 7(1–4):221–242, 1990.

[492] M. Pratt and A. Geisow. Surface/surface intersection problems. In J. Gregory, editor, *The Mathematics of Surfaces*. Clarendon Press, 1986.

[493] H. Prautzsch. Degree elevation of B-spline curves. *Computer Aided Geometric Design*, 1(12):193–198, 1984.

[494] H. Prautzsch. On Degen's conjecture. *Computer Aided Geometric Design*, 11(5):593–596, 1994.

[495] H. Prautzsch and C. Micchelli. Computing curves invariant under halving. *Computer Aided Geometric Design*, 4(1–2):133–140, 1987.

[496] H. Prautzsch and B. Piper. A fast algorithm to raise the degree of B-spline curves. *Computer Aided Geometric Design*, 8(4):253–266, 1991.

[497] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[498] L. Ramshaw. Blossoming: a connect-the-dots approach to splines. Technical report, Digital Systems Research Center, Palo Alto, Ca, 1987.

[499] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6(4):323–359, 1989.

[500] T. Rando and J. Roulier. Designing faired parametric surfaces. *Computer Aided Design*, 23(7):492–497, 1991.

[501] A. Razdan and G. Farin. Determination of end conditions for NURB surface interpolation. *Computer Aided Geometric Design*, 15(7):757–768, 1998.

[502] D. Reese, M. Reidger, and R. Lang. Flaechenhaftes Glaetten und Veraendern von Schiffsoberflaechen. Technical Report MTK 0243, T. U. Berlin, 1983.

[503] U. Reif. A unified approach to subdivision algorithms near extraordinary vartices. *Computer Aided Geometric Design*, 12(2):153–174, 1995.

[504] U. Reif. On the local existence of the quadratic geometric Hermite interpolant. *Computer Aided Geometric Design*, 16(3):217–221, 1999.

[505] G. Renner. Inter-patch continuity of surfaces. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 237–254. Oxford University Press, 1987.

[506] A. Renyi. *Wahrscheinlichkeitsrechnung*. VEB Deutscher Verlag der Wissenschaften, 1962.

[507] W. Renz. Interactive smoothing of digitized point data. *Computer Aided Design*, 14(5):267–269, 1982.

[508] R. Riesenfeld. On Chaikin's algorithm. *Computer Graphics and Image Processing*, 4(3):304–310, 1975.

[509] D. Rogers and L. Adlum. Dynamic rational B-spline surfaces. *Computer Aided Design*, 22(9):609–616, 1990.

[510] P. Rosin. A note on the least squares fitting of ellipses. *Pattern Recognition Letters*, 14:799–808, 1995.

[511] J. Rossignac and A. Requicha. Offsetting operations in solid modelling. *Computer Aided Geometric Design*, 3(2):129–148, 1986.

[512] J. Roulier and E. Passow. Monotone and convex spline interpolation. *SIAM J Numer. Analysis*, 14(5):904–909, 1977.

[513] C. Runge. Ueber empirische Funktionen und die Interpolation zwischen aequidistanten Ordinaten. *ZAMM*, 46:224–243, 1901.

[514] M. Sabin. General interrogations of parametric surfaces. Technical Report VTO/MS/150, British Aircraft Corporation, 1968.

[515] M. Sabin. A method for displaying the intersection curve of two quadric surfaces. *The Computer J*, 19:336–338, 1976.

[516] M. Sabin. *The use of piecewise forms for the numerical representation of shape*. PhD thesis, Hungarian Academy of Sciences, Budapest, Hungary, 1976.

[517] M. Sabin. Recursive subdivision. In J. Gregory, editor, *The Mathematics of Surfaces*, pages 269–281. Clarendon Press, 1986.

[518] M. Sabin. Some negative results in n-sided patches. *Computer Aided Design*, 18(1):38–44, 1986.

[519] M. Sabin. Recursive subdivision surfaces. In *Handbook of CAGD*. Elsevier, 2001.

[520] M. Sabin and F. Kimura. Letters to the editor. *Computer Aided Geometric Design*, 1(3):289–290, 1984. Concerning n-sided patches.

[521] P. Sablonnière. Spline and Bézier polygons associated with a polynomial spline curve. *Computer Aided Design*, 10(4):257–261, 1978.

[522] P. Sablonnière. *Bases de Bernstein et approximants splines*. PhD thesis, Univ. of Lille, 1982.

[523] P. Sablonnière. Interpolation by quadratic splines on triangles and squares. *Computers in Industry*, 3:45–52, 1982.

[524] P. Sablonnière. Bernstein-Bézier methods for the construction of bivariate spline approximants. *Computer Aided Geometric Design*, 2(1–3):29–36, 1985.

[525] P. Sablonnière. Composite finite elements of class C^k. *J of Computational and Appl. Math.*, 12,13:542–550, 1985.

[526] M. Sakai. Inflection points and singularities on planar rational cubic curve segments. *Computer Aided Geometric Design*, 16(3):149–156, 1999.

[527] K. Salkauskas. C^1 splines for interpolation of rapidly varying data. *Rocky Mtn. J of Math.*, 14(1):239–250, 1984.

[528] J. Sánchez-Reyes. Single-valued curves in polar coordinates. *Computer Aided Design*, 22(1):19–26, 1990.

[529] J. Sánchez-Reyes. Single-valued surfaces in cylindrical coordinates. *Computer Aided Design*, 23(8):561–568, 1991.

[530] J. Sánchez-Reyes. The symmetric analogue of the polynomial power basis. *ACM Transactions on Graphics*, 16(3):319–357, 1997.

[531] J. Sánchez-Reyes. Harmonic rational Bézier curves, p-Bézier curves and trigonometric polynomials. *Computer Aided Geometric Design*, 15(9): 909–923, 1998.

[532] N. Sapidis, editor. *Designing Fair Curves and Surfaces*. SIAM, Philadelphia, 1994.

