Introduction to Curves and Surfaces

SIGGRAPH 1996

Dr. Alyn P. Rockwood
Organizer

Peter Chambers
Author & Presenter

Dr. Hans Hagen
Presenter

Thomas McInerney
Presenter

Abstract

This course presents an introduction to CAGD—Computer-Aided Geometric Design. The mathematical tools required to create well-behaved curves and surfaces are covered, together with efficient algorithms for their implementation. Topics covered include basis functions, Bézier curves, B-splines, and surface patches.

Table of Contents

Course Contributors

Dr. Alyn P. Rockwood

Alyn P. Rockwood completed B.S. and M.S. degrees in mathematics at Brigham Young University and a Ph.D. at the Dept. of Applied Mathematics and Theoretical Physics of Cambridge University, Cambridge, England. He worked in industrial research for 13 years, including supervisory and research positions at Evans and Sutherland Computer Corp. and Silicon Graphics, Inc. He was involved in flight simulation, CAD/CAM and surface rendering projects at these companies.

Currently, he is on the computer science faculty at Arizona State University. His interests include computer graphics, scientific visualization and computer aided geometric design. He has several patents and many publications in these fields.

Peter Chambers

Peter Chambers received his B.Sc. from the University of Exeter, England, and his M.S. from Arizona State University. Peter is an Engineering Fellow in the Advanced Multimedia Group at VLSI Technology, Tempe, Arizona. His areas of interest at VLSI include circuit design, computer architectures, and peripheral interfaces for personal computers.

Peter has architected and designed numerous products, including the Input/Output systems for minicomputers, and many large integrated circuits. Peter's other interests include high level hardware description languages, techniques for robust design, and interface performance analysis.

Peter's involvement with Arizona State University includes research on texturing methods in computer graphics and interactive learning tools. His most recent publication is Interactive Curves and Surfaces, in collaboration with Alyn Rockwood, upon which these course notes are based.

Dr. Hans Hagen

Hans Hagen is a professor of computer science at the University of Kaiserslautern, Germany. His research centers on geometric modeling and scientific visualization.

Dr. Hagen received his B.A. and his M.S. from the University of Freiburg and his Ph.D. from the University of Dortmund. Before moving to Kaiserslautern he held faculty positions at the University of Braunschweig and at Arizona State University. He is one of the current directors at the Dagstuhl Conference and Research Center.

Dr. Hagen is on the editorial board of Computer Aided Geometric Design, Transactions on Visualization and Computer Graphics, Computer Aided Design, Computing, and Surveys on Mathematics for Industry.

Thomas McInerney

Thomas McInerney writes code for Apple Computer during his summers. His interests are in computer graphics and computer networks. He is completing his B.S. in Computer Science from Arizona State University. He is also writing the Java version of the curves and surfaces book with Alyn Rockwood and Peter Chambers.

Contact Information

Alyn Rockwood

Dr. Alyn P. Rockwood
Department of Computer Science
Arizona State University
Tempe
Arizona 85287

Phone: 602 965 8267

email: rockwood@asu.edu

Peter Chambers

Peter Chambers
VLSI Technology
8375 South River Parkway
Tempe
Arizona 85284

Phone: 602 752 6395

email: peter.chambers@vlsi.com

Hans Hagen

Prof. Dr. Hans Hagen
University of Kaiserslautern
FB Informatik / AG Hagen 36/226
Postfach 3049
67663 Kaiserslautern
Germany

Phone: 49 631 205 4071

email: hagen@informatik.uni-kl.de

Thomas G. McInerney

Thomas G. McInerney
2929 N. 70th St. # 2089
Scottsdale
Arizona 85251

Phone: 602 970 7643

email: tgm@asu.edu

Acknowledgments

Alyn Rockwood and Peter Chambers wish to thank:

Mike Morgan and Marilyn Uffner Alan at Morgan Kaufman, for making the *Interactive Curves and Surfaces* project possible.

Spencer Thomas, Jim Miller, Aristides Requicha, and Jules Bloomenthal, for their excellent comments and ideas.

Shara-Dawn Simon, for her diligent proof-reading, and the significant improvements in layout and flow she suggested.

Professor Don Evans, director of the Center of Innovation in Engineering Education, of the College of Engineering and Applied Science, Arizona State University, for support and encouragement.

The authors also appreciate the sponsorship of the US/Hungarian Science and Technology Joint Fund, in cooperation with Arizona State University, project 396.

Topic 1

Introduction to CAGD

Topic 1: Introduction to CAGD

In this topic, you will:

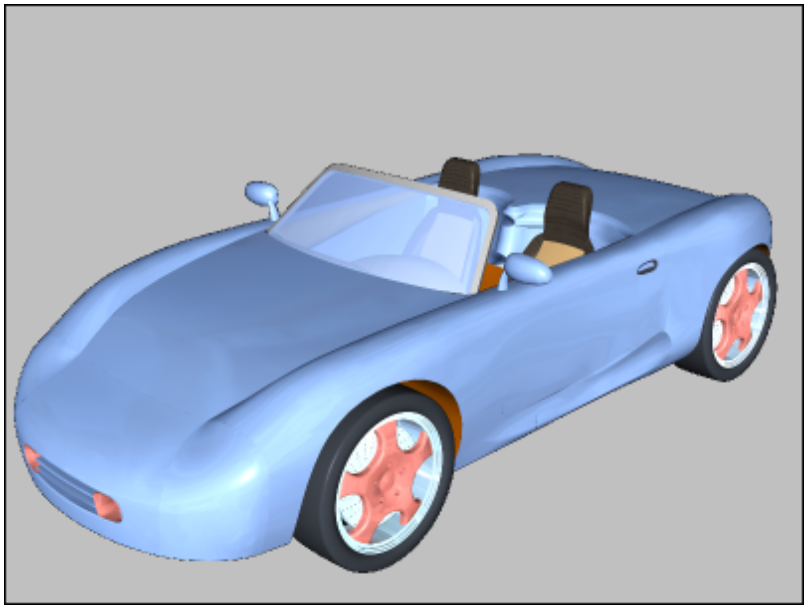
- Learn what CAGD is about.
- Learn some of the history of CAGD.
- See some typical applications of CAGD tools.

What CAGD is All About

Computer-Aided Geometric Design is a new field that initially developed to bring the advantages of computers to industries such as:

- Automotive
- Aerospace
- Shipbuilding

CAGD expanded rapidly and now pervades many areas, from pharmaceutical design to animation. We are surrounded by products that are first visualized on a computer. These products are modified and refined entirely within the computer; when the product enters production, the tools and dies are produced directly from the geometry stored in the computer. This process is known as virtual prototyping.



(c) 1995 Autodesk, Inc. This image was provided courtesy of Autodesk, Inc.

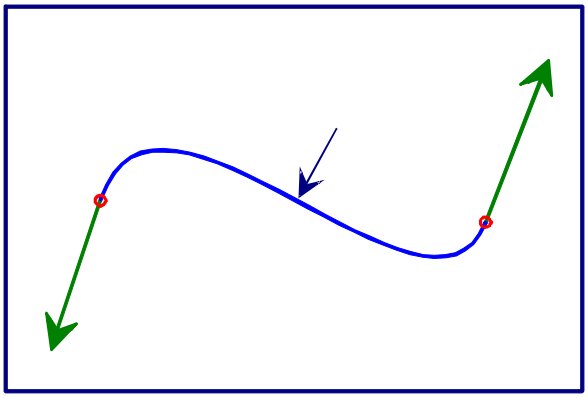
Computer visualizations of new products reduce the design cycle by easing the process of design modification and tool production.

CAGD is based on the creation of curves and surfaces, and is accurately described as curve and surface modeling. Using CAGD tools with elaborate user interfaces, designers create and refine their ideas to produce complex results. They combine large numbers of curve and surface segments to realize their ideas. However, the individual segments they use are relatively simple, and it is at this level that the study of CAGD is concentrated.

A Design Challenge: The Need for CAGD

When creating products and artwork, designers face tasks such as this:

You are given two points in a plane and two directions associated with the points. Find the curve that passes through the points that is tangent to the given directions:



This is a simple task for anyone with a pencil and paper who is familiar with parametric forms, the types of curves used in CAGD.

CAGD produces tools are meant to be:

- Intuitive
- Simple to use

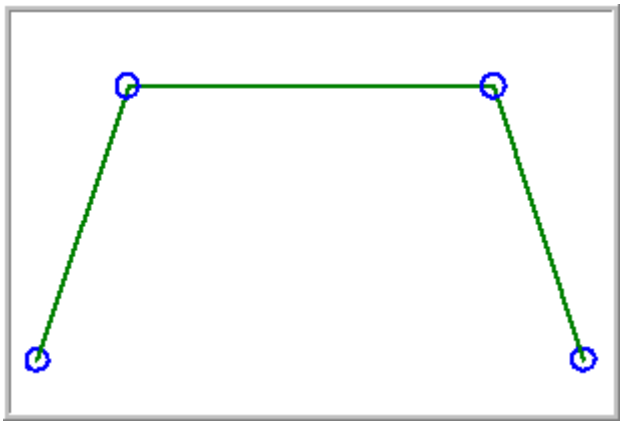
Sometimes the mathematics underlying the tools becomes quite sophisticated, yet the result is meant to be easily understood and geometrically intuitive.

The technical person often benefits from these intuitive and visually related tools when considering deeper mathematical problems. The geometry of CAGD is very amenable to visual demonstration.

CAGD: From Points to Teapots

There is a natural progression of the geometry behind CAGD. With small incremental steps it is possible to describe complex objects in terms of simple primitives, such as points and lines.

- Control Points: The Start of CAGD

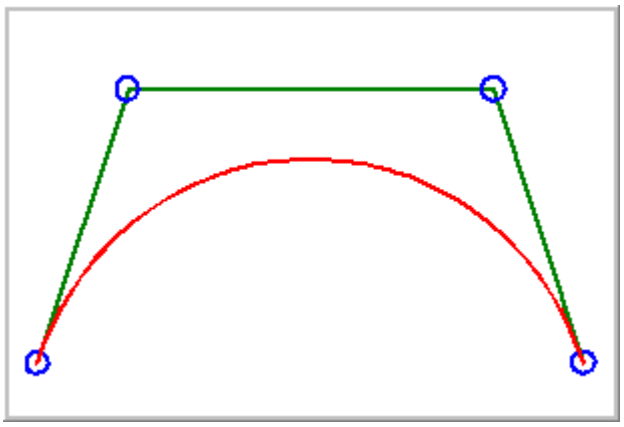


Take four points in a plane and connecting them to form a polygon. The points are known as Control Points [1], and the polygon as the Control Polygon [2]. The control points and polygon determine the approximate shape of the curve to be formed.

[1] Control points are points in two or more dimensions which define the behavior of the resulting curve.

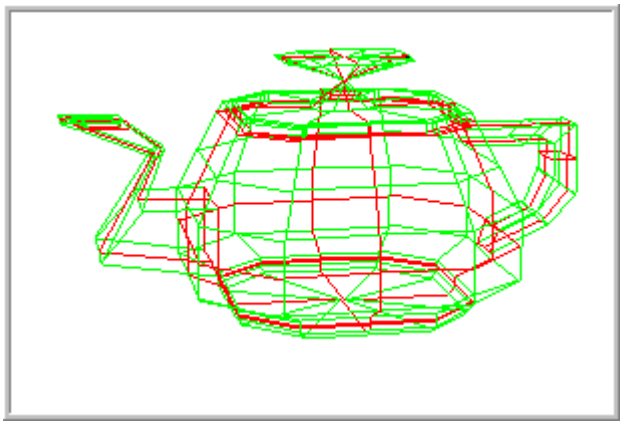
[2] The control polygon is formed by connecting the control points in the correct order. The control polygon provides a crude analogy of the refined curve. Note that the control polygon is typically open (the ends are not coincident), and it may self-intersect arbitrarily.

- A CAGD Favorite: The Bézier Curve



A special curve known as the Bézier Curve [3] may be generated by the control points. Note that while the curve passes through the end points, it only comes close to the other points.

- Three-Dimensional Control Polygons for Surfaces (Control Meshes)

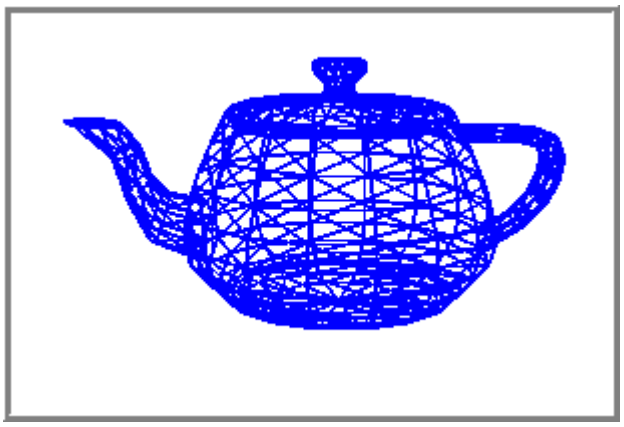


Bézier curves behave just as well in three dimensions as in two. This figure shows the control polygons for a three-dimensional object consisting of Bézier patches[4].

[3] The Bézier curve, named after the French researcher Pierre Bézier, is a simple and useful CAGD curve. It is a very well behaved curve with useful properties, as you will discover in Topic 3, The Bézier Curve.

[4] A Bézier patch is a three-dimensional extension of a Bézier curve. It is formed by extruding a Bézier curve through space to form a surface.

- Three-Dimensional Wireframe Model



When the Bézier curves are created and connected in three dimensions, a wireframe model of the object is produced.

- Three-Dimensional Shaded Object

If the surface produced by the three-dimensional Bézier patches is illuminated and shaded, an object with a realistic appearance results. The Utah teapot, on display at the Boston Museum of Computer History, is a classic in CAGD and computer graphics.

The History of CAGD

Computer Aided Geometric Design has mathematical roots that stretch back to Euclid and Descartes. Its practical application began with automated machinery to compute, draft, and manufacture objects with freeform surfaces. Production pressures in the aircraft industry during World War II stimulated many new devices to enhance and accelerate design and manufacturing. For example, in 1944 Liming designed fuselage spars with a "super-elliptic" method that could be implemented with an electro-mechanical calculator.

Shipbuilders also became interested in CAGD early on. One example of their motivation may sound whimsical but was a serious impediment to ship design. The only place large enough to draw full scale plans for a ship was in the loft of the shipbuilders dry-dock. The huge drawings would warp and shrink in the moist air, causing very real manufacturing problems.

Computers provided the greatest stimulus because of their power to enable new ideas. In 1963, Ferguson developed one of the first surface patch systems by which individual curvilinear patches are joined smoothly to create the surface "quilt." He also introduced the notion of parametrically defined surfaces which has become the standard because it provides freedom from an arbitrarily fixed coordinate system. Vertical tangent vectors can be defined by differentiation, for instance, which is not possible in explicit Cartesian form.

In the mid 1960's automotive companies became involved in CAGD, as a way to drive milling machines. Car bodies were designed by artists using clay models. Painstaking measurements produced data that could drive numerically controlled milling machines to produce stamp molds. The initial use of CAGD was to represent the data as a smooth surface for numerical control. It soon became apparent that the surfaces could be used for the design.

In 1971, Pierre Bézier reformulated Ferguson's ideas so that a draftsman without any extensive mathematical training could design a surface. Bezier's system, UNISURF, was used by Renault and became a milestone in the development of CAGD. It epitomized the difference between surface fitting and surface design. The purpose of design was to provide the draftsman, who had strong intuition about shape but limited mathematical training, computer tools that empowered him or her to use the sophisticated mathematics of surface representation.

In the meantime, the mathematical underpinnings of CAGD continued to advance. de Casteljau examined triangular patches and developed evaluation techniques. Coons [Coons64] unified much of the previous work into a general scheme which became the basis of the early modeler PDGS made by Ford. At General Motors in 1974, Gordon and Riesenfeld exploited the properties of B-spline curves and surfaces for design.

Driven primarily by the automotive, shipbuilding and aerospace industries, both the mathematics of CAGD and the designer interface tools continued to

improve through the 1970's. The first CAGD conference was organized by Barnhill and Riesenfeld in 1974, where the term "CAGD" was first used [Barnhill74].

In the 1980's, the power and versatility of computer-aided designing seemed suddenly to be discovered by anyone who had a freeform geometric surface application. Industrial designers were smitten with the power of computer design, and many commercial modelers become the basis of several substantial applications, including: CATIA, EUCLID, STRIM, ANVIL, and GEOMOD. Geoscience used CAGD methods to represent seismic horizons; computer graphics designers modeled their objects with surfaces, as did molecule designers for pharmaceuticals. Architects discovered them, word processing and drafting programs based their interface protocols on freeform curves (PostScript[5]), and even moviemakers discovered the power of animating with such surfaces, beginning with TRON, continuing through Jurassic Park[6], and beyond.

What has Been Accomplished in this Topic

The ideas and principles behind CAGD have been introduced, together with the classic example of surface patch use, the Utah teapot. The short but significant history of CAGD has been summarized.

---

[5] PostScript is a proprietary page-description language used by typesetters to define elements of printed text, including letter outlines, text layout, and graphical images.

[6] Jurassic Park made extensive use of CAGD and computer graphics to visualize animated objects.

Topic 2

Preliminary Mathematics

Topic 2: Preliminary Mathematics

In this topic, you will learn:

- The basic math needed for the rest of the book.
- An overview of parametric forms: a convenient way of describing curves and surfaces.
- An illustration of linear interpolation, one of the most fundamental concepts in CAGD.
- The idea of continuity, to ensure that curves and surfaces join together smoothly.

The Mathematics of CAGD

CAGD treats points, lines, and surfaces as mathematical objects, described geometrically in two- or three-dimensional space.

This book develops the mathematics in a step-by-step fashion, and does not demand a rigorous mathematical background. It is recommended that the reader be familiar with the following topics:

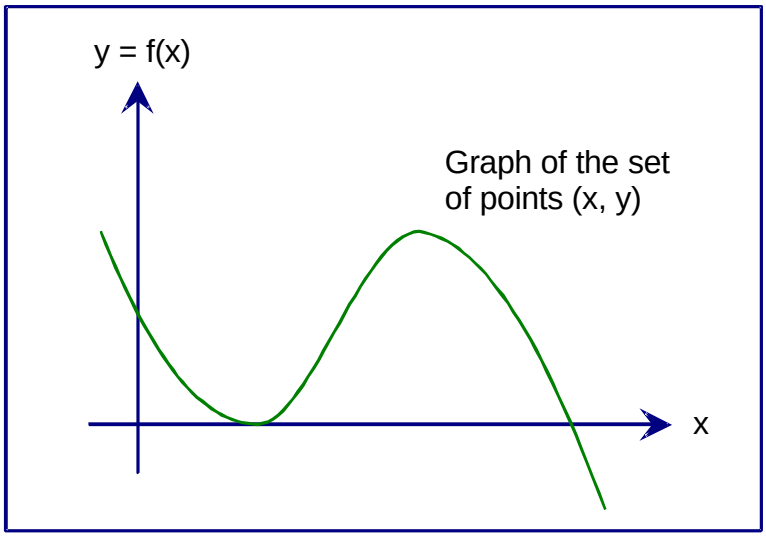
- Geometry of points, lines, and planes.
- Equations in parametric form.
- Basic calculus.

Parametric Forms

CAGD relies on parametric forms to describe curves and surfaces. Many students of CAGD do not initially appreciate the subtlety and importance of this form. If you are confident with parametric forms then you may skip this section.