[533] N. Sapidis and G. Farin. Automatic fairing algorithm for B-spline curves. *Computer Aided Design*, 22(2):121–129, 1990.

[534] B. Sarkar and C. Meng. Parameter optimization in approximating curves and surfaces to measurement data. *Computer Aided Geometric Design*, 8(4):267–290, 1991.

[535] B. Sarkar and C-H. Meng. Smooth-surface approximation and reverse engineering. *Computer Aided Design*, 23(9):623–628, 1991.

[536] R. Sarraga. G^1 interpolation of generally unrestricted cubic Bézier curves. *Computer Aided Geometric Design*, 4(1–2):23–40, 1987.

[537] R. Sarraga. Errata: G^1 interpolation of generally unrestricted cubic Bézier curves. *Computer Aided Geometric Design*, 6(2):167–172, 1989.

[538] R. Sarraga. Recent methods for surface shape optimization. *Computer Aided Geometric Design*, 15(5):417–436, 1998.

[539] J. Schaaf and B. Ravani. Geometric continuity of ruled surfaces. *Computer Aided Geometric Design*, 15(3):289–310, 1998.

[540] J. Schelske. *Lokale Glaettung segmentierter Bézierkurven und Bézierflaechen*. PhD thesis, TH Darmstadt, Germany, 1984.

[541] F. Schneider. Interpolation and approximation using rational B-splines. Technical report, TH Darmstadt, 1993.

[542] I. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99, 1946.

[543] I. Schoenberg. On variation diminishing approximation methods. In R. E. Langer, editor, *On Numerical Approximation*, pages 249–274. Univ. of Wisconsin Press, 1953.

[544] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 1996. Code at *http://www.cs.rpi.edu:80/ martink/*.

[545] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, July 1992.

[546] L. Schumaker. *Spline Functions: Basic Theory*. Wiley, 1981.

[547] L. Schumaker. Bounds on the dimension of spaces of multivariate piecewise polynomials. *Rocky Mtn. J of Math.*, 14(1):251–264, 1984.

[548] L. Schumaker and W. Volk. Efficient evaluation of multivariate polynomials. *Computer Aided Geometric Design*, 3(2):149–154, 1986.

[549] A. Schwartz. Subdividing Bézier curves and surfaces. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 55–66. SIAM, Philadelphia, 1987.

[550] T. Sederberg. Improperly parametrized rational curves. *Computer Aided Geometric Design*, 3(1):67–75, 1986.

[551] T. Sederberg. Point and tangent computation of tensor product rational Bézier surfaces. *Computer Aided Geometric Design*, 12(1):103–106, 1995.

[552] T. Sederberg and D. Anderson. Steiner surface patches. *IEEE Computer Graphics and Applications*, 5(5):23–36, 1985.

[553] T. Sederberg, H. Christiansen, and S. Katz. Improved test for closed loops in surface intersections. *Computer Aided Design*, 21(8):505–508, 1989.

[554] T. Sederberg and M. Kakimoto. Approximating rational curves using polymial curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 149–158. SIAM, 1991.

[555] T. Sederberg and R. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5(2):161–171, 1988.

[556] T. Sederberg and T. Nishita. Geometric Hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8(2):97–114, 1991.

[557] T. Sederberg and S. Parry. A comparison of three curve intersection algorithms. *Computer Aided Design*, 18(1):58–63, 1986.

[558] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986. SIGGRAPH proceedings.

[559] T. Sederberg and X. Wang. Rational hodographs. *Computer Aided Geometric Design*, 4(4):333–335, 1987.

[560] T. Sederberg, S. White, and A. Zundel. Fat arcs: a bounding region with cubic convergence. *Computer Aided Geometric Design*, 6(3):205–218, 1989.

[561] T. Sederberg, J. Zheng, D. Sewell, and M. Sabin. Non-uniform subdivision surfaces. *Computer Graphics*, 32:387–394, 1998. SIGGRAPH 1998.

[562] H.-P. Seidel. Knot insertion from a blossoming point of view. *Computer Aided Geometric Design*, 5(1):81–86, 1988.

[563] H.-P. Seidel. Computing B-spline control points. In W. Strasser and H.-P. Seidel, editors, *Theory and Practice of Geometric Modeling*, pages 17–32. Springer-Verlag, Berlin, 1989.

[564] H.-P. Seidel. A general subdivision theorem for Bézier triangles. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 573–582. Academic Press, 1989.

[565] H.-P. Seidel. A new multiaffine approach to B-splines. *Computer Aided Geometric Design*, 6(1):23–32, 1989.

[566] H.-P. Seidel. Symmetric triangular algorithms for curves. *Computer Aided Geometric Design*, 7(1–4):57–68, 1990.

[567] H.-P. Seidel. Computing B-spline control points using polar forms. *Computer Aided Design*, 23(9):634–640, 1991.

[568] H.-P. Seidel. Symmetric recursive algorithms for surfaces: B-patches and the de Boor algorithm for polynomials over triangles. *Constructive Approximation*, 7(2):257–279, 1991.

[569] H.-P. Seidel. Polar forms for geometrically continuous spline curves of arbitrary degree. *ACM Transactions on Graphics*, 12(1):1–34, 1993.

[570] S. Selesnick. Local invariants and twist vectors in CAGD. *Computer Graphics and Image Processing*, 17(2):145–160, 1981.

[571] M. Shantz and S. Chang. Rendering trimmed NURBS with adaptive forward differencing. *Computer Graphics*, 22(4):189–198, 1988.

[572] S. Shetty and P. White. Curvature-continuous extensions for rational B-spline curves and surfaces. *Computer Aided Design*, 23(7):484–491, 1991.

[573] L. Shirman and C. Séquin. Local surface interpolation with Bézier patches. *Computer Aided Geometric Design*, 4(4):279–295, 1987.

[574] L. Shirman and C. Séquin. Local surface interpolation with shape parameters between adjoining Gregory patches. *Computer Aided Geometric Design*, 7(5):375–388, 1990.

[575] L. Shirman and C. Séquin. Local surface interpolation with Bézier patches: errata and improvements. *Computer Aided Geometric Design*, 8(3):217–222, 1991.

[576] R. Sibson. A brief description of the natural neighbour interpolant. In V. Barnett, editor, *Interpreting Multivariate Data*. John Wiley & Sons, 1981.

[577] T. Speer, M. Kuppe, and J. Hoschek. Global reparametrization for curve approximation. *Computer Aided Geometric Design*, 15(9):869–877, 1998.

[578] E. Staerk. *Mehrfach differenzierbare Bézierkurven und Bézierflächen*. PhD thesis, T. U. Braunschweig, 1976.

[579] J. Stam. On subdivision schemes generalizing uniform B-spline surfaces of arbitrary degree. *Computer Aided Geometric Design*, 18(5):383–396, 2001.

[580] D. Stancu. Some Bernstein polynomials in two variables and their applications. *Soviet Mathematics*, 1:1025–1028, 1960.

[581] D. Storry and A. Ball. Design of an *n*-sided surface patch from Hermite boundary data. *Computer Aided Geometric Design*, 6(2):111–120, 1989.

[582] G. Strang and G. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, 1973.

[583] B.-Q. Su and D.-Y. Liu. *Computational Geometry.* Academic Press, 1989.

[584] H. Suzuki, S. Takeuchi, T. Kanai, and F. Kimura. Subdivision surface fitting to a range of points. *IEEE Computer Graphics and Applications*, pages 158–167, 1999.

[585] M. Szilvasi-Nagy. Flexible rounding operation for polyhedra. *Computer Aided Design*, 23(9):629–633, 1991.

[586] H. Theisel. Using Farin points for rational Bezier surfaces. *Computer Aided Geometric Design*, 16(8):817–835, 1999.

[587] H. Theisel and G. Farin. The curvature of characteristic curves on surfaces. *IEEE Computer Graphics and Applications*, pages 88–96, 1997.

[588] J. Thompson, Z. Warsi, and C. Mastin. *Numerical Grid Generation: Foundations and Applications.* North-Holland, 1985.

[589] W. Tiller. Rational B-splines for curve and surface representation. *IEEE Computer Graphics and Applications*, 3(6):61–69, 1983.

[590] W. Tiller and E. Hanson. Offsets of two-dimensional profiles. *IEEE Computer Graphics and Applications*, 4:36–46, 1984.

[591] P. Todd and R. McLeod. Numerical estimation of the curvature of surfaces. *Computer Aided Design*, 18(1):33–37, 1986.

[592] A. Toga, editor. *Brain Warping.* Academic Press, 1999.

[593] C. van Overveld. Family of recursively defined curves, related to the cubic Bézier curve. *Computer Aided Design*, 22(9):591–597, 1990.

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

[595] T. Várady. Survey and new results in *n*-sided patch generation. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 203–236. Oxford University Press, 1987.

[596] T. Várady. Overlap patches: a new scheme for interpolating curve networks with *n*-sided regions. *Computer Aided Geometric Design*, 8(1):7–26, 1991.

[597] T. Várady and A. Rockwood. Vertex blending based on the setback split. In M. Daehlen, T. Lyche, and L. Schumaker, editors, *Mathematical Meth-*

ods for Curve and Surface Design, pages 527–542. Vanderbilt University Press, 1995.

[598] L. Velho and D. Zorin. 4-8 subdivision. *Computer Aided Geometric Design*, 18(5):397–428, 2001.

[599] D. Vernet. Expression mathématique des formes. *Ingenieurs de l'Automobile*, 10:509–520, 1971.

[600] M. Veron, G. Ris, and J. Musse. Continuity of biparametric surface patches. *Computer Aided Design*, 8(4):267–273, 1976.

[601] K. Versprille. *Computer aided design applications of the rational B-spline approximation form.* PhD thesis, Syracuse U., 1975.

[602] M. Vigo and P. Brunet. Piecewise linear approximation of trimmed surfaces. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modeling*, page 341. Springer, Vienna, 1995.

[603] A. Vinacua and P. Brunet. A construction for VC^1 continuity for rational Bézier patches. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 601–611. Academic Press, 1989.

[604] M. Wagner and B. Ravani. Curves with rational Frenet–Serret motion. *Computer Aided Geometric Design*, 15(1):79–101, 1997.

[605] M. Walker. Current experience with transfinite interpolation. *Computer Aided Geometric Design*, 16(2):77–83, 1999.

[606] R. Walter. Visibility of surfaces via differential geometry. *Computer Aided Geometric Design*, 7(1–4):353, 1990.

[607] C. Wang. Shape classification of the parametric cubic curve and parametric B-spline cubic curve. *Computer Aided Design*, 13(4):199–206, 1981.

[608] L. Wang. Coons type blended B-spline surface and its conversion to NURBS surface. *Computer and Graphics*, 21(3):387–401, 1997.

[609] W. Wang and B. Joe. Interpolation on quadric surfaces with rational quadratic spline curves. *Computer Aided Geometric Design*, 14(3):207–230, 1997.

[610] J. Warren. Creating multisided rational Bézier surfaces using base points. *ACM Transactions on Graphics*, 11(2):127–139, 1992.

[611] J. Warren. Sparse filter banks for binary subdivision schemes. In T. Goodman and R. Martin, editors, *The Mathematics of Surfaces VII*, pages 427–438. Information Geometers, Winchester, UK, 1997.

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

[613] U. Wever. Optimal parametrization for cubic splines. *Computer Aided Design*, 23(9):641–644, 1991.

[614] R. Wielinga. Constrained interpolation using Bézier curves as a new tool in computer aided geometric design. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 153–172. Academic Press, 1974.

[615] G. Wolberg, S. Lee, and S. Shin. Scattered data interpolation with multilevel B-splines. *IEEE Visualization and Computer Graphics*, 17(3):228–244, 1997.