The Parametric Curve

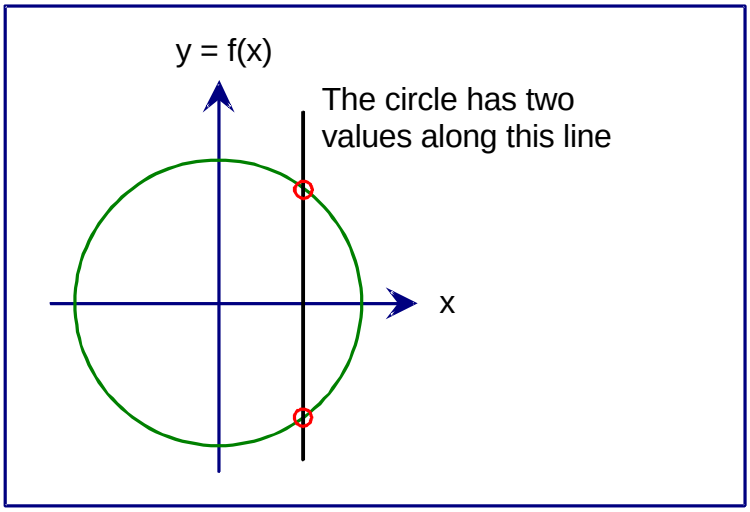
Typically, when a student takes mathematics, a curve is presented as a graph of a function f(x).



As x is varied, y = f(x) is computed by the function f, and the pair of coordinates (x, y) sweeps out the curve. This is called the *explicit* form of the curve.

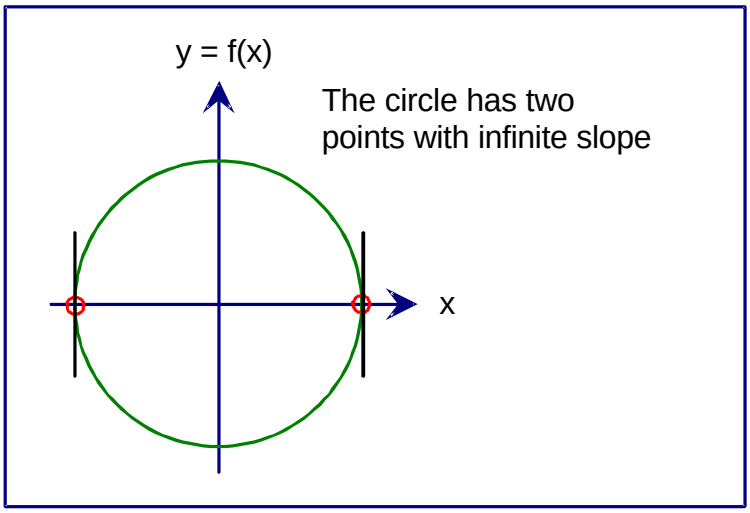
From a design standpoint the explicit form is deficient in several ways:

- Single-Valued



The curve is single-valued along any line parallel to the y axis. For example, only parts of the circle may be defined explicitly.

- Infinite Slope



An explicit curve cannot have infinite slope; the derivative f' (x) is not defined parallel to the y axis. Hence there are two points on the circle that cannot be defined.

- Transformation Problems

Any transformation, such as rotation or shear, may cause an explicit curve to violate the two points above.

The parametric form of a curve is not subject to these limitations. Moreover, it provides a method, known as parameterization [1], that defines motion on the curve. Motion on the curve refers to the way that the point (x, y) traces out the curve.

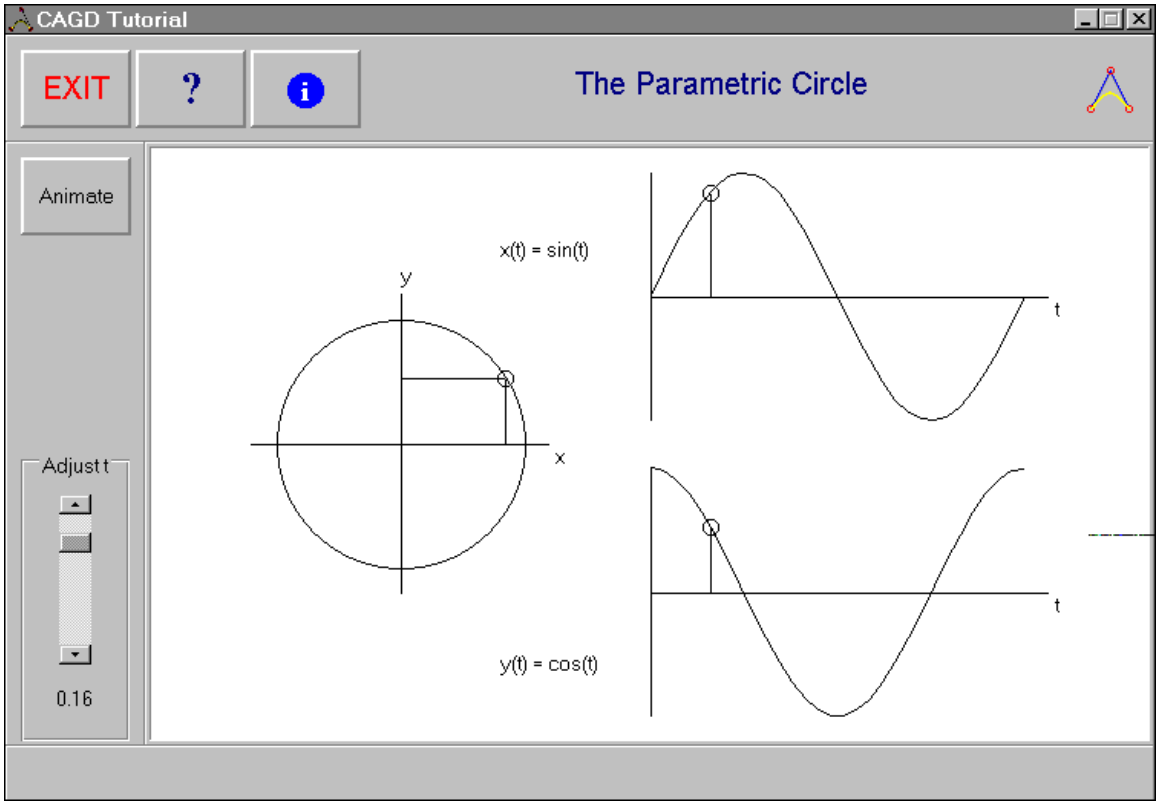
Defining the Parametric Curve

A parametric curve that lies in a plane is defined by two functions, x(t) and y(t), which use the independent parameter t. x(t) and y(t) are coordinate functions, since their values represent the coordinates of points on the curve. As t varies, the coordinates (x(t), y(t)) sweep out the curve. As an example consider the two functions:

$$x(t) = \sin(t),\ y(t) = \cos(t). \qquad (2.1)$$

As t varies from zero to 2π, a circle is swept out by $(x(t),\ y(t))$.

- The Parametric Circle: t = 0.16 (approximately 57 degr ees)



[1] Parameterization uses an independent parameter or variable to compute points on the curve. It gives the "motion" of a point on the curve.

CAGD deals primarily with polynomial [2] or rational functions [3], not trigonometric functions as shown in the examples above. For example, the circle can also be given by allowing t to vary from -infinity to +infinity in the following functions:

$$x(t) = \frac{2t}{(1+t^2)}, \; y(t) = \frac{(1-t^2)}{(1+t^2)}. \qquad (2.2)$$

As an exercise, verify for yourself that the functions in equation 2.2 do indeed generate a circle. Plot points (x(t), y(t)) or write a program to do this for you.

Both equation 2.1 and equation 2.2 yield circles, so how do they differ? It is the parameterization. The motion of the point (x(t), y(t)) is different, even if the paths (the circles) are the same.

A good physical model for parametric curves is that of a moving particle [4]. The parameter t represents time. At any time t the position of the particle is (x(t), y(t)). Two paths (curves) may be identical even though the motion (parameterization) is different.

Parametric curves are not constrained to be single-valued along any line (recall the single-valued deficiency of the explicit form), and the slope of a parametric curve segment may be defined vertically. The slope is given by the tangent line at any point, computed by finding the derivative vector (x'(t), y'(t)) at any point t. This vector determines the speed at which the point traces out the curve as t changes.

Curves defined by points whose speed may drop to zero do cause problems that will be considered later under the discussion of continuity.

---

[2] A polynomial is a function of the form:

$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n$, where the a_i are scalars or vectors.

[3] A rational function is made by dividing one polynomial by another, for example:

$$r(t) = \frac{1 - 2t + 3t^3 + t^4}{1 - 2t^2}.$$

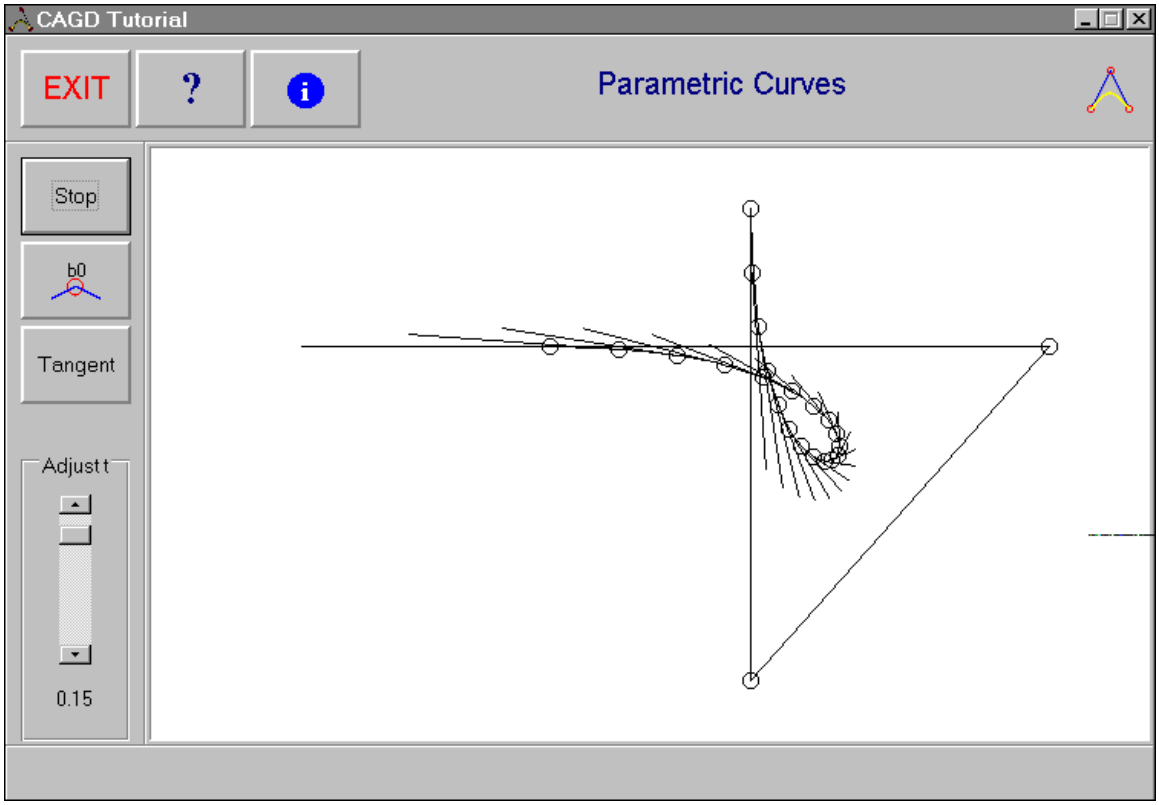
A rational function may contain vector coefficients only within the numerator.

[4] As the parameter t changes, the coordinate point (x(t), y(t)) traces the curve. This point can be thought of as a particle that moves under the influence of changes in the value of the parameter t.

Consider the parametric curve given by these two coordinate functions:

$$x(t) = 6t - 9t^2 + 4t^3,$$
$$y(t) = 4t^3 - 3t^2. \qquad (2.3)$$

- Bézier tangent demonstration



The illustration above demonstrates the following for this parametric curve, as t varies between 0 and 1:

- The parameter t moves the point (x(t), y(t)) along the path of the curve.
- The point's speed varies as t varies. The speed is higher at the ends of the curve.
- The derivative vector changes in length, reflecting the variation in the speed of the point.
- In the demonstration, the curve crosses itself, which can easily happen with parametric curves.

A convenient notation for equation 2.3 is:

$$\mathbf{f}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} 6 \\ 0 \end{pmatrix} t + \begin{pmatrix} -9 \\ -3 \end{pmatrix} t^2 + \begin{pmatrix} 4 \\ 4 \end{pmatrix} t^3. \qquad (2.4)$$

Equations 2.3 and 2.4 are the same. We simply save on notation by writing the basis functions[5] only once, which are then multiplied by the appropriate vectors. When a vector is multiplied by a scalar, each coordinate in the vector is individually multiplied by the scalar.

In general, a parametric polynomial is written as:

$$\mathbf{f}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \ldots \mathbf{a}_n t^n. \qquad (2.5)$$

where **f**(t) is a vector-valued function, and the **a**'s are vectors. The vectors are not restricted to two dimensions. The **a**'s might be vectors of three dimensions, for instance. In this case the function **f**(t) would have three coordinate functions x(t), y(t), and z(t). The curve would be a curve in space, and the derivative **f**'(t) would be given by the vector of the derivative coordinate functions (x'(t), y'(t), z'(t)).

The general case described by equation 2.5 includes a constant term ($\mathbf{a}_0$), which the example given by equations 2.3 and 2.4 does not have.

---

[5] Here, the basis functions are $1,\ t,\ t^2,$ and t^3.
The coefficient for the basis function "1" is $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

The Parametric Surface

As with curves, it is typical for the reader to have encountered surfaces explicitly as z = f(x, y). Often called *elevation* surfaces or *terrain*, the height z is given at a point on the plane by computing f(x, y). Such a surface definition shares the same flaws mentioned previously for curves:

- They must be single-valued for any point on the plane.
- They cannot have vertical tangent planes.
- Transformations may cause the above two difficulties.

The parametric form of the surface corrects these problems. In order to define a parametric surface, it is best to first define a parametric curve, and then sweep the curve through space to define the surface.

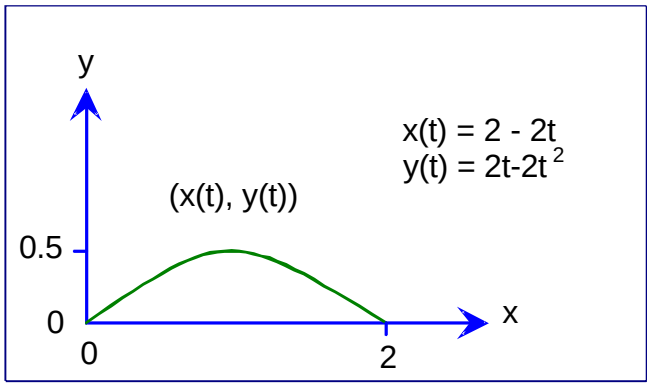
Consider a planar curve given by:

$$\begin{aligned} x(t) &= 2 - 2\,t, \\ y(t) &= 2\,t - 2\,t^2. \end{aligned} \qquad (2.6)$$

Or in vector form,

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} + \begin{pmatrix} -2 \\ 2 \end{pmatrix} t + \begin{pmatrix} 0 \\ -2 \end{pmatrix} t^2. \qquad (2.7)$$

The curve given by x(t) and y(t) looks like this:

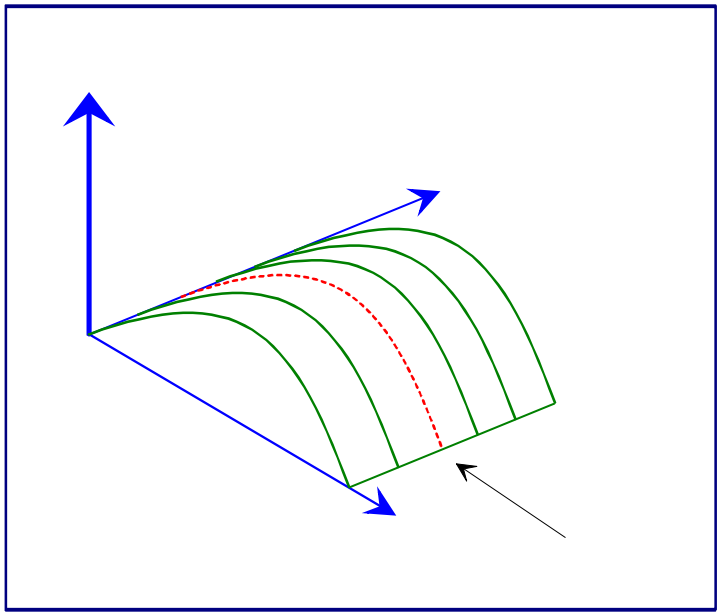


Here, the parameter t is limited to the range 0 to 1.

The curve becomes a surface in three dimensions if another parameter s and another coordinate function z are added. Consider, for instance:

$$x(s,t) = 2 - 2\,t,$$
$$y(s,t) = 2\,t - 2\,t^2, \qquad (2.8)$$
$$z(s,t) = s.$$

When s = 1, the curve defined by equation 2.5 is produced on the plane z = 1. As this curve changes in s it sweeps out a surface. A parametric surface may be thought of as a bundle of parametric curves; by fixing s or t on a surface, one single curve from this bundle is selected.



In this figure, the planar curve is extruded through the z dimension to become a surface. When s = 0.8, the curve is produced as t varies between 0 and 1.

In equation 2.8, x(s,t) and y(s,t) have no terms in s, and z(s,t) has no term in t. The terms are limited to simplify the example, but this is not typical. In general the surface may be written as the parametric polynomial:

$$\mathbf{f}(s,t) = \begin{pmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{pmatrix} \qquad (2.9)$$
$$= \mathbf{a}_{00} + \mathbf{a}_{10}\,s + \mathbf{a}_{01}\,t + \mathbf{a}_{11}\,s\,t + \mathbf{a}_{20}\,s^2 + \mathbf{a}_{02}\,t^2 + \mathbf{a}_{21}\,s^2\,t \ldots$$

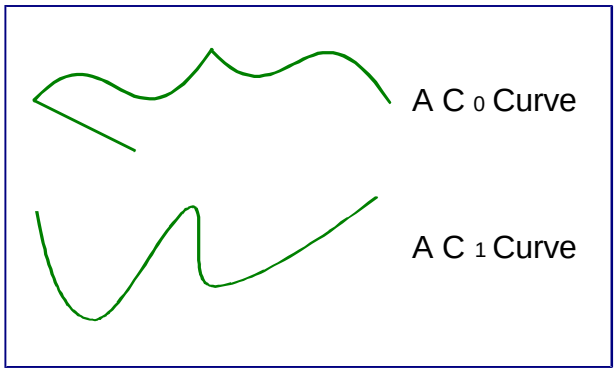
Bold letters indicate vector quantities. The indices of the **a**-vectors correspond to the parametric powers.

Continuity

The notion of continuity [6] was developed for explicit functions to describe when a curve does not break or tear. If it meets these conditions, it is described as C_0. C_0 continuity is defined by the popular description: "A curve is continuous if it can be drawn without lifting the pencil from the paper."

If the derivative curve is also continuous, then the curve is *first-order* differentiable and is said to be C_1 continuous. Extending this idea, it is said that a curve is C_k *differentiable* if the kth derivative curve is continuous.

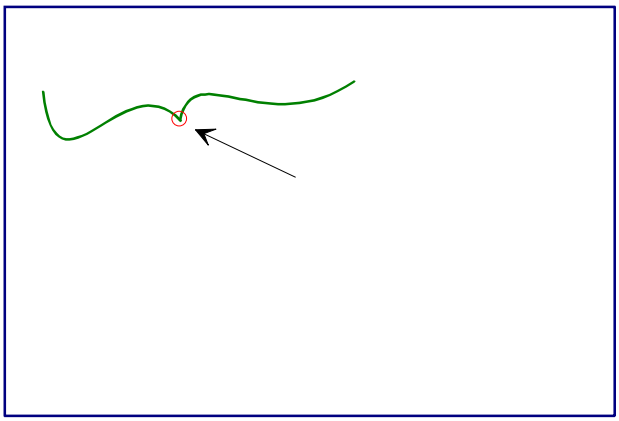
Practically, this means that a C_1 continuous curve will not kink. Higher degrees of continuity imply a smoother curve.



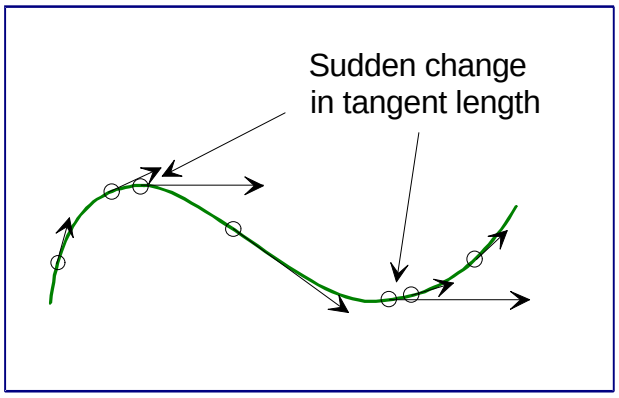
Two curves are shown here, one that is C_0 and one that is C_1. The C_0 curve has a kink, while the C_1 curve is generally smooth.

The traditional notion of C_1 continuity does not, in fact, ensure much about the curve's properties. Imagine, for instance, a particle that travels in a straight line but has distinct jumps in velocity. It is not C_1, but the curve is certainly smooth. Conversely, it is possible to have a C_1 curve with a kink in it. This can occur when the velocity of the particle goes to zero where it changes direction and starts up again. This is illustrated in the following figure:

[6] Continuity implies a notion of smoothness, that is, curves which are not jagged or which break. Commercial applications of CAGD, for example car body design, frequently require that curves and surfaces are continuous.



Mathematicians have developed the concept of a manifold as a new way of describing continuity. In CAGD there is a simpler concept to achieve the same end. It is the idea of geometric continuity [7]. If a curve is C_0, it is G_0 continuous. If a curve's tangent direction changes continuously then it is G_1 continuous. Its magnitude may jump discontinuously but the curve is still G_1. Hence a particle traveling at erratically changing speeds may still trace out a smooth curve if its direction changes smoothly. This is illustrated in the following figure:



If a C_1 curve has kinks because its derivative goes to zero at a point, then this curve will not be G_1, since the tangent direction changes discontinuously at the kink. Hence the notion of geometric continuity provides a useful way to understand the smoothness of a curve or surface.

---

[7] Geometric continuity introduces a notation that immediately tells the designer whether or not the curve is smooth.

Linear Interpolation

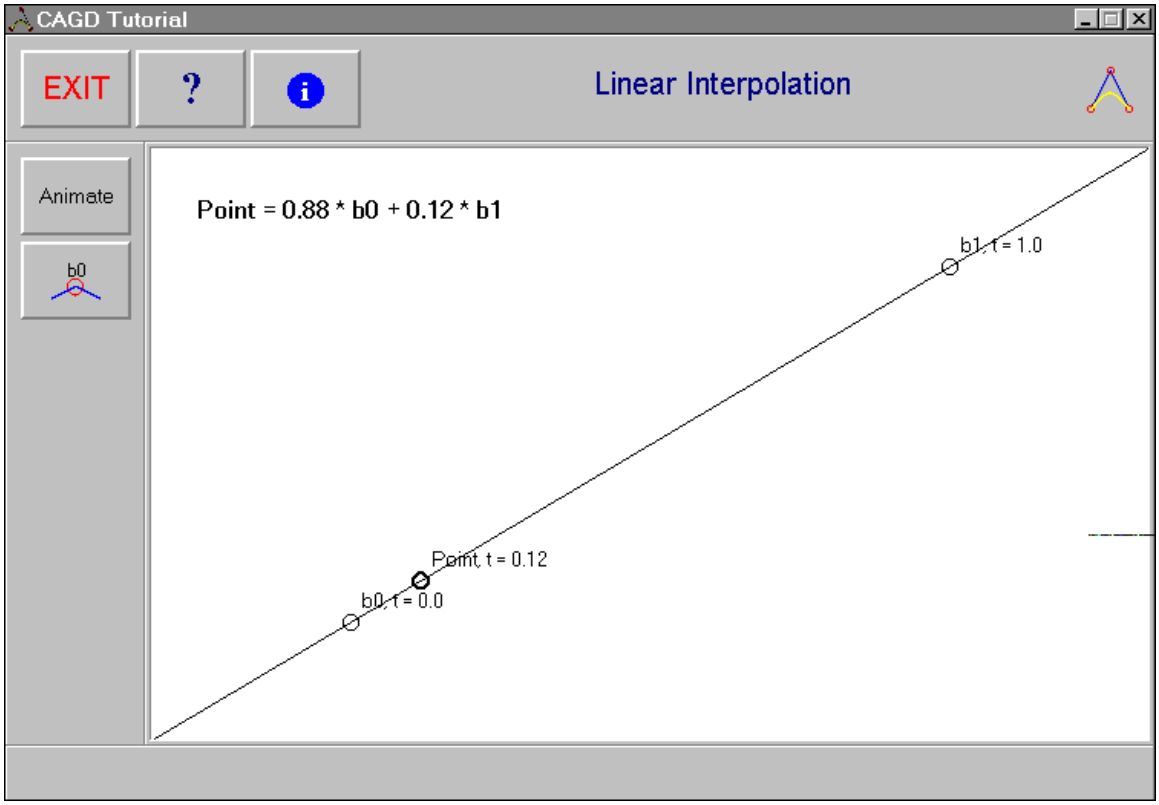
Given two points in space, a line can be defined that passes through them both in parametric form:

$$\mathbf{l}(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1, \qquad (2.11)$$

where $\mathbf{b}_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ and $\mathbf{b}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$, the two points in space.

Thus **l**(t) is a point somewhere in space, depending on the parameter t.

- Linear interpolation between two points: t = 0.125.
  The screen captured from the interactive demonstration displays only two places after the decimal point.



Linear interpolation is perhaps the most fundamental concept. All subsequent curves and surfaces are defined by repeated linear interpolation in some form.

Other forms of linear interpolation are possible. For example,

$$\mathbf{l}(t) = \frac{t\,\mathbf{b}_0}{10} + \left(1 - \frac{t}{10}\right)\mathbf{b}_1,$$

which also gives a straight line through the two points. Note that:

$$\mathbf{l}(0) = \mathbf{b}_1 \text{ and } \mathbf{l}(10) = \mathbf{b}_0.$$

This is the same straight line (a linear combination of the two points), but it has a different parameterization. That is, the motion of a particle at t is different. In most cases it is preferable to start at t = 0 and end at t = 1.

What has Been Accomplished in this Topic

The background to parameterized curves and surfaces has been covered in considerable detail. The hodograph visualized the behavior of the derivative of a curve, and clarified the concept of the motion of a point on the curve.

Continuity was introduced to manage the boundaries between multiple curves or surfaces, and extended to include geometric continuity. Finally, linear interpolation was considered as a prerequisite for Bézier curves and blossoming.

Topic 3

The Bézier Curve

Topic 3: The Bézier Curve

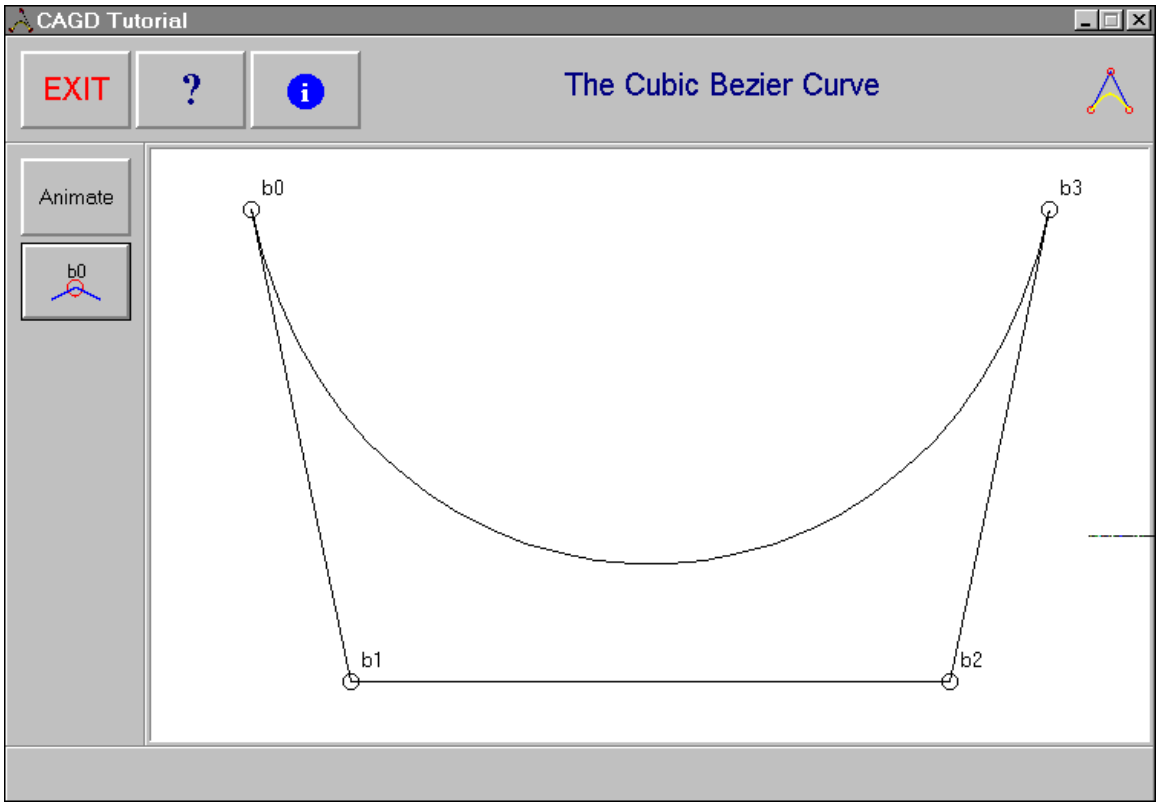
In this topic, you will learn:

- What a Bézier curve is.
- The properties and behavior of the Bézier curve.
- How to create a Bézier curve.
- The de Casteljau algorithm for evaluati on of a Bézier curve.
- Subdivision and differentiation of the Bézier curve.

Introduction to the Bézier Curve

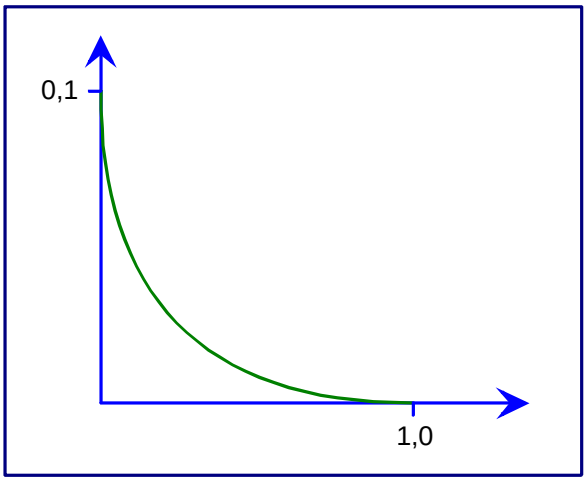
The Bézier curve is a good place to start a study of CAGD. It underpins other concepts such as B-splines and surface patches. It is visually engaging and exhibits many desirable properties for design.

- The cubic Bézier curve:



Mathematical Properties of the Bézier Curve

Consider the parabola that passes through (0,1) and (1,0) and is tangent to the x and y axes at these points:



The parametric form of a parabola looks like:

$$\mathbf{f}(t) = \mathbf{a}\,t^2 + \mathbf{b}\,t + \mathbf{c}, \qquad (3.1)$$

where **a**, **b**, and **c** are vector coefficients. **f**(t) is a vector function which has two components, i.e. **f**(t) = (x(t), y(t)). The above parabola can be written as

$$\mathbf{f}(t) = \begin{pmatrix}1\\1\end{pmatrix}t^2 + \begin{pmatrix}-2\\0\end{pmatrix}t + \begin{pmatrix}1\\0\end{pmatrix}. \qquad (3.2)$$

This can be rewritten as:

$$\mathbf{f}(t) = \begin{pmatrix}1\\0\end{pmatrix}(1-t)^2 + \begin{pmatrix}0\\0\end{pmatrix}2t(1-t) + \begin{pmatrix}0\\1\end{pmatrix}t^2. \qquad (3.3)$$

The reformation process is shown below [1].

---

[1] This parabola is defined parametrically by:

$$y(t) = t^2$$

and:

$$x(t) = 1 - 2t + t^2 = (1-t)^2.$$

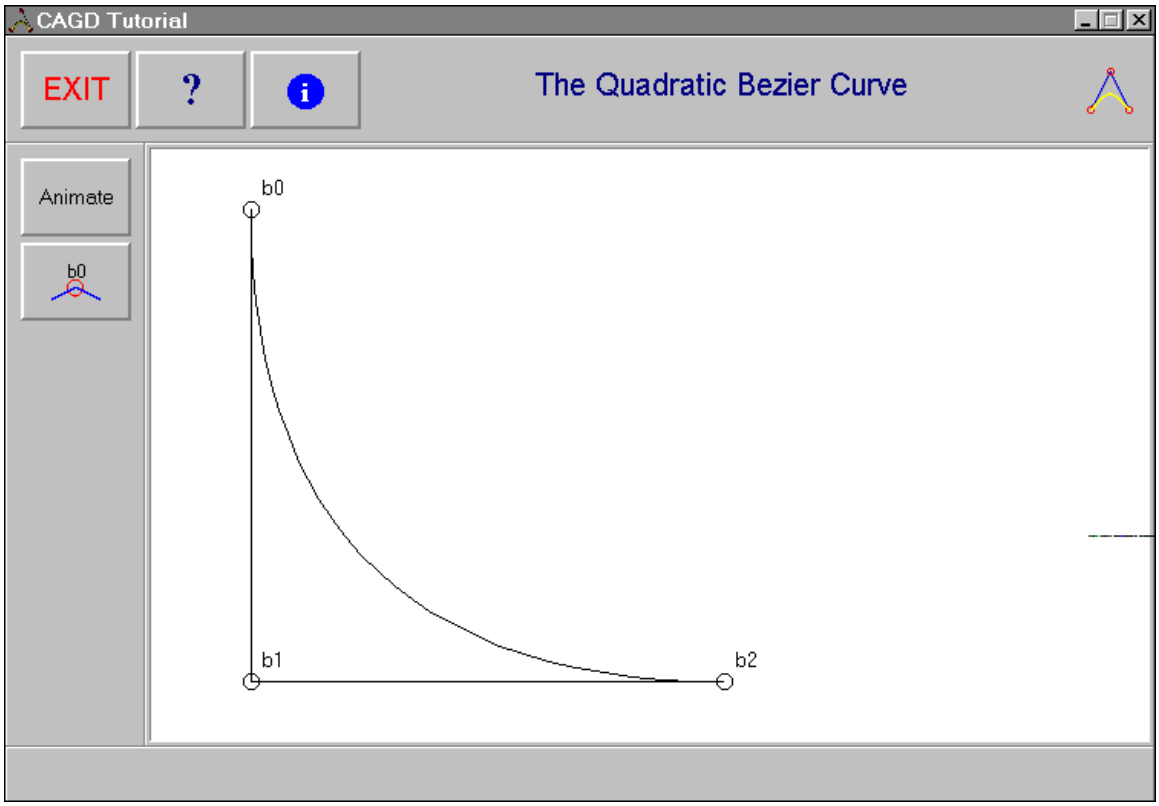
This is exactly the same curve as equation 3.2, so what advantage is there in rewriting? The advantage lies in the geometrical meaning of the coefficients: (1,0), (0,0), and (0,1). These are called control points. Together, the control points form the control polygon.

Observe:

- The curve passes through the endpoints of the control polygon.
- The curve is cotangent to the control polygon at these endpoints.

The curve in this form is called the Bézier curve, and the observations hold in general for any coefficients. This means that if the coefficients are changed, the curve changes in an easy-to-understand way.

- The quadratic Bézier curve:



In vector form, it may be expressed as:

$$\mathbf{f}(t)=\begin{pmatrix}x(t)\\y(t)\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}(1-t)^2+\begin{pmatrix}0\\0\end{pmatrix}2t(1-t)+\begin{pmatrix}0\\1\end{pmatrix}t^2.$$

The general form for a quadratic Bézier curve is:

$$\mathbf{f}(t) = \mathbf{b}_0 (1-t)^2 + \mathbf{b}_1 2t(1-t) + \mathbf{b}_2 t^2. \qquad (3.4)$$

This is a parabola exactly as equation 3.1 but it is rewritten so that the control points
$\mathbf{b}_0$, $\mathbf{b}_1$, and $\mathbf{b}_2$ have geometrical significance as the control points of the parabola.

Bézier Curves of General Degree

The general form of a Bézier curve of degree n is:

$$\mathbf{f}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t), \qquad (3.5)$$

where $\mathbf{b}_i$ are vector coefficients, the now-familiar control points, and:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}. \qquad (3.6)$$

$\binom{n}{i}$ is the binomial coefficient; $B_i^n(t)$ are called the Bernstein functions.

Binomial Coefficients

The binomial coefficients, commonly derived from Pascal's triangle, may be computed:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \tag{3.7}$$

for i, an integer ≥ 0.

Note that $\binom{n}{0} = 1$ and, in particular, $\binom{0}{0} = 1$.

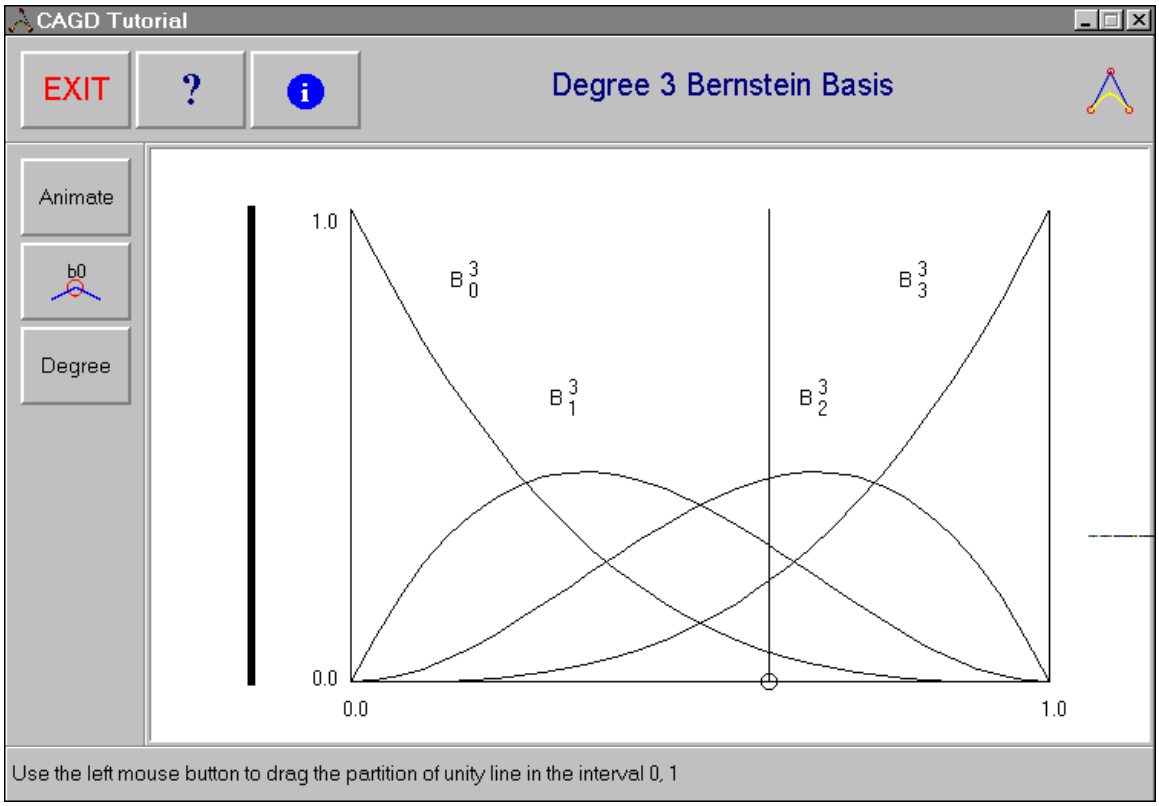
Bernstein Functions

The Bernstein functions were originally devised by Bernstein to prove the famous Weierstrass Theorem over 150 years ago. They are formally given by:

$$B_i^n(t) = \frac{n!}{i!(n-i)!}(1-t)^{n-i}\,t^i. \tag{3.8}$$

They have many useful properties for curve generation.