[616] F. Wolter and S. Tuohy. Curvature computations for degenerate surface patches. *Computer Aided Geometric Design*, 7, 1992.

[617] H. Wolters, G. Wu, and G. Farin. Degree reduction of B-spline curves. In G. Farin, H. Bieri, G. Brunnett, and T. DeRose, editors, *Geometric Modelling*. Springer-Verlag, 1998.

[618] A. Worsey and G. Farin. An *n*-dimensional Clough-Tocher element. *Constructive Approximation*, 3:99–110, 1987.

[619] A. Worsey and G. Farin. Contouring a bivariate quadratic polynomial over a triangle. *Computer Aided Geometric Design*, 7(1–4):337–352, 1990.

[620] Z. Xie and G. Farin. Hierarchical B-spline deformation with an application to brain imaging. In L. Schumaker and T. Lyche, editors, *Mathematical Methods in CAGD*. Vanderbilt University Press, 2001.

[621] F. Yamaguchi. *Curves and Surfaces in Computer Aided Geometric Design*. Springer, 1988.

[622] C. Yao and J. Rokne. An efficient algorithm for subdividing linear Coons surfaces. *Computer Aided Geometric Design*, 8(4):291–304, 1991.

[623] X. Ye. Curvature continuous interpolation of curve meshes. *Computer Aided Geometric Design*, 14(2):169–190, 1997.

[624] J. Yong, S. Hu, J. Sun, and X. Tan. Degree reduction of B-spline curves. *Computer Aided Geometric Design*, 18(2):117–128, 2001.

[625] A. Zenisek. Interpolation polynomials on the triangle. *Numerische Math.*, 15:283–296, 1970.

[626] A. Zenisek. Polynomial approximation on tetrahedrons in the finite element method. *J Approx. Theory*, 7:334–351, 1973.

[627] Y. Zhao and A. Rockwood. A convolution approach to *n*-sided patches and vertex blending. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 295–314. SIAM, Philadelphia, 1994.

[628] J. Zheng and A. Ball. Control point surfaces over non-four-sided areas. *Computer Aided Geometric Design*, 14(9):807–821, 1997.

[629] C.-Z. Zhou. On the convexity of parametric Bézier triangular surfaces. *Computer Aided Geometric Design*, 7(6):459–464, 1990.

[630] J. Zhou. *The positivity and convexity of Bézier polynomials over triangles*. PhD thesis, Beijing Univ., 1985.

[631] D. Zorin and P. Schroeder. A unified framework for primal/dual quadrilateral subdivision schemes. *Computer Aided Geometric Design*, 18(5):429–454, 2001.

This Page Intentionally Left Blank

Index

Abi-Ezzi, S., 116
absolute curvature, 425
Ackland, T., 173
adaptive forward differencing, 116
Adini's twist, 291
affine combination, 15
affine domain transformation, 105
affine geometry, 205
affine invariance, 29, 35, 60, 97
  of Bézier curves, 47
  of Bézier patches, 253
  of Bézier triangles, 311
affinely invariant, 44, 171
affine map, 17, 24
affine pair, 324, 367
affine parameter transformation, 162, 163
affine reparametrization, 87
affine space, 14
Aitken, A., 95
Aitken's algorithm, 97, 243
Akima, H., 174
Akima's interpolant, 177
Albrecht, G., 337
Alfeld, P., 327, 342
Andersson, R., 427
arc length, 159, 180, 191, 200, 429
  parametrization, 222
  of surface curve, 350
area element, 350
area, of triangle, 35
asymptotic line, 360
augment variation, 98

Baer, G., 195
Barnhill, R., 291, 309, 327, 399, 408, 414, 417
Barry, P., 144
Barsky, B., 195, 202
Bartels, R., 116, 377
barycentric combination, 15, 41, 97, 224
barycentric coordinates, 27, 34, 414
basis, 23
basis transformation, 75, 265
BBG interpolant, 414
Beck, J., 425
Bédard, C, 417
Bernstein polynomial, 8, 57
Bernstein, S., 57
Bessel end condition, 158, 163
Bessel tangent, 173, 177
Bessel twist, 292
Bézier abscissa, 266
Bézier curve
  piecewise, 137
Bézier function, 87
Bézier net, 248
Bézier ordinate, 87, 266

Bézier, P., 64, 89, 195, 301, 304
Bézier patch, trivariate, 302
Bézier points, 45
Bézier polygon, 45
Bézier surface, 248
Bézier tetrahedron, 333
biaffine, 247
bias, 195
bicubic Hermite interpolant, 405
bicubic Hermite patch, 276
bilinear interpolant, 245, 271, 292, 400
bilinearly blended Coons patch, 291, 402, 408
binomial coefficient, 57
binormal, 181
Birkhoff, G., 414, 420
Blaschke, W., 45
blending function, 402, 412
blossom, 31, 51, 74, 313
  of tensor product surface, 258
  for triangular patches, 313
Bobrow, J., 305
Boehm, W., 30, 43, 50, 119, 189, 194, 195, 202, 314, 340, 364, 367, 376
Bol, G., 189
Bonneau, G., 305
Boolean sum, 412, 414
boundary edge, 394
boundary point, 39
boundary vertex, 394
brain imaging, 302
Brianchon, C., 206
Brown, J., 41, 291
Brunet, P., 293, 305, 367
Brunnett, G., 305
B-spline polygon, 289, 419, 431
B-splines, 399, 431
butterfly scheme, 391

canonical coordinates, 181
canonical form, 184
cardinal, 99, 103, 106
Casale, M., 305
Catmull–Clark surfaces, 383, 392
Catmull, E., 383
Catmull–Rom spline, 173
centroid, 15, 18
Ceva's theorem, 36
Chaikin, G., 132
Chang, G., 74
Chang, S., 116, 305
Charrot, P., 330, 367, 417
Chebychev polynomials, 399
Chiyokura, H., 410
chord length, 180
  parametrization, 170
Chou, J., 238
Chrysler, 399
circle, 2, 5, 159, 221, 239, 299
  osculating, 184
circulant matrix, 383, 386
clamped end condition, 155
Clark, J., 383
clay model, 6
closed mesh, 395
Clough-Tocher interpolant, 343
Cobb, J., 339
Cohen, E., 68, 132, 163
Cohn-Vossen, S., 425
collinear, 25
compatibility, 408
complementary segment of a conic, 220
computational fluid dynamics, 399
computation count, 268
conic precision, 240
conic section, 44, 205, 209, 337, 359, 364
conjugate direction, 361
connection matrix, 202
CONS, 304
control net, 248
control points, 45
  infinite, 224
control polygon, 45
  of a conic, 211
control vectors, 238
conversion of B-spline surfaces, 288
convex, 41, 85
convex combination, 16, 273
convex hull, 16, 37, 41, 48

convex hull property, 48, 60, 98, 224, 335
of rational Bézier curves, 228
of Bézier patches, 254
of Bezier triangles, 312
convexity, 270
convexity of surfaces, 264
convex set, 16
Coons patch
discrete, 272
Coons patch, bilinearly blended, 291, 402
Coons, S., 2, 205, 227, 399
coordinate-free, 13
coordinate system, 13
corner, 170
corner-cutting, 132
corner of a curve, 162
corner twist, 290
correction surface, 413
Coxeter, H., 15
Cox, M., 119, 142
Cramer's rule, 34
cross-boundary derivative, 257, 319, 403
cross-boundary normal derivative, 344
cross plot, 87
cross ratio, 206, 213, 219, 231
cross ratio theorem, 206
Crouch, P., 168
cubic Hermite interpolant, 416
cubic Hermite interpolation, 404, 413, 415
cubic Hermite polynomial, 104, 199, 403
cubic precision, 176, 240
curvature, 167, 183, 419
Gaussian, 270
curvature,
derivative of, 188, 421
of nonparametric curve, 187
of rational Bézier curve, 185
signed, 235, 243
curvature plot, 159, 163, 419
curve on surface, 304
cusp, 166
cylinder, 300
Dahmen. W., 68
Daniel, M, 432
data transfer, 83
Daubisse, J., 432
Davis, P., 22, 57, 90
de Boor algorithm, 123
rational, 242
for rational B-splines, 241
de Boor, C., 22, 119, 123, 132, 142, 235, 285, 309
de Casteljau, P., 2, 30, 43, 50, 68, 119, 245, 309
de Casteljau algorithm, 45, 68, 247
degenerate patch, 261
Degen, W., 202
degree elevation, 238, 326
for B-splines, 144
for Bernstein polynomials, 92
for Bézier curves, 81, 132
for Bézier patches, 254
for rational Bézier curves, 236
degree reduction, 85
for B-spline curves, 133
for Bézier triangles, 333
Delaunay triangulation, 39
depth
of an S-patch, 330
de Rham G., 132
derivative
of a B-spline, 144
of a conic, 215
of a rational Bézier curve, 233
DeRose, T., 17, 50, 202, 260, 301, 330, 367, 390, 434
design, 49
developable surface, 270, 358
diagonalization, 378
difference operator, 66, 256
Dill, J., 420, 425
dimension
of B-spline spaces, 140
directional derivative, 316
directrix, 9
Dirichlet tessellation, 37
discrete Coons patch, 272
discriminant, 351
divided differences, 119

do Carmo, M., 202
domain, 68, 246
Doo, D., 380
Doo-Sabin surfaces, 380, 392
duck, 167
Dupin's indicatrix, 359, 364, 376
Dyn, N., 41, 202, 379, 391

ellipse, 220, 359
elliptic point, 357
end condition
  Bessel, 158, 306
  clamped, 155
  natural, 157
  not-a-knot, 158
endpoint interpolation, 61
  of Bézier curves, 49
Epstein, M., 162
euclidean map, 19
euclidean space, 14
Euler, L., 17, 179, 359
Euler-Poincaré formula, 395
extraordinary vertex, 381, 385, 388
extrapolation, 68

fairness of a curve, 420
Farin, G., 84, 202, 309, 326, 327, 422
Farouki, R., 222, 305, 425, 432
Faux, I., 74
Ferguson, J., 2, 285, 290
fine tune, 197
Fiorot, J., 227
first fundamental form, 350
flat point, 357
flat spot, 404
Floater, M., 233, 242
FMILL, 172
Foley, T., 162, 163
font design, 19, 71
Ford, 399
Ford, H., 12
Forrest, A. R., 64, 84, 205, 213, 227, 420, 425
Forsey, D., 377
forward difference, 64
forward differencing, 114
four-point scheme, 379
four tangent theorem, 213
fractals, 79
Frederickson, L., 309
Frenet frame, 181
Frenet frame continuity, 202
Frenet-Serret formulas, 183
functional curve, 86

Gallier, J., 119
Garland, M, 396
Gauss, C., 179, 357
Gaussian curvature, 270, 357, 425
Gauss-Seidel iteration, 157
Geise, G., 195, 202
generalized Vandermonde matrix, 101
General Motors, 399
generatrix, 300, 361
genus, 395
geodesic, 363
geometric continuity, 189, 191, 222
geometric programming, 17
geometry matrix, 265
Goldman, R., 13, 17, 50, 144, 189, 202, 314
Gonska, H., 57
Goodman, T., 202, 234
Gordon surface, 410
  spline-blended, 412
Gordon, W., 2, 399, 410, 414
gradient, of implicit conic, 217
Grassmann, H., 41
Gregory, J., 202, 330, 367, 379, 391, 408, 417
Gregory's square, 409
Greville abscissa, 135, 148

Hagen, H., 202, 293
Hahn, J., 367
Hansford, D., 40, 340
harmonic function, 8
Hartley, P., 163
hat function, 23
Heckbert, P., 396