- The cubic Bernstein basis functions:



An important characteristic of the Bernstein functions is the partition of unity. This simply means that the sum of the functions is always one, for all values of t:

$$\sum_{i=0}^{n} B_i^n(t) = 1 \qquad (3.9)$$

The collection of Bernstein functions for i = 0, 1, ... n is the Bernstein Basis.

The Bernstein Basis is a key to understanding Bézier curves. Many of the important properties that make Bézier curves useful in design derive from these basis functions.

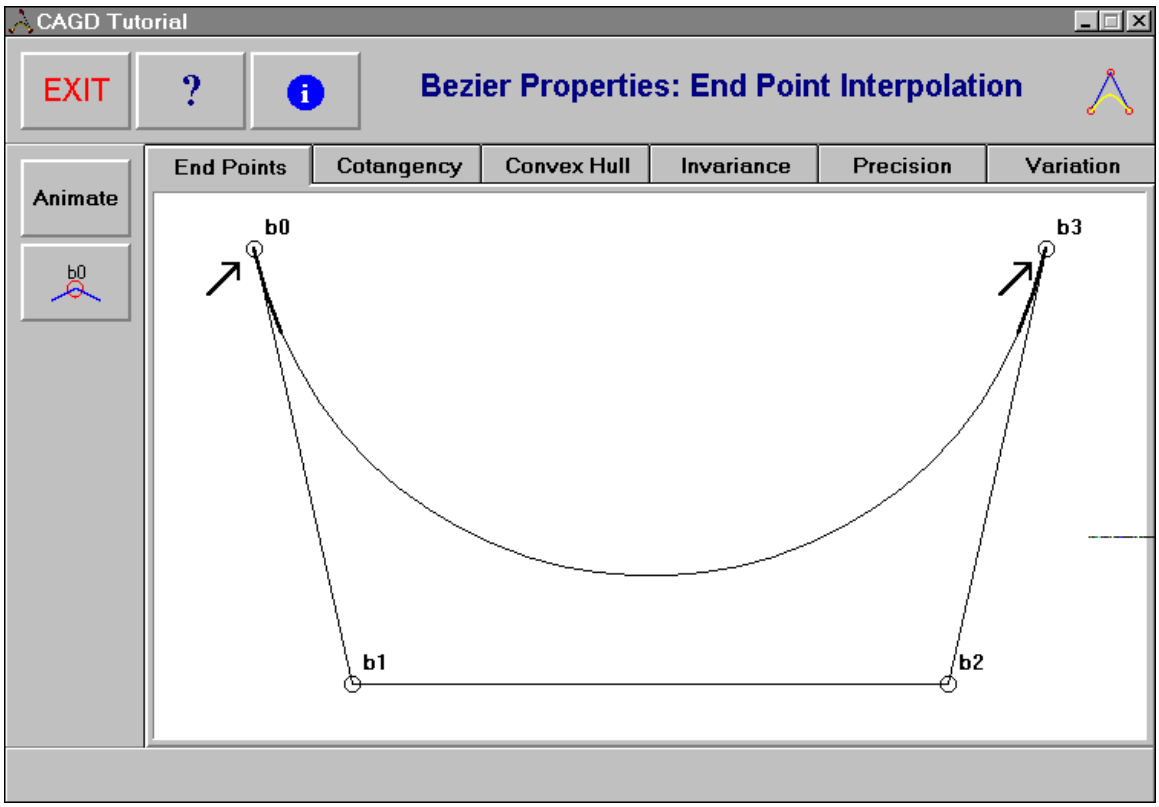
Characteristics of the Bézier Curve

Bézier curves have a number of characteristics which define their behavior.

- Endpoint Interpolation

  The Bézier curve interpolates the first and last points $\mathbf{b}_0$ and $\mathbf{b}_n$. In terms of the interpolation parameter t: $\mathbf{f}(0) = \mathbf{b}_0$ and $\mathbf{f}(1) = \mathbf{b}_n$. This property derives from the Bernstein functions, since at the the endpoints the Bernstein functions are zero except:

  At $\mathbf{b}_0$, $B_0^3 = 1$; at $\mathbf{b}_3$, $B_3^3 = 1$

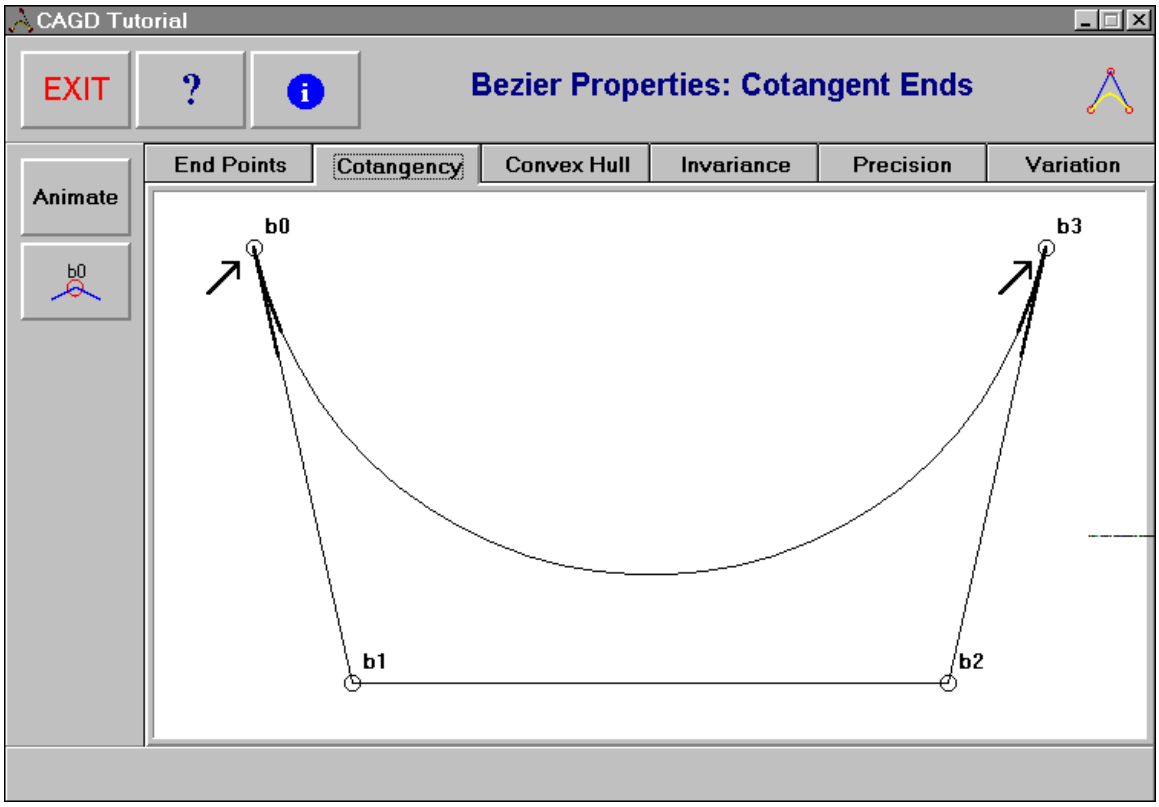


- Tangent Conditions

  The Bézier curve is tangent to the first and last segments of the control polygon, at the first and last control points. In fact:

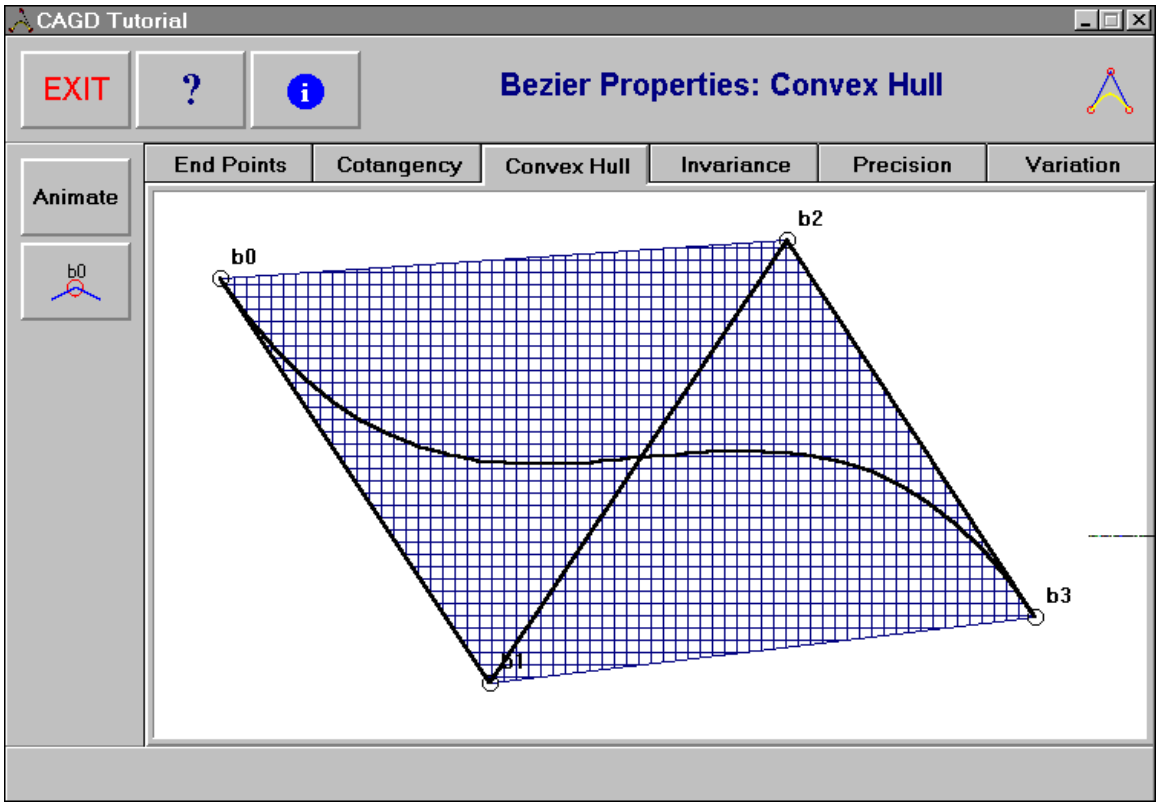
  $$\mathbf{f}'(0) = (\mathbf{b}_1 - \mathbf{b}_0)n \text{ and } \mathbf{f}'(1) = (\mathbf{b}_n - \mathbf{b}_{n-1})n,$$

  where n is a constant.

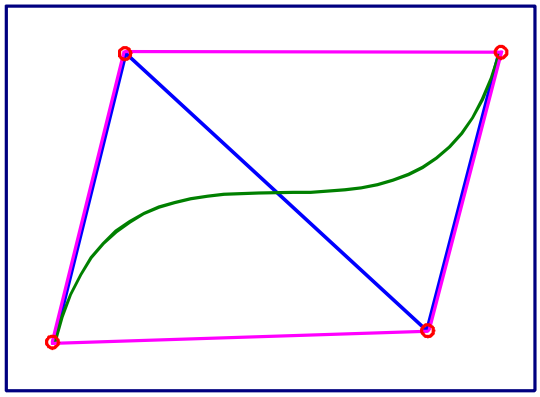


- Convex Hull

  The Bézier curve is contained in the convex hull [2] of its control points for $0 \le t \le 1$.

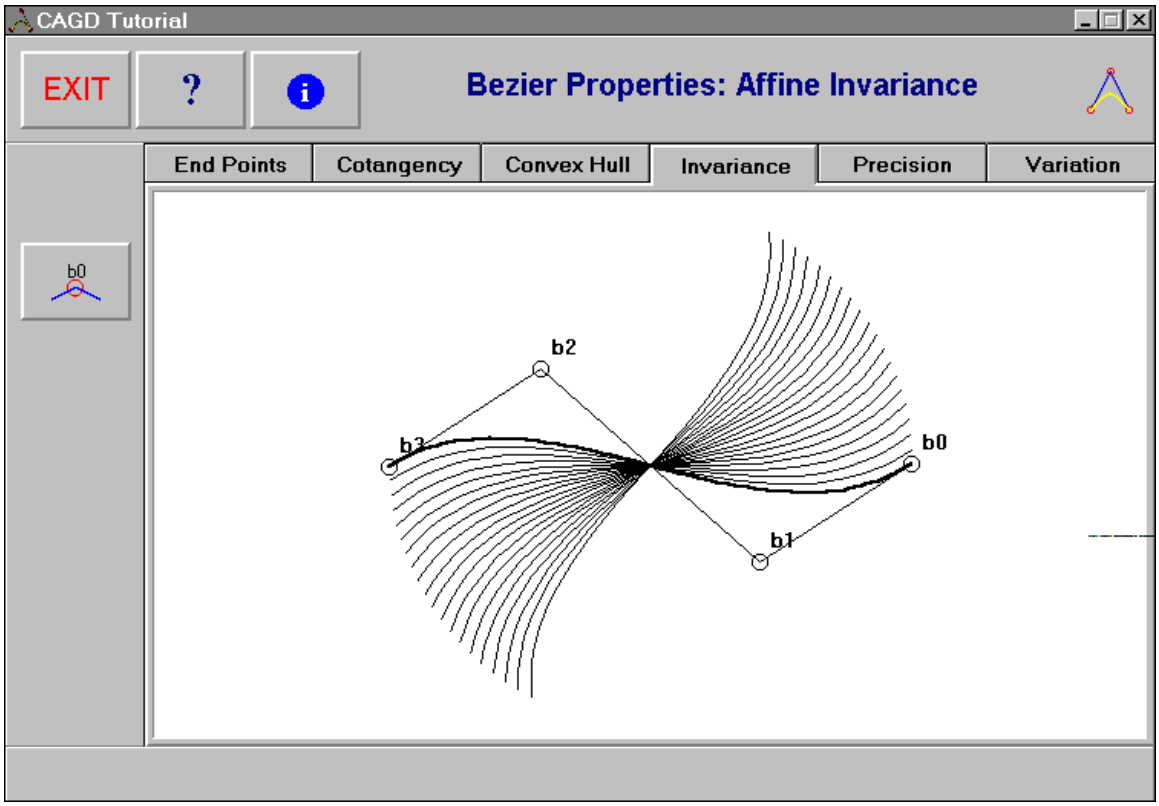


[2] The convex hull of a control polygon is the minimal convex enclosure of the control polygon.



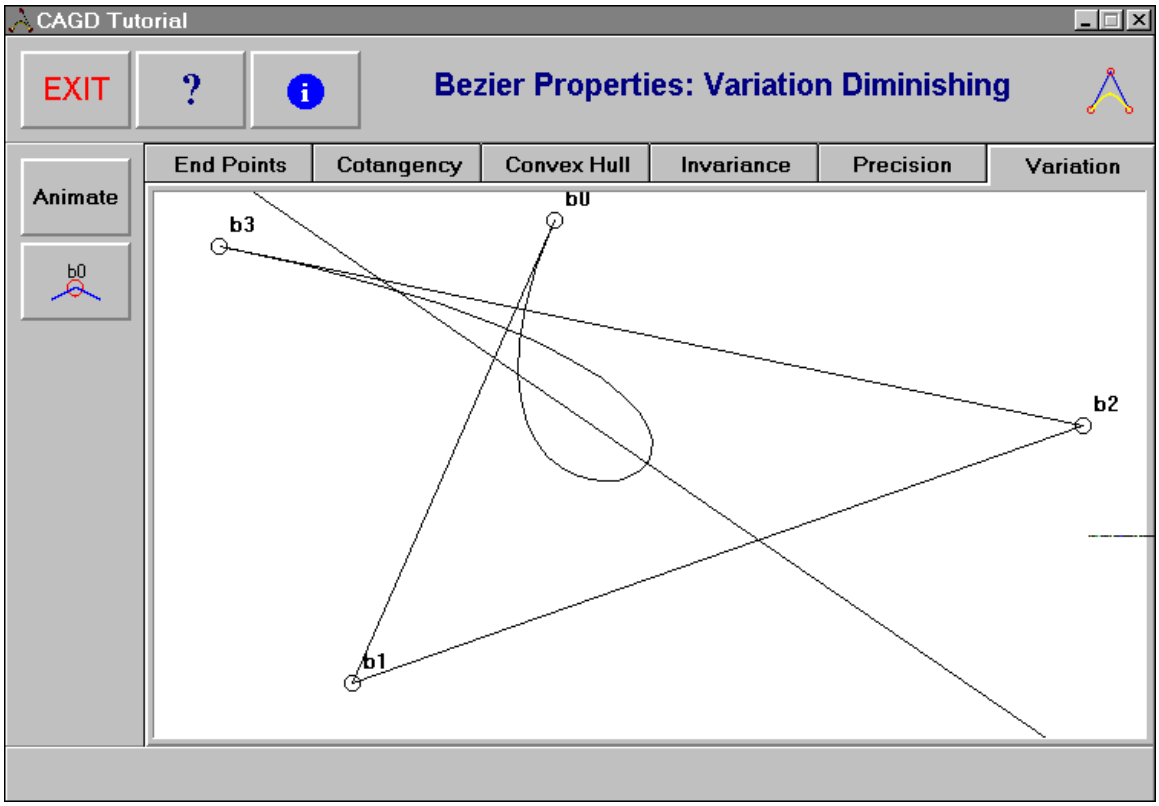
- Affine Invariance

  The Bézier curve is affinely invariant with respect to its control points. This means that any linear transformation (such as rotation or scaling) or translation of the control points defines a new curve which is just the transformation or translation of the original curve.



- Variation Diminishing

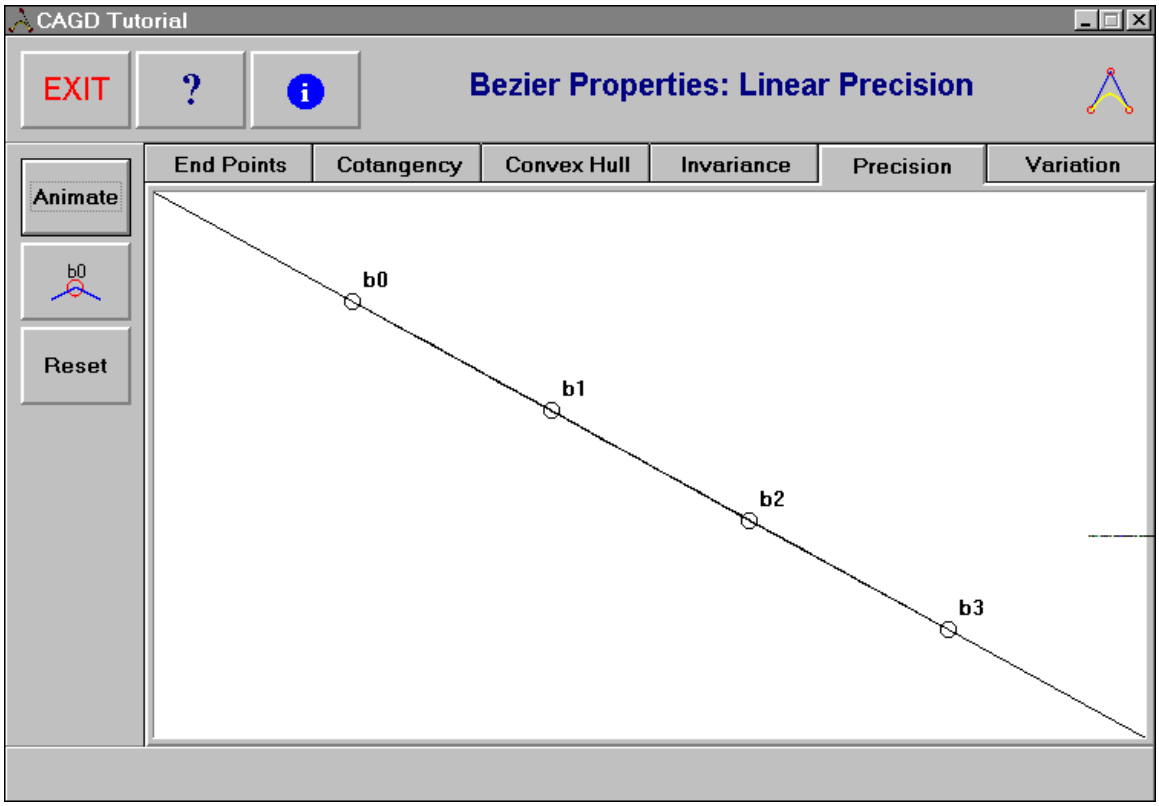
  The Bézier curve is variation diminishing. It does not wiggle [3] any more than its control polygon; it may wiggle less. In this figure, notice that the straight line intersects the convex hull three times and also intersects the Bézier curve three times.



[3] By "wiggle," we mean the way in which a curve or surface changes direction. This is more precisely expressed as a change in sign of the curvature of the curve or surface.

- Linear Precision

  The Bézier curve has linear precision: If all the control points form a straight line, the curve also forms a line. This follows from the convex hull property; as the convex hull becomes a line, so does the curve.



The de Casteljau Algorithm

Evaluation of the Bézier curve function at a given value t produces a point **f**(t). As t varies from 0 to 1, the point **f**(t) traces out the curve segment. One way to evaluate equation 3.5 is by direct substitution, that is, by applying the value of t to the formula and computing the result.

This is probably the worst method of evaluating a point on the curve! Numerical instability, caused by raising small values to high powers, generates errors.

There are several better methods available for evaluating the Bézier curve. One such method is the de Casteljau algorithm. This method not only provides a general, relatively fast, and robust algorithm, but it gives insight into the behavior of Bézier curves and leads to several important operations on the curves, such as:

- Computing Derivatives

  The derivative of the curve gives the tangent vector at a point.

- Subdividing the Curve

  It is sometimes necessary to take a single Bézier curve and produce two separate curve segments that together are identical to the original. To accomplish this, it is necessary to find two sets of control points for the two new curves.

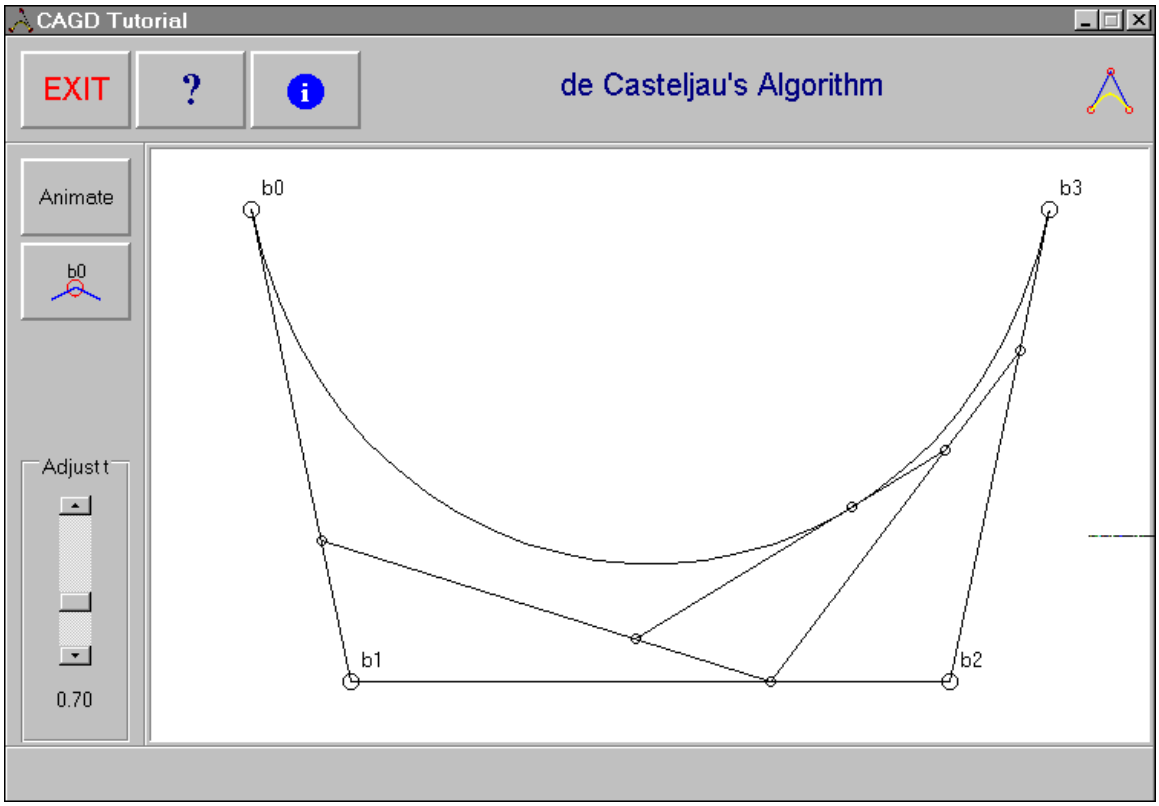
The de Casteljau algorithm can be regarded as repeated linear interpolation.

As described in the section on linear interpolation in Topic 2, Preliminary Mathematics, it is possible to interpolate between two points $\mathbf{b}_0$ and $\mathbf{b}_1$ with the equation:

$$\mathbf{f}(t) = \mathbf{b}_0 (1 - t) + \mathbf{b}_1 t. \qquad (3.10)$$

If t = 0.5 then **f**(t) is the midpoint of the line between the endpoints.
Equation 3.10 is just a Bézier curve of degree n = 1.

- A demonstration of de Casteljau's algorithm:



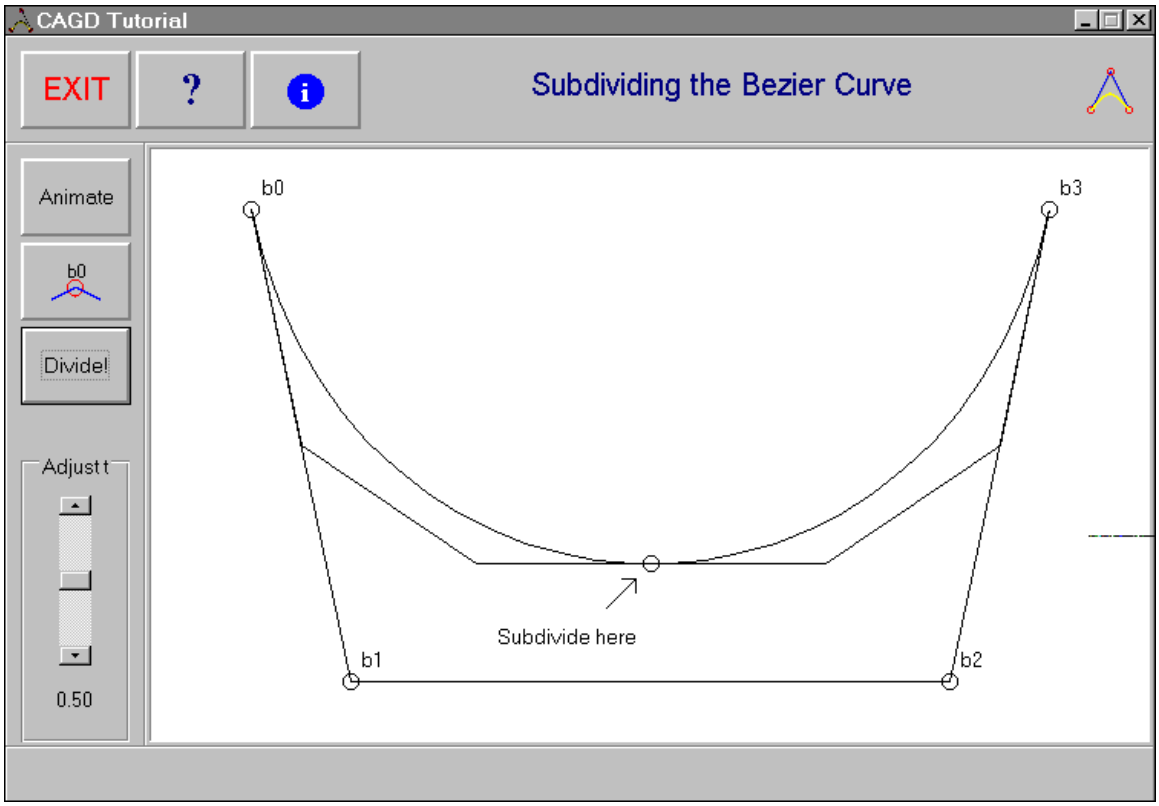
Subdivision of a Bézier Curve

One of the most important operations on a curve is that of subdividing it. The de Casteljau algorithm not only evaluates a point on the curve, it also subdivides a curve into two parts as a bonus. The control points of the two new curves appear along the sides of the systolic array. The new curves match the original in position, although they differ in parameterization.

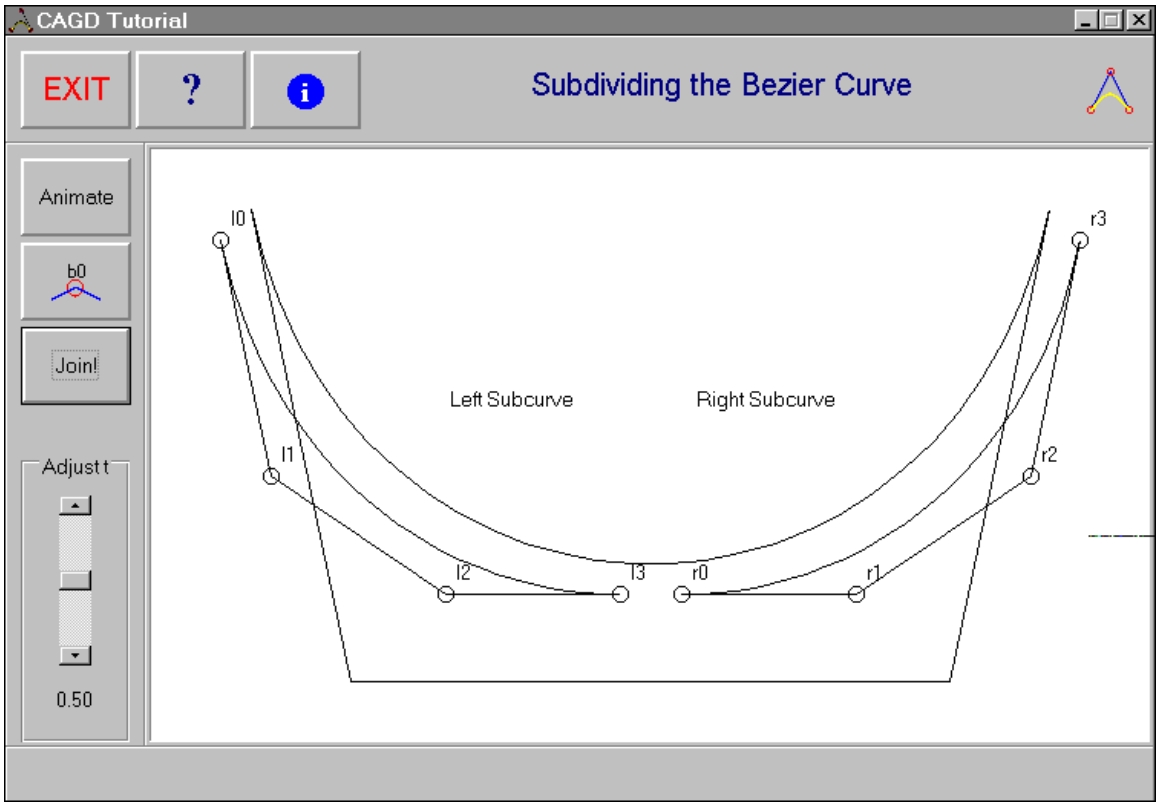
Uses of Subdivision

- Design Refinement
  Subdivision permits existing designs to be refined and modified. For example, additional curves may be incorporated into an object. This is accomplished by adding more control points for local control.
- Clipping a Curve to a Boundary
  One method of intersecting a Bézier curve with a line is to recursively subdivide the curve, testing for intersections of the curve's control polygons with the line. Curve segments not intersecting the line are discarded. This process is continued until a sufficiently fine intersection is attained.

- A cubic Bézier curve before subdivision:



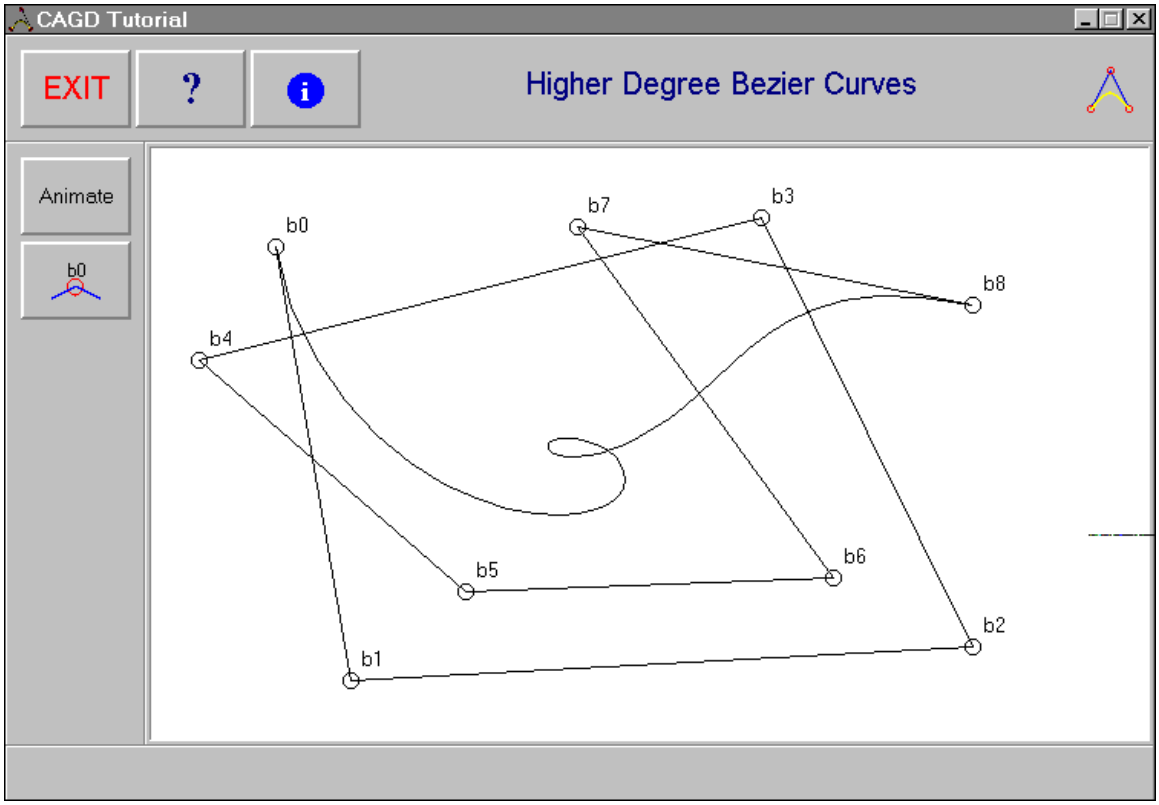
- A cubic Bézier curve after subdivision:



Higher Degree Bézier Curves

Bézier curves of any degree may be created. Degree 2 curves (quadratic curves) are the lowest degree useful. Many commercial applications and drawing packages use degree 3 Bézier curves (cubic curves) as a drawing primitive.

- A degree 8 Bézier curve:



The Derivative of the Bézier Curve

It is a straightforward exercise in algebra to differentiate the Bézier function and then to formulate the derivative in Bézier form. The derivative of a Bézier curve of degree n is:

$$\mathbf{f}'(t) = n\sum_{i=0}^{n-1}(\mathbf{b}_{i+1} - \mathbf{b}_i)B_i^{n-1}(t). \qquad (3.12)$$

There is one less term in the derivative than in the original function. Also, the degree of the Bernstein polynomials is one less. The control points for the

derivative curve are successive differences of the original curve's control points, having the form:

$$\mathbf{b}_{i+1} - \mathbf{b}_i.$$

Consider the derivatives at the endpoints of the Bézier curve:

$$\mathbf{f}'(0) = n(\mathbf{b}_1 - \mathbf{b}_0), \text{ and } \mathbf{f}'(1) = n(\mathbf{b}_n - \mathbf{b}_{n-1}), \qquad (3.13)$$

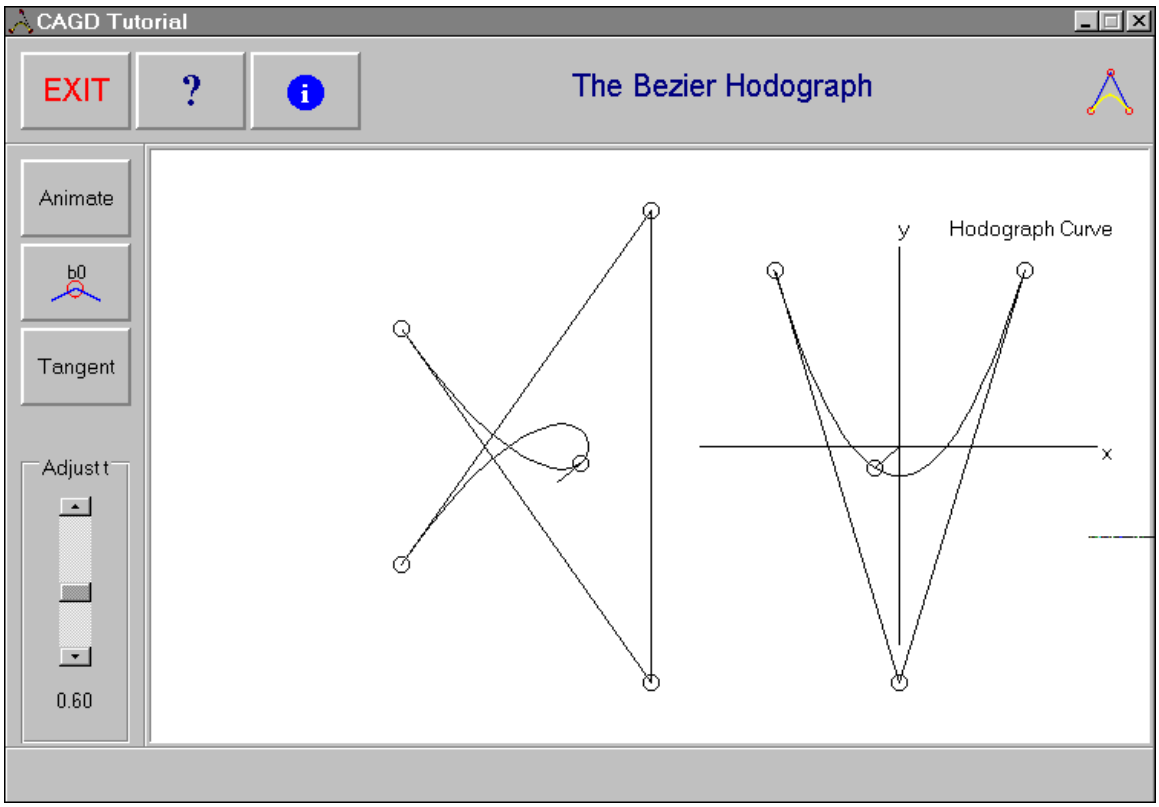
where n is a constant.