Hermite form, 435
Hermite interpolation, 102, 375, 413
Hermite patch, 276
Herron, G., 367, 376
hierarchical B-spline surface, 377
Hilbert, D., 425
Hinds, J., 305, 425
hodograph, 64
Höllig, K., 119, 235
Holman, T., 434
homogeneous coordinates, 210
Horner's scheme, 75
Hoschek, J., 153, 367, 424
hyperbola, 220, 247, 359
hyperbolic paraboloid, 246
hyperbolic point, 357

idempotent operator, 414
identity map, 18
IGES, 143, 217
image plane, 206
image registration, 22
implicit form
  of conic, 216
  of quadric, 340
  of surface, 349
  tolerances, 225
improper parametrization, 237
incenter, 345
infinite control point, 224, 238
inflection point, 429
integrals, 88
integrals of tensor product patches, 267
interference checking, 49
interior vertex, 394
interpolating
  conic, 218
  polynomial, 95
  quadric, 340
intrinsic geometry, 357
invariance under affine parameter transformations, 435
  of Bézier curves, 48
isoparametric curve, 246, 252, 410
isophotes, 425
isotropic directions, 351

Jeannin, P., 227
Jensen, T., 367
Jolles, S., 74
Jones, A., 367, 427
Judd, C., 163
junction points, 170

Kahmann, J., 367, 376
Kaufmann, E., 427
Kiciak, P., 367
Kimura, F., 410
Kjellander, J., 424
Klass, R., 235, 425, 427
Klein bottle, 308
Klucewicz, I., 291
knot insertion for rational B-splines, 242
knot multiplicities, in IGES, 143
Kobbelt, L., 389
Korovkin, P., 57, 90, 91
Kronecker Delta, 61
Kruth, J., 240

Lacombe, C., 417
Lagrange interpolation, 99
Lagrange polynomial, 98, 411
Lane, J., 68
Laplacian smoothing, 281
Lasser, D., 305
least squares, 108, 149, 278, 346
  tensor product, 278
Lee, E., 50, 162, 220, 242
Leibniz formula, 33
Levin, D., 41, 391
Levin, S., 379
Levoy, M., 394
Lien, S., 116
Liming, R., 205, 219
linear independence of B-splines, 141
linear interpolation, 25
  bivariate, 36
linear interpolation, repeated, 44
linearly independent, 22
linear operator, 23, 45
linear parameter transformation, 48

linear precision, 62, 98, 171
  of Bézier triangles, 327
  of rational curves, 228
linear space, 14, 22
line of curvature, 356
  of a surface of revolution, 356
Little, F., 40
Liu, D., 367, 420
Lobachevsky, N., 119
local control, 129
local coordinates, 48
local parameter, 28
local support, 140
lofted surface, 269, 411
loop, 170
Loop, C., 330, 387
Lorentz, L., 57
Lounsbery, M., 390
Lucian, M., 242
Lyche, T., 132

main normal vector, 181
Manning, J., 195, 202
Mann, S., 260, 390
Mansfield, L., 119, 142
Marhsall, J., 417
mass point, 41
matrix circulant, 383, 386
matrix form of curves, 74
Ma, W., 240
maxmin criterion, 40
McConalogue, D., 163, 174
mean curvature, 357, 425
Meier, J., 57
Memke's theorem, 189, 194
Menelaos' theorem, 30
meridian, 299
Meusnier's theorem, 354
Micchelli, C., 132, 189, 202
Michelangelo, 394
  digital, 394
Miller, J., 305
minimal support of B-splines, 140
minmax box, 49, 69
minmax criterion, 40
Mitchell, A., 417
mixed partials, 257
Moebius, F., 17, 26, 34, 206
Monge, C., 179
monomial basis, 75, 432
monomial form, 67
  of Bézier patches, 265
monomials, 23, 99
Mortenson, M., 74
multiaffine, 31, 53
multiaffine function, 313
multiplicities, of end knots, 143
multiplicity
  of a knot, 127
multiresolution, 397
Munchmeyer, F., 425
Musse, J., 367

Nachman, L., 403
NAPOLEON, 11
Nasri, A., 390
natural end condition, 157, 162
nested multiplication, 75
neutral set, 39
Neville, 97
Newton form, 114
Newton iteration, 153
Nielson, G., 107, 163, 194, 199, 367, 414, 416, 417
nine parameter interpolant, 341
nonparametric curve, 187
nonparametric patch, 266
normal curvature, 354
normal equations, 86, 150, 347
normal section, 354
normal vector, 261
numerical control, 1
NURBS, 227

O'Dell, C., 163
open mesh, 395
order of contact, 202
orthonormal matrix, 19
osculant, of tensor product surface, 259
osculating circle, 184
osculating plane, 73, 182, 203

osculatory interpolation, 234
Oslo algorithm, 132
Overhauser spline, 173, 177

parabola, 2, 5, 43, 44, 173, 185, 209, 220, 227, 240, 247, 339
parabolic cylinder, 339
parabolic point, 357
paraboloid, 339, 359
parallel projection, 19, 20
parameter correction, 153
parametrization, 170, 179, 222, 339
  centripetal, 150, 162
  chord length, 162
  equidistant, 161
  for tensor product interpolation, 295
  uniform, 161
partial derivatives, 256
partition, 23
  of unity, 141
Pascal's theorem, 218
Patterson, R., 205, 236
Pegna, J., 364, 376
Penna, M., 205
Petersen, C., 327, 333
Peters, G., 285
Peters, J., 386, 392
piecewise Bézier curve, 137
piecewise bilinear interpolation, 255
piecewise linear interpolation, 82, 84
Piegl, L., 224, 227, 339
Piper, B., 107, 367, 375
pixel, 116
Poeschl, T., 425
point, 14
  at infinity, 224
polar, 73
  of tensor product surface, 259
polar form, 351
polygon, 29
polynomial space, 22
Pottmann, H., 202, 421
Powell, M., 345
power basis, 432
Pratt, M., 74
Pratt, V., 116
Prautzsch, H., 85, 119, 132, 144
principal curvatures, 357
principal direction, 356
projective geometry, 205
projective invariance, 48
projective map, 205, 208
projector, 414