The tangent endpoint property is derived from these two derivatives. It may be seen that the derivatives (the tangents) at the endpoints are n times the first and last legs of the control polygon.

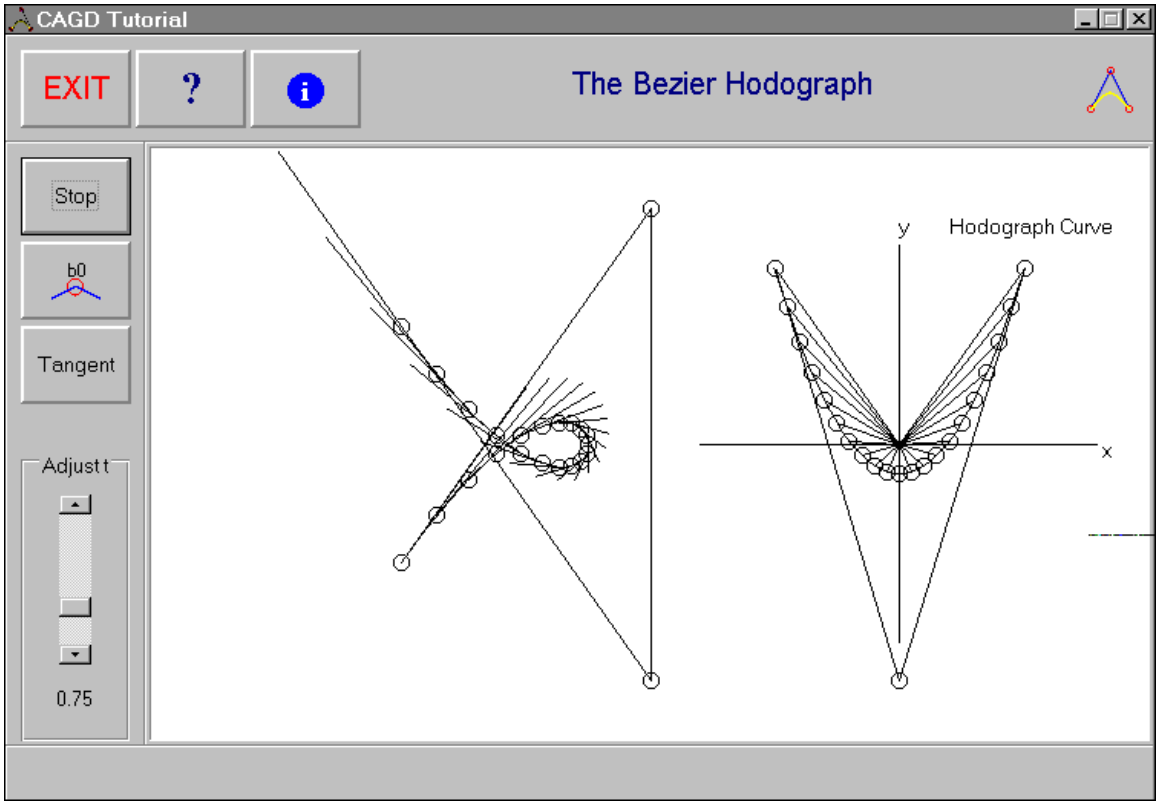
The hodograph of the Bézier curve is easy to construct in Bézier form. It is a Bézier curve with control points given by:

$$\Delta\mathbf{b}_i = \mathbf{b}_{i+1} - \mathbf{b}_i, \; i = 0,1,\ldots,n-1 \qquad (3.14)$$

- The hodograph of the Bézier curve with its control polygon shown:

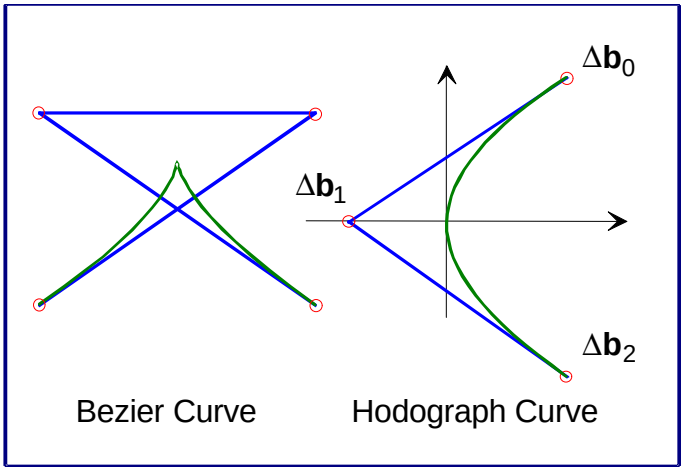


- As before, the hodograph is created by plotting a series of tangent vectors of the original Bézier curve:



Continuity of the Bézier Curve

Earlier discussions on continuity focused on the differences between C1 and G1 continuity. The following figure shows a curve that is C1 but not G1; a particle tracing out the path of the curve would undergo a sudden reversal of direction at the kink, so the curve is not G1. However, its hodograph is smooth, which makes the curve C1.



Notice how the derivative goes to zero at the kink of the curve. This permits a smooth hodograph, without a smooth curve. Clearly, the superior measure of smoothness provided by geometric continuity is desirable.

What has Been Accomplished in this Topic

The Bézier curve has been covered in depth, with discussions of the properties of the curve. de Casteljau's algorithm generates the curve using iterated linear interpolation, providing a fast and stable way of creating Bézier curves. Subdivision, degree elevation, and differentiation of the Bézier curve completed the topic.

Topic 4

Blossoms

Topic 4: Blossoms

In this topic, you will learn:

- What a blossom is, and the connection between blossoms and CAGD.
- The characteristics of blossoms.
- The application of blossoming to Bézier and B-spline curves.
- The uses of blossoms in CAGD.

Introduction

"What's in a name? That which we call a rose, by any other name would smell as sweet."

Shakespeare, Romeo and Juliet.

In some instances, the name given to an object has little importance. A fragrance, for example, does not depend on its name. This is not true in mathematics, however; labels are significant. Well-chosen notation serves not only as a tag, but also suggests how a concept is defined and used. This is especially true of a technique called blossoming. Its power lies in its ability to suggest fundamentals, algorithms, and theorems in CAGD through labeling.

The theory of blossoms was introduced by Ramshaw and de Casteljau.

Blossom Basics

The basic principle of blossoming arises from linear interpolation [1]. Recall that **b**(0) and **b**(1) are points on the line segment at t = 0 and t = 1. Any point on the line is given by **b**(**a**). The distance from **b**(0) to **b**(**a**) is proportional to $|\ 0 - \mathbf{a}\ |$. Similarly, the distance from **b**(**a**) to **b**(1) is proportional to $|\ 1 - \mathbf{a}\ |$. It is very useful to think of **a** as a measure of how far a point travels from **b**(0) to **b**(**a**). For example, if **a** = 0.5, then **b**(**a**) is halfway between **b**(**a**) and **b**(1).

Notation

To further emphasize this relationship between **a** and **b**(0)-**b**(1), the functional notation is dropped and simply written **b**(0) = **0**, **b**(1) = **1**, and **b**(**a**) = **a**. **a** is spoken of loosely as the affine distance [2] from **0** to **a**.

Essentially, a point is designated by its parameter value **a** and the endpoints of the segment on which it lies, with the understanding that the point is obtained by linear interpolation along the line.

This is shown in the following figure, which clarifies the new nomenclature.

---

[1] Given two points in space, a line in parametric form can be defined that passes through them:

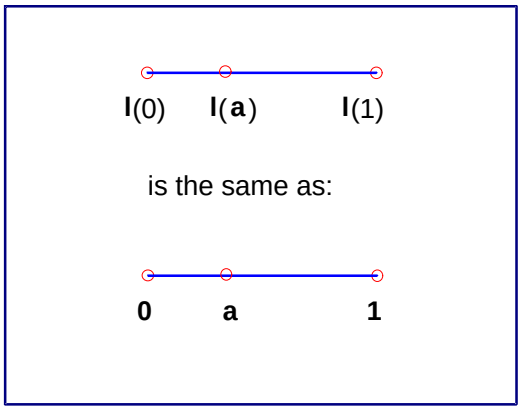
$$\mathbf{l}(t) = (1-t)\mathbf{b}_0 + t\,\mathbf{b}_1,$$

where $\mathbf{b}_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ and $\mathbf{b}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$, the two points in space.

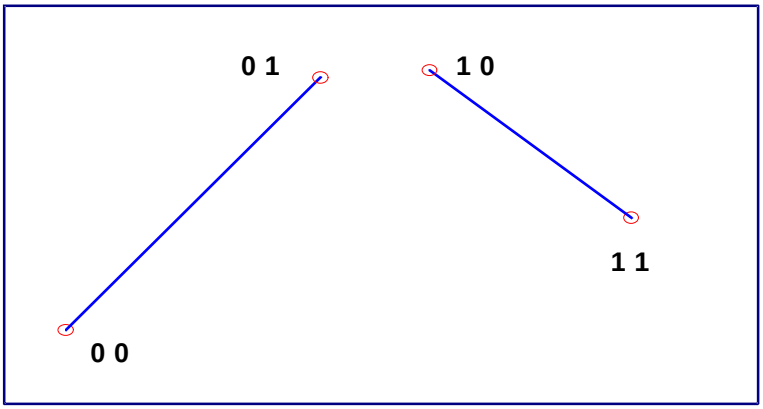
Thus **l**(t) is a point somewhere on the line between the two points, depending on the parameter t.

[2] The affine distance of a point **a** to a point **b** on a line **bc** is the ratio of the distances:

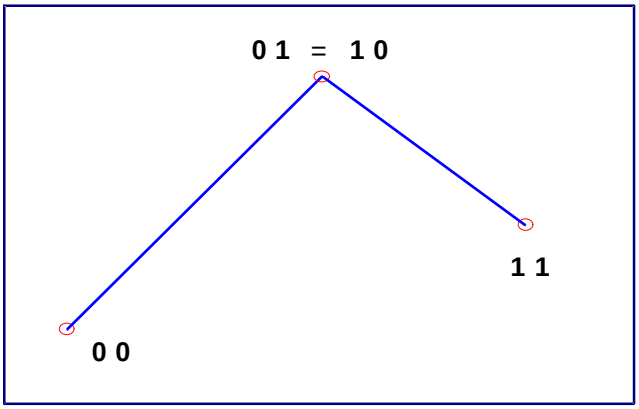
$$\frac{|\mathbf{ba}|}{|\mathbf{bc}|}.$$



Next an important extension is made to the notation in order to describe two lines that share a point. This is done by adding an additional digit as shown in the figures below. First, add the digits to the two individual lines. The first line is identified with a leading **0**, it goes from **00** to **01**. The second line has a leading **1**, going from **10** to **11**.

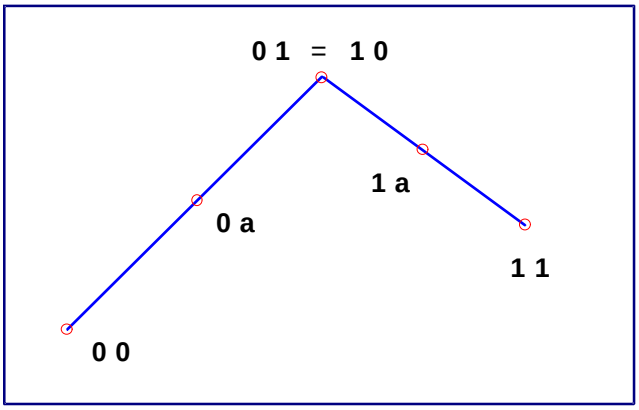


Next, bring the lines together so that they share an endpoint:



Note here (this is most important, even though it seems obvious) that point **01** and point **10** are identical.

When there was a single line, the point **a** was on the line from **0** to **1**. In the two-line figure, if the point **0a** is on the line from **00** to **01**, then where is the point **1a**? Clearly, on the new line:

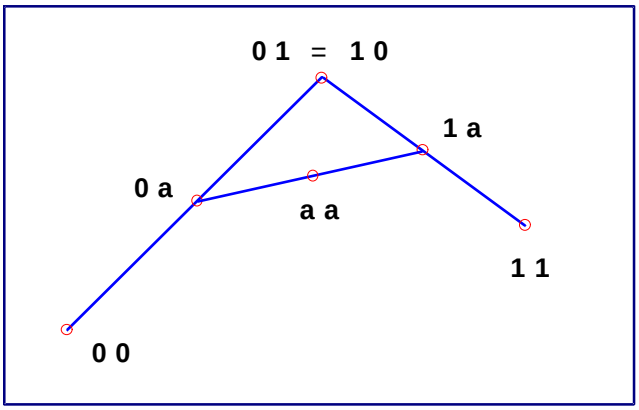


Point Ordering is Not Significant

The next generalization comes from noticing that the order of the digits is not important: **0a** or **a0** can be used to identify uniquely the point on the first line. Therefore the digits may be in any order:

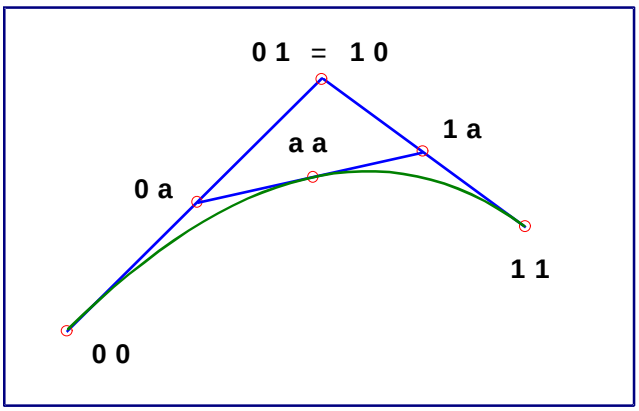
01 ≡ **10**,
a0 ≡ **0a**,
a1 ≡ **1a**.

Now consider a new line from **0a** to **1a**. Where is **aa**? It is the affine distance **a** between the line's endpoints, **0a** and **1a**. This is shown in the following figure:

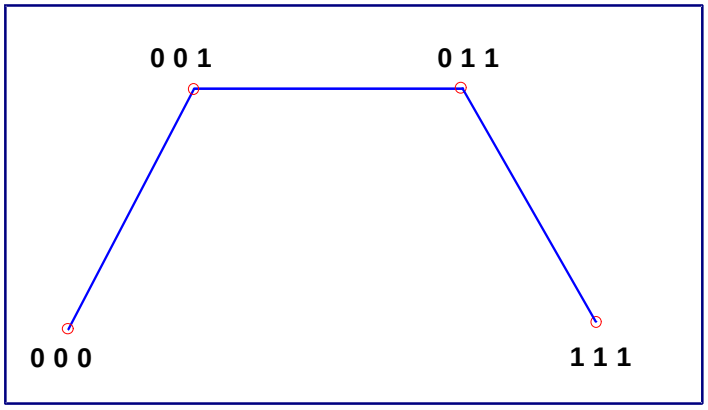


Furthermore it should be clear that **aa**, by its construction, is a point on a quadratic Bézier curve with control points **00**, **01**, and **11**. This follows from

understanding de Casteljau's algorithm, which is base d on repeated linear interpolation. The Bézier curve is shown in the following figure:

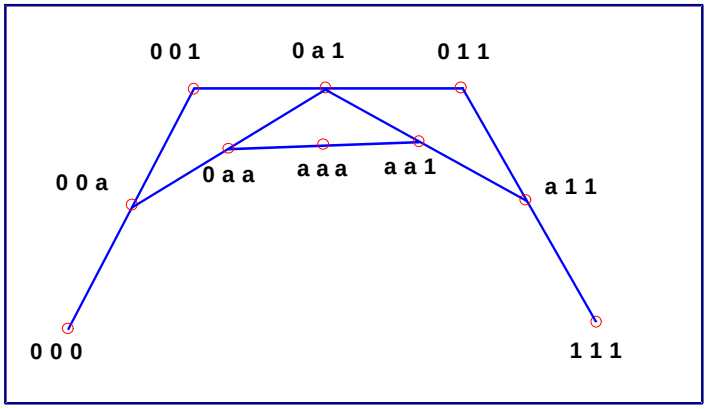


To add another control point, add another digit:

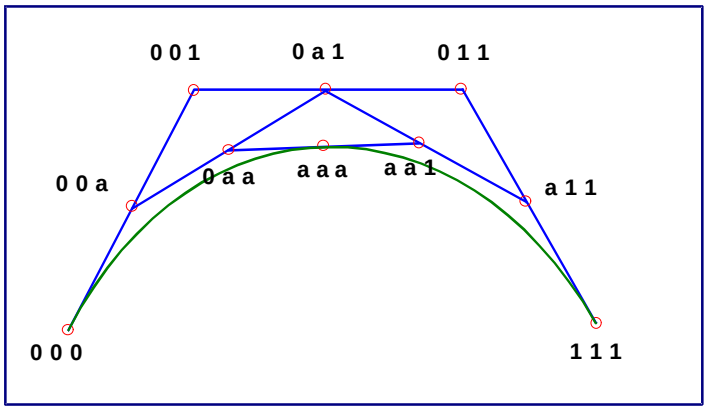


Here a convention is adopted of ordering the digits from smallest to largest. For example, **0a1** is preferred to **1a0**, so long as $0 \leq a \leq 1$. Sometimes commas are used to separate the digits, if required for clarity. The point **0**,**.5**,**1** is not **0.5 1**.

In the last figure, where is the point **aaa**? It can be found by recursive replacement. This is the so-called blossoming principle [3]. First find **00a**, **0a1**, and **a11**, on the appropriate lines. Then find **0aa** and **aa1**. Finally, **aaa** may be found on the line between **0aa** and **aa1**:



By construction, the point **aaa** is on the Bézier curve with control points **000**, **001**, **011**, and **111**:



---

[3] Informally, the digit that differs between two blossoms is replaced with another value, giving the affine distance along the line segment for the new value.

Thus **axx** is found on the line segment between **bxx** and **cxx**, an affine distance given by:

$$\frac{|\mathbf{bxx} - \mathbf{axx}|}{|\mathbf{cxx} - \mathbf{bxx}|}.$$

Summary of Blossoming from Linear Interpolation

The generalization of this method for a Bézier curve of any degree should now be clear. The principles generated thus far may be summarized:

1. There is a Bézier curve of degree n that is given by points of the form **aa....a**. There are n **a**'s.
2. The points are equivalent regardless of the ordering of the digits: **a10** ≡ **01a** ≡ **0a1**.
3. The point **0...0a1...1** is an affine distance **a** along the line from **0...001...1** to **0...011...1**. In other words:

   0a1 = (1 - **a**) **001** + (**a**) **011**.

   This is simple linear interpolation.

The History of Blossoms

It was proved by Ramshaw and de Casteljau that for every polynomial P(u) of degree n there is a unique function of n variables $p(u_1, u_2, ..., u_n)$ which has the three properties listed in the summary above. Ramshaw called this function the blossom of the polynomial P(u). Quite often this blossom is given only as a list of its variables, as shown. The blossom notation suggests methods and algorithms naturally. The de Casteljau algorithm arose by recursively applying the principles until **a...a** was found.

The Power of the Blossom Form

Blossoms are very powerful. Once the principle is understood, methods may be generated to evaluate the Bézier curve, prove properties, define and evaluate B-spline curves, convert between B-spline and Bézier form, and much more.

The following section is intended for those who want more practice with blossoms. The material proves some new and some old things about Bézier curves by using blossoms. For example, one thing shown is that Bézier control points always have the form:

$\mathbf{b}_i = 0\ldots01\ldots1$ (there are i ones).

The Bézier Curve in Blossom Form

Using the three properties of blossoms previously described, the following can be observed:

For the blossom $p(u_1, u_2, \ldots, u_N)$:

$$
\begin{aligned}
p(u,u,\ldots,u) &= p\big((1-u)\cdot 0 + u\cdot 1, u, \ldots, u\big) \\
&= (1-u)p(0,u,\ldots,u) + up(1,u,\ldots,u) \\
&= (1-u)^2 p(0,0,u,\ldots,u) + 2(1-u)up(0,1,u,\ldots,u) + u^2 p(1,1,u,\ldots,u) \\
&\ \ \vdots \\
&= (1-u)^N p(0,\ldots,0) + N(1-u)^{N-1} up(0,\ldots,0,1) + \ldots \\
&\quad + N(1-u)u^{N-1} p(0,1,\ldots,1) + u^N p(1,\ldots,1).
\end{aligned}
$$

The point p(u, u,..., u) is on the curve P(u) by the diagonal property. In the first equation, the first u in the argument list is written as an affine combination: u = (1 - u).0 + u.1.

These steps are repeated for the next u in the argument list and the terms are combined using the symmetry property to arrive at the third line. Continuing in this manner, the argument list is exhausted.

The polynomial, as seen in the last line, is a summation of coefficients, written as blossoms, multiplied by the Bernstein basis polynomials. This polynomial is a Bézier curve of degree N. This can already be seen to be emerging in the third equation; the polynomials in each term are quadratic Bernstein polynomials.

The coefficients, or control points, are blossoms evaluated with only 0's and 1's, so the blossoms p(0, 0,..., 1, 1) are then the Bézier control points, that is:

$$\mathbf{b}_i = p(0,\ldots,0,1,\ldots,1);$$

There are i ones in the blossom. This control point is the blossom evaluated at (N - i) 0's and i 1's; the order does not matter.

Evaluating a Blossom from Bézier Control Points

Suppose that a set of Bézier control points is given. What is the point given by the blossom at $\mathbf{u}_1 \ldots \mathbf{u}_n$? The blossom properties lead to an algorithm that evaluates the blossom. It has the familiar form of iterated linear interpolation.

EvalBlossomProg: A General Blossom Algorithm

The de Casteljau algorithm generalizes to the EvalBlossom procedure. This assumes a [0, 1] range of curve parameterization, which may be a limitation in some circumstances. If EvalBlossom is extended to handle an arbitrary blossom parameterization, the result is the generalization known as EvalBlossomProg.

The algorithm is given by:

```
EvalBlossomProg(b[i], . . ., b[i + d], u[1], . . ., u[d], t[i], . . ., t[i + 2d -1])

for k = 0 to (d - 1) do
    for j = 0 to (d - k - 1) do
        Beta = (u[k + 1] - t[i + k + j]) / (t[i + d + j] - t[i + k + j])
        Alpha = 1 - Beta
        b[j] = Alpha b[j] + Beta b[j + 1]
    end
end
return b[0]
```

In this algorithm **b**[i] are the de Boor points, which will be discussed in depth in the topic on The B-Spline Curve. t[i] are the blossom parameterization values.

What Has Been Accomplished in this Topic

A strong base of knowledge concerning the blossom form has been built. This has allowed the investigation of the Bézier curve more deeply, and will motivate the development of B-spline curves.

Topic 5

The B-Spline Curve

Topic 5: The B-Spline Curve

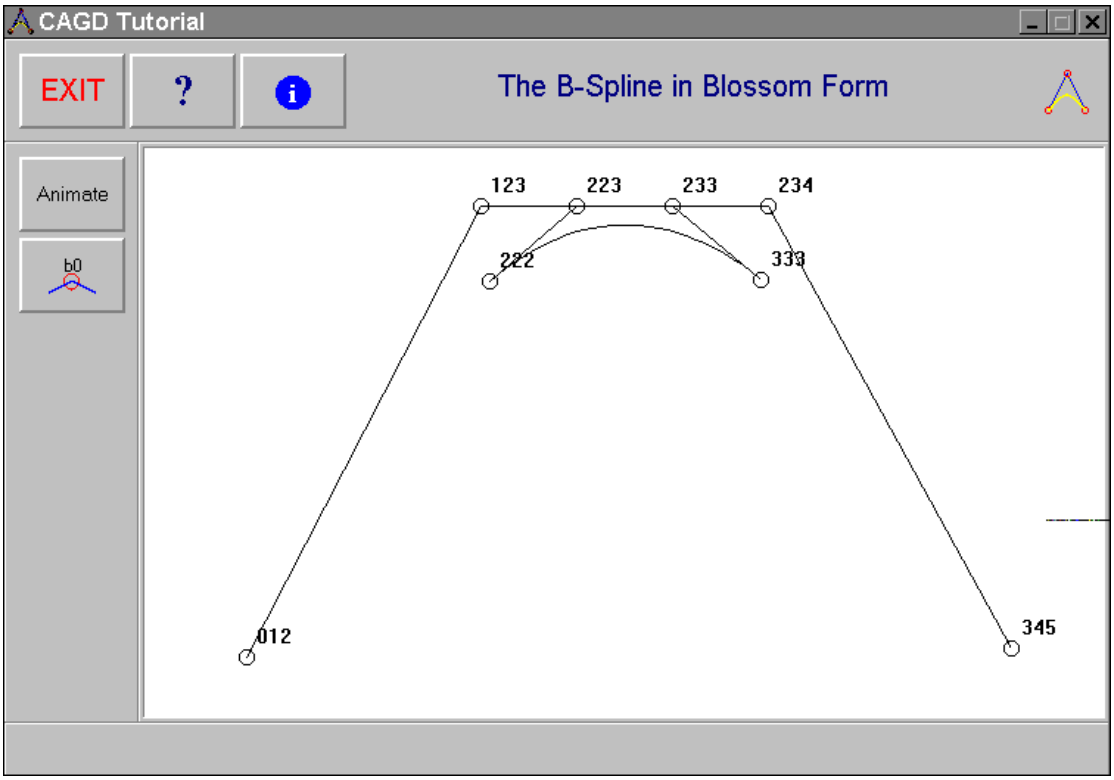
In this topic, you will learn:

- The characteristics of B-spline curves.
- The derivation of B-spline curves from blossoming.
- The differences between Bézier and B-spline curve s.

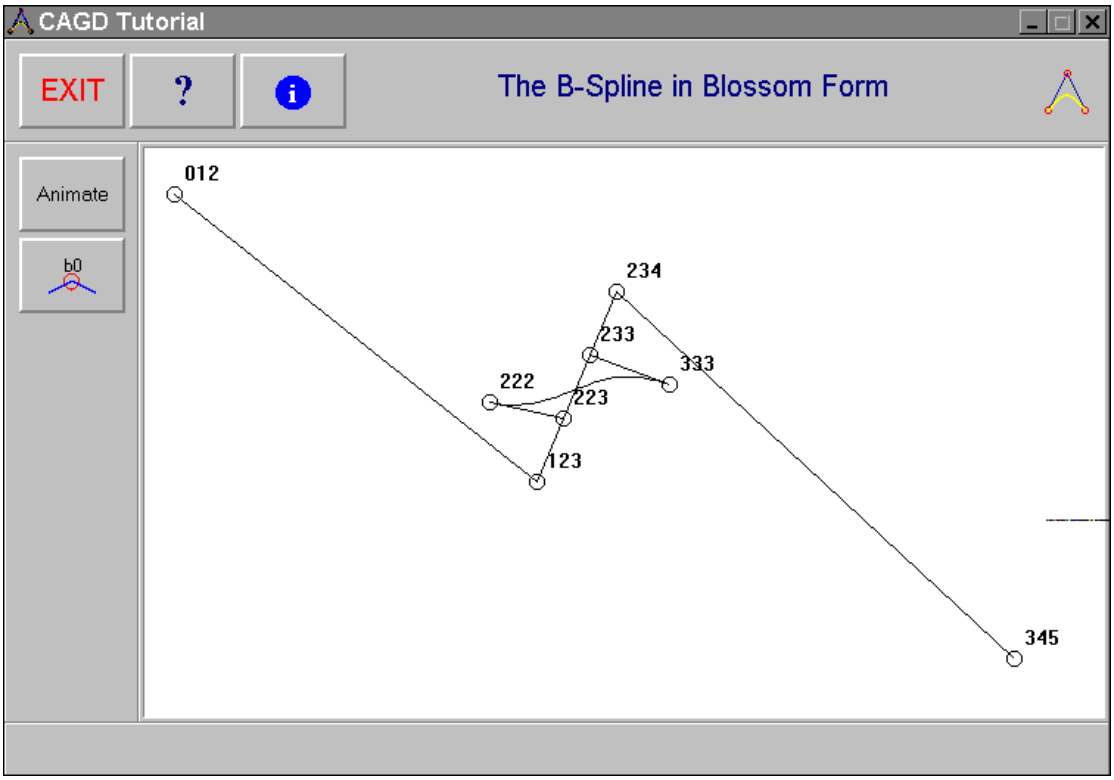
Introduction

The discussion of B-spline curves begins by considering a Bézier curve in blossom form.

In the following figure, a Bézier curve has control points **222**, **223**, **233**, and **333** in the parameter interval 2 to 3. Also shown are the blossom points **012**, **123**, **234**, and **345**.



The next figure shows a new Bézier curve obtained by variation of the Bézier control points. Note how dramatically the blossom points **012**, **123**, **234**, and **345** respond to small changes in the original control points.



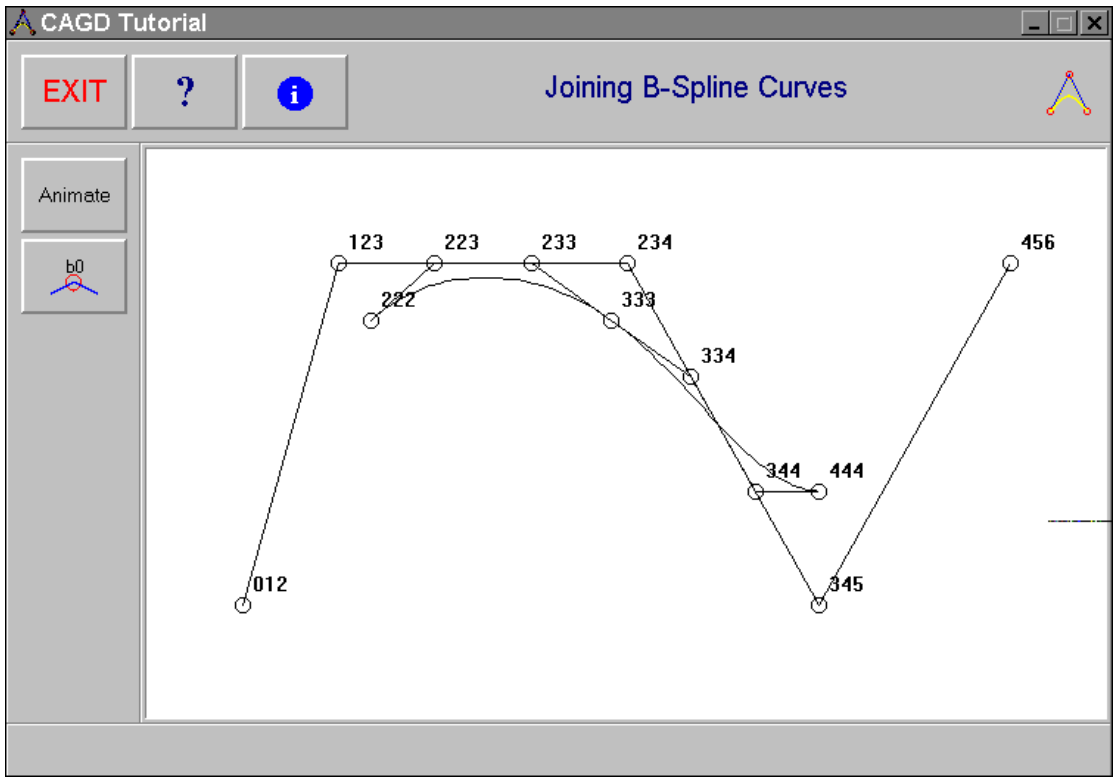
The additional points are computed with the EvalBlossomProg routine, which was introduced in the topic on blossoms. The new points **012**, **123**, **234**, and **345** are called the de Boor[1] points. When a curve is given by these points (instead of, for example, the Bézier control points), it is called a B-spline curve[2].

Superficially it seems that the Bézier form of the curve is better; it interpolates the endpoints, has endpoint tangents, and more closely approximates the control polygon. The advantage of B-spline curves comes when another blossom point is added at an arbitrary point in space: **456**. This point, when taken with **123**, **234**, and **345**, defines a new B-spline curve

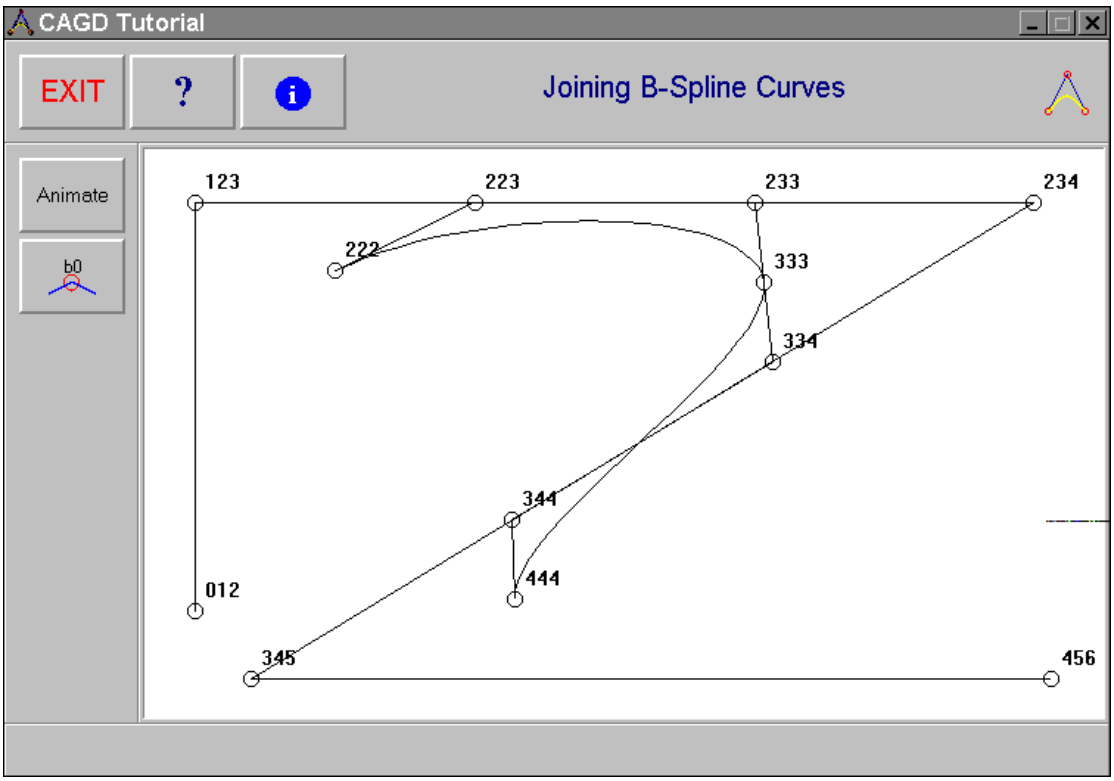
[1] Carl de Boor, an American mathematician, developed the special case of EvalBlossomProg when the u(i) parameters are equal. It evaluates a B-spline.

[2] A spline was originally a strip made of wood or metal that is used to create a smooth curve through a set of points. This was the original method for creating smooth curves.

segment which runs from parameter value 3 to 4, as shown in the following figure:



If the de Boor points are changed, the two curve segments retain their continuity:



Notice that the two curve segments join smoothly without special effort. This is the strength and beauty of B-spline curves. In the cubic case they automatically join with C_2 continuity[3], so long as the curve is not degenerate.

The power of B-spline curves is that one can create with ease a very complex curve that is smoothly connected. They are formed from sequential shifts: **012**, then **123**, then **234**... This is true for B-spline curves of any degree. Points **012**, . . . , **345** can be computed from EvalBlossomProg with **222**, . . . , **333** as input. de Boor point **456** does not come from these input points; rather, it was arbitrarily added. It actually belongs with a different curve, the second curve segment. EvalBlossomProg can also compute a point **456**; this is, of course, different from the point that was added.

What is needed is a method to evaluate the curves' Bézier control points, given the B-spline points. This is not hard when recalling the blossom properties; affine replacement still works and **aaa** is still on the curve.

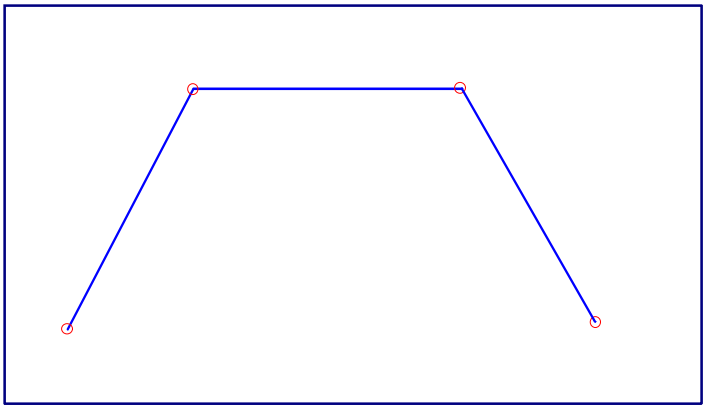
[3] Recall C2 continuity: This means that the second derivatives of the curves are continuous.

These ideas may be used to derive the de Boor algorithm for evaluating B-splines.

The de Boor Algorithm

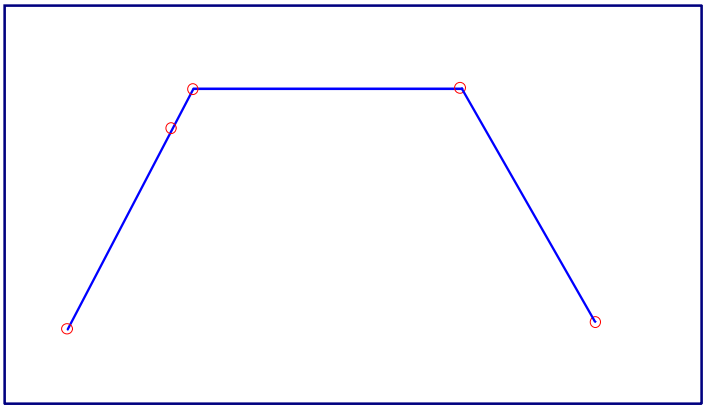
Start with the original set of de Boor points and find the point **2.5**, **2.5**, **2.5**, which is in the curve between parameter values 2 and 3.

Here is the starting set of de Boor points:

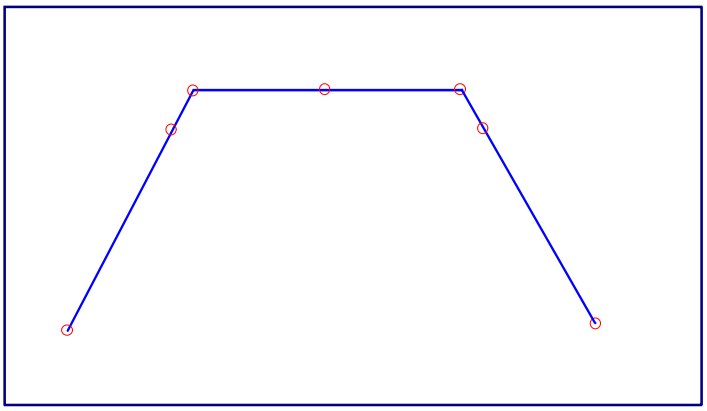


Begin with the first leg of the curve (**012** to **123**). The digits **1** and **2** are common; **0** changes to **3** as we progress along the curve.