quadratic precision, 342
quadric, 337, 396
quintic Hermite Interpolation, 106, 177

radial line, 322
radius of curvature, 184
Rajan, V., 432
Ramshaw, L., 50, 68, 74, 119, 247, 336
range, 246
rank, 20
ratio, 24, 27, 206, 286
rational linear parameter transformation, 213
rational linear transformation, 208
rational quadratic form for conics, 211
rational quadratics, 337
recursive subdivision, 380
reflection lines, 425
regular parametrization, 180
Reif, U., 377, 386
Renyi, A., 119
Renz, W., 424
reparametrization, 87, 180, 191, 201, 299, 311, 349, 360
  of rational Bézier curves, 235
  of rational B-splines, 242
  of surfaces, 376
Riesenfeld, R., 2, 68, 132
rigid body motion, 19
Rippa, S., 41
Ris, G., 367
Rochetti, R., 116
rotation, 18
rough sketch, 197
roundoff, 435
ruled surface, 246, 269, 361
ruling, 270
Runge phenomenon, 102

S. Parry, 301
Sabin, M., 235, 309, 345, 380, 389, 420
Sablonniére, P., 309, 327
Sakkalis, T., 222
Sapidis, N., 419
Sarraga, R., 367, 375
scaling, 18
scattered data interpolation, 341
Schelske, J., 264
Schneider, F., 240
Schoenberg, I., 84, 119
Schoenberg-Whitney theorem, 150
Schulze, G., 293
Schumaker, L., 68, 327
second fundamental form, 353
Sederberg, T., 49, 64, 237, 260, 301
Seidel, H., 50, 314
Selesnick, S., 293
self-conjugate, 361
semicircle, 224
Séquin, C., 367
Shantz, M., 116, 305
shape preservation, 101, 270
shear, 19, 24
Shirman, L., 367
shoulder point, 213, 219
shoulder tangent, 219
signed curvature, 187
simplest scheme, 386
slanted font, 19
smoothness
  of piecewise Bézier curves, 137
solid modeling, 338
sphere, 300
spline, 2, 167, 173, 240, 421, 424
spline interpolation, 412
  iterative, 157
stability of the monomial form, 432
Staerk, E., 68
Stancu, D., 309
standard cubic, 55
standard form
  for conics, 213
  of rational Bézier curve, 236, 299
star, 396
Steiner, J., 213
stochastic matrix, 385
storage for B-spline curves, 435
straight line, 25
strain energy, 167
Su, B., 420
subdivision, 68
  formula, 68
  of Bézier curves, 68
  of Bézier patches, 259
  of Bezier triangles, 321
  of rational Bézier curves, 232
  in terms of Bernstein polynomials, 92
subspace, 22
surface, 9
  butterfly, 391
  conical, 270
  cylindrical, 270
  developable, 270
  lofted, 269
  ruled, 269
  translational, 272
surface of revolution, 299
Swimmer, A., 41
symmetric function, 313
symmetry, 61

tangent length, 435
tangent plane of Bézier triangle, 318
tangent ribbon, 404
tangent vector, 181
Taylor series, 67
tension, 195
tensor product, 246, 412
  interpolant, 405, 435
  surfaces, 245
tetrahedron, 333
texture mapping, 425
theorema egregium, 357
Thiessen region, 37
three tangent theorem, 44
Tiller, W., 227, 339
topology of triangulations, 394
torsion, 183
  continuity, 202
  of nonparametric curve, 187
  of rational Bézier curve, 185, 186, 234

torus, 300
transfinite interpolation, 399, 410
translation, 14, 18
translational surface, 264, 272, 282, 290
transposant, 304
triangle
  area of, 35
  numbers, 310
triangulation, 39
  data dependent, 41
trimmed surfaces, 304
trinomial coefficient, 315
trivariate B-splines, 302
trivial reject, 69
twist estimation, 406
twist incompatibility, 417
twist, variable, 409
twist vector, 257, 264
type check, 17

umbilical point, 355
uniform parametrization, 222
UNISURF, 1
unit interval, 28
unit sphere, 339
unit square, 246

valency, 381
Vandermonde, 100, 275
van Wijk, J., 367
variable twists, 409
variation diminishing property, 29, 41, 84, 98, 202
  for Bézier patches, 254
  of rational Bézier curves, 228
vectors, 14
Vernet, P., 89
Veron, M., 367
Versprille, K., 224, 238
vertex split, 397
Vigo, M., 305
Vinacua, A., 367
volume distortion, 301
Voronoi diagram, 37

Wang, X., 64
Weierstrass approximation theorem, 90
weight point, 214, 219, 230, 231, 235, 336
weights, 211
  of rational Bézier curves, 227
Wolter, F., 364, 376
Worsey, A., 235

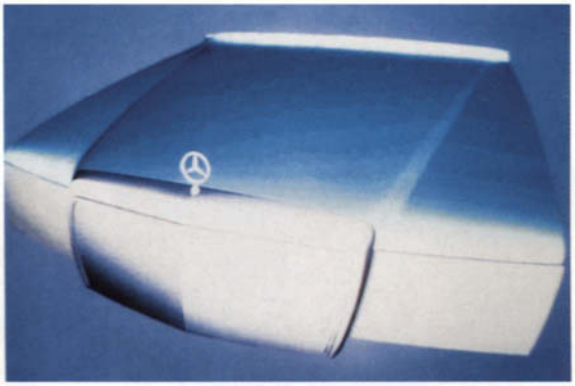
Yamaguchi, F., 157

zero twist, 273, 290, 404
Zhou, J., 84

This Page Intentionally Left Blank

Plate I.
An automobile.
Figure courtesy of
Mercedes-Benz, FRG.

Plate II.
Color rendering of the
hood. Figure courtesy of
Mercedes-Benz, FRG.



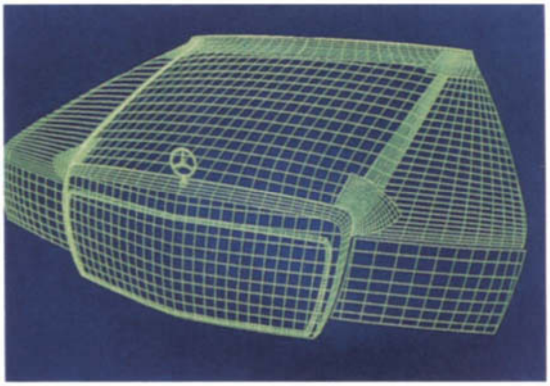


Plate III.
Wire frame rendering of
the hood. Figure courtesy
of Mercedes-Benz, FRG.

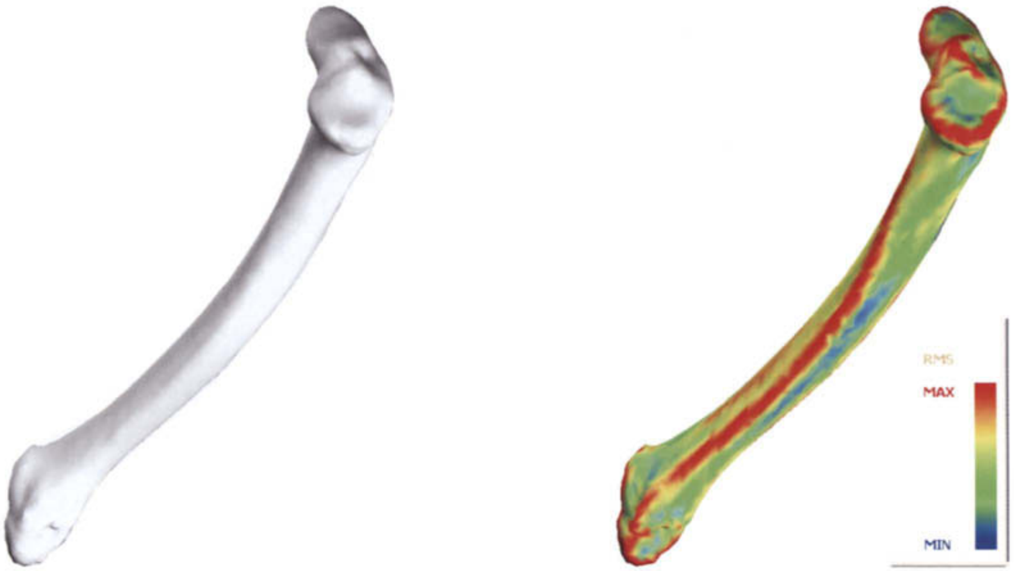
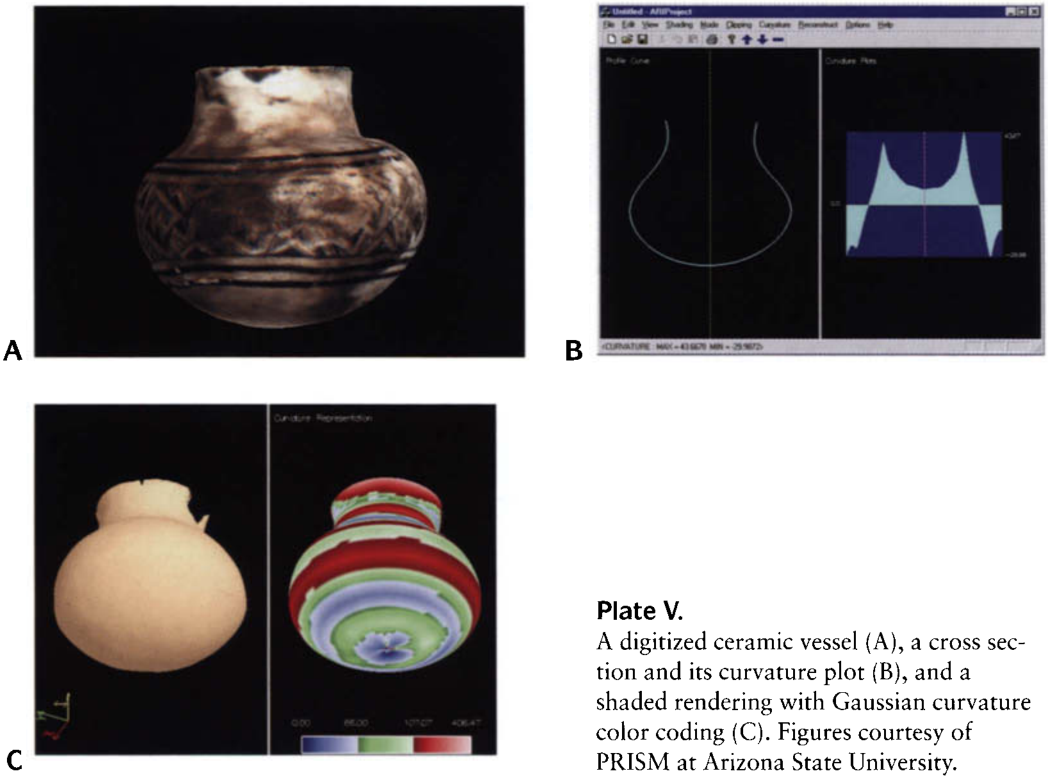


Plate IV.
A digitized bone (left) and absolute curvature color coding (right). Figure courtesy of PRISM at Arizona State University.



<CURVATURE : MAX = 43.6670 MIN = -25.9872>

Plate V.
A digitized ceramic vessel (A), a cross section and its curvature plot (B), and a shaded rendering with Gaussian curvature color coding (C). Figures courtesy of PRISM at Arizona State University.