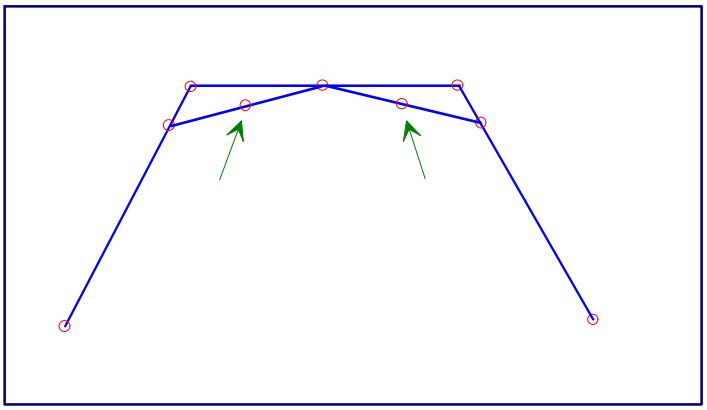
Where is **1**, **2**, **2.5** in this progression? It is linearly interpolated along the line; it is 2.5 / 3 in affine distance along the polygon leg:



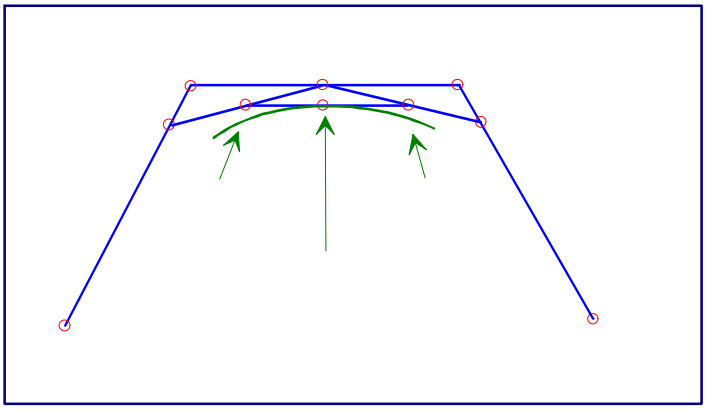
Similarly, **2**, **2.5**, **3** is found on the second leg, and **2.5**, **3**, **4** on the last leg:



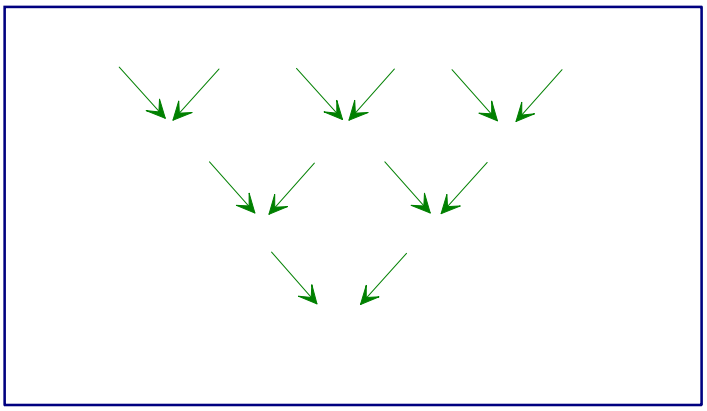
The replacement is repeated to find **2**, **2.5**, **2.5** (as **1** goes to **3**) and **2.5**, **2.5**, **3** (as **2** goes to **3**):



The final step yields **2.5**, **2.5**, **2.5**, a point on the curve:



This process may be described as a systolic array:



The condition to derive the point **1**, **2**, **2.5** is:

$$\mathbf{1,2,3.5} = \frac{(2.5-0)}{(3-0)}\mathbf{1,2,3} + \frac{(3-2.5)}{(3-0)}\mathbf{0,1,2}.$$

As an exercise, be sure you are able to derive the other blossom points.
If a parameter t is used instead of 2.5, then the condition above becomes:

$$\mathbf{1,2,t} = \frac{(t-0)}{(3-0)}\mathbf{1,2,3} + \frac{(3-t)}{(3-0)}\mathbf{0,1,2}.$$

For a different interval, starting at the parameter value i, the condition is:

$$\mathbf{i+1,i+2,t} \quad = \quad \frac{(t-i)}{(i+3-i)}\mathbf{i+1,i+2,i+3} \quad + \quad \frac{(i+3-t)}{(i+3-i)}\mathbf{i,i+1,i+2}.$$

Finally, change the degree to n:

$$\mathbf{i+1,i+2,t} \quad = \quad \frac{(t-i)}{n}\mathbf{i+1,i+2,i+3} \quad + \quad \frac{(i+n-t)}{n}\mathbf{i,i+1,i+2}.$$

This is for a single step in the array. The formal de Boor algorithm is written as:

$$\mathbf{d}_i^K = \frac{(u_{i+n-k} - u)}{(u_{i+n-k} - u_{i-1})}\mathbf{d}_{i-1}^{K-1} + \frac{(u - u_{i-1})}{(u_{i+n-k} - u_{i-1})}\mathbf{d}_i^{K-1}, \qquad (6.1)$$

where the d_i^0 are the de Boor points.

The u values are the parameter intervals; n is the degree of the curve.

This equation is the well-known de Boor algorithm. The de Boor points **d** are associated with the blossom counterparts. The location of each succeeding level of points is made clear by the blossoms.

Summary of B-Splines in Blossom Form

The EvalBlossomProg procedure, in its more general form, gives:

- A method to evaluate B-spline curves.
- A reparameterization method for B-spline curves. This also permits subdivision[4] of the curve.
- A method to transform between B-spline curve segments and Bézier curves.

The B-spline control points in blossom form reveal much about the curve segment:

- The number of digits equals the degree (**012** is a control point for a cubic curve).
- There are sequential shifts of adjacent digits (**012**, **123**, **234**...).
- The last digit of the first point and the first digit of the last point give the valid parameterization interval for the curve segment. In the cubic example, this interval is in the range 2...3.
- Adding one new point to the existing set gives another segment of the B-spline curve. This new segment typically joins the existing curve with degree n - 1 continuity.

---

[4] Subdivision is the process, usually through reparameterization, of separating a single curve segment into two parts.

Basis Functions

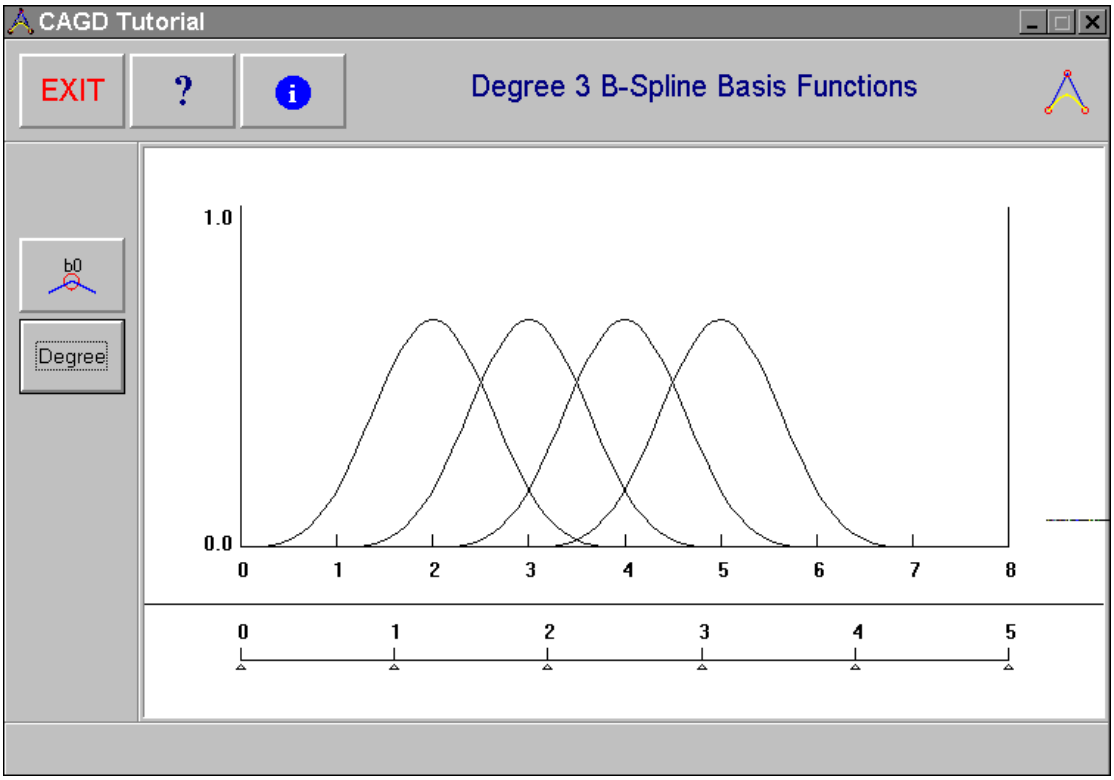
A single B-spline curve segment is defined much like a Bézier curve. It looks like this:

$$\mathbf{d}(t) = \sum_{i=0}^{n} \mathbf{d}_i \, N_i(t), \qquad (6.2)$$

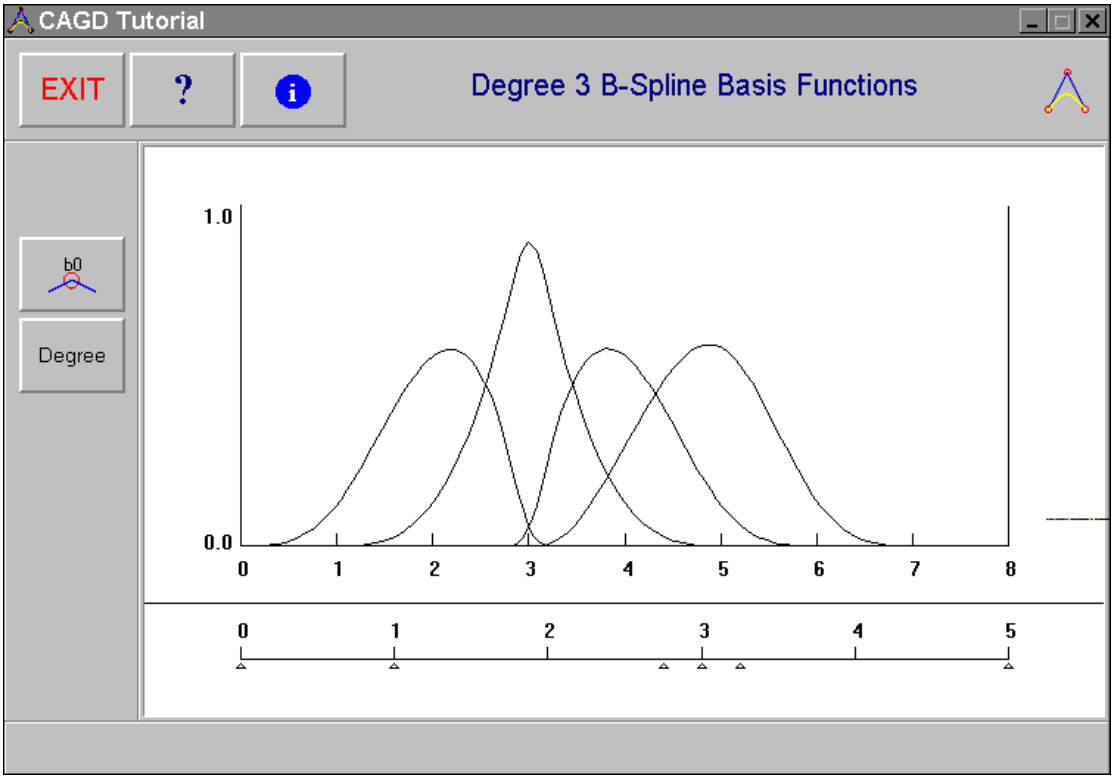
where the **d** points are the de Boor points, the N(t) are the basis functions, and n is the degree of the curve.

The basis functions used here are different from the Bernstein basis functions used by the Bézier curve.

Here are the cubic basis functions:



Here, the figure shows the cubic B-spline basis functions for a non-uniform knot sequence. The knot sequence is shown beneath the basis functions.



The basis functions for the B-spline curves are placed next to each other and overlap in a similar way to the control points. When they are drawn together bell-shaped functions are generated that are zero outside a certain region of "support." It is also clear that they are simply translations of each other, for uniform knots.

It can be seen that (in the cubic case) for any parameter value t, only four basis functions are non-zero; thus only four control points affect the curve at t. If a control point is moved, it influences only a limited portion of the curve.

This locality of influence is known as the local support property. In the same way as the Bernstein polynomials, the B-spline basis functions conform to the partition of unity:

$$\sum N_i(t) = 1$$

This is used to prove that any point on the curve is a convex combination of the de Boor points, that is, it must be in the convex hull of the control points associated with the non-zero basis functions.

What Has Been Accomplished in this Topic

The blossom principle has led naturally to B-splines, a type of curve used extensively in industry. This topic discussed the relationship between B-splines and Bézier curves, how the knot sequence modifies the curve, and how the characteristics of B-splines make them attractive as design tools.

Topic 6

Surfaces

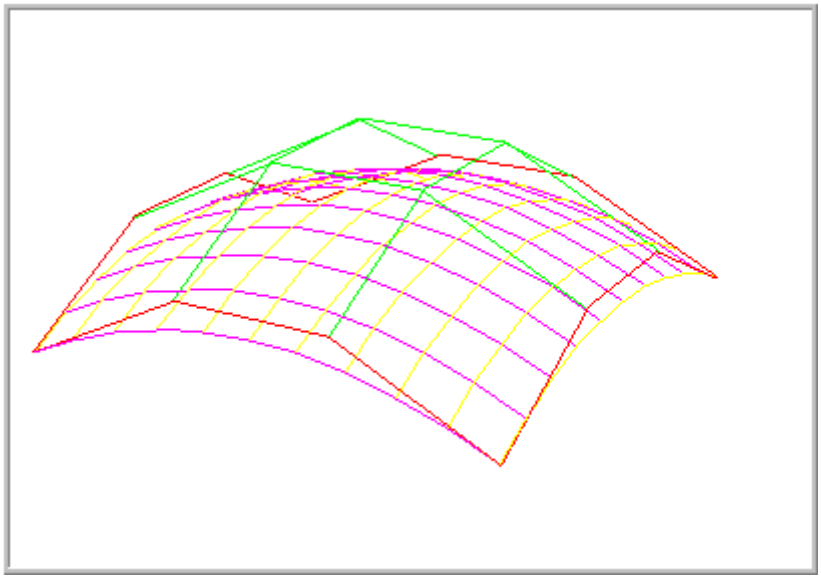
Topic 6: Surfaces

In this topic, you will learn:

- The characteristics of surfaces.
- The use of Bézier surface patches.
- The use of B-spline surface patches.

Introduction to Surfaces

Imagine moving the set of control points of the Bézier curve in three dimensions. As they move in space new curves are generated. If they are moved smoothly then the curves formed create a surface, which may be thought of as a bundle of curves. If each of the control points is moved along a Bézier curve of its own, then a Bézier surface patch is created.



This can be described by changing the control points in the Bézier formula [1] into Bézier curves; thus a surface is defined by:

---

[1] The equation for the standard Bézier curve is:

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

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \mathbf{b}_i(u) B_i(v). \qquad (8.1)$$

Notice there is one parameter for the control curves and one for the "swept" curve. It is convenient to write the control curves as Bézier curves which have the same degree as the original control curves. Given that:

- The ith control curve has control points $\mathbf{b}_{ij}$,

then the surface given in equation 8.1 above can be written as:

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} \mathbf{b}_{ij}\ B_j(u) \right) B_i(v), \qquad (8.2)$$

where m is the degree of the control curves.

Such a surface can be thought of as nesting one set of curves inside another. From this simple characteristic, many properties and operations for surfaces may be derived.

Simple algebra changes equation 8.2 above into:

$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} \mathbf{b}_{ij}\ B_j(u) \right) B_i(v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{b}_{ij}\ B_i(v) B_j(u). \qquad (8.3)$$

That is, even though one curve was swept along the other, there is no preferred direction. The surface patch could have been written as:

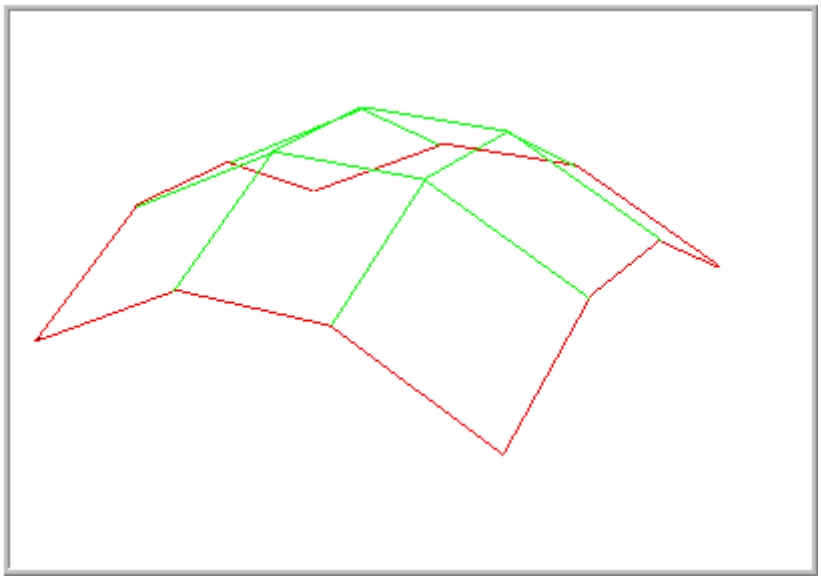
$$\mathbf{s}(u,v) = \sum_{i=0}^{n} \mathbf{b}_j(v) B_j(u) \qquad (8.4)$$

where:

$$\mathbf{b}_j(v) = \sum_{i=0}^{n} \mathbf{b}_{ij}\ B_i(v). \qquad (8.5)$$

The curve is simply swept in the other direction.

The set of control points forms a rectangular control mesh. A 3 by 3 (bicubic) control mesh is shown here:

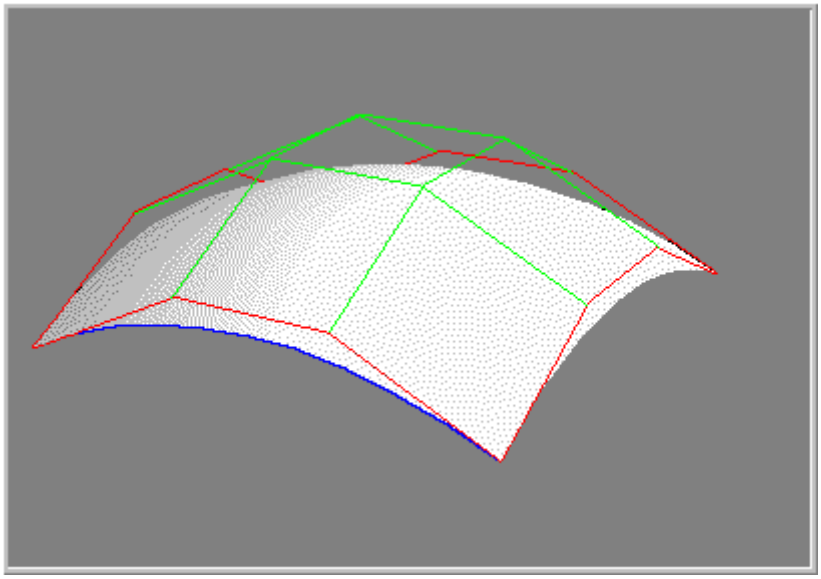


There are 16 control points in the bicubic control mesh. In general there will be (n + 1) by (m + 1) control points. By convention, the "i" index is associated with the "u" parameter, and the "j" index with the "v" parameter. Hence:

$$\mathbf{b}_{i0}, i = 1 \ldots n,$$

gives the Bézier curve:

$$\mathbf{b}(u,0) = \sum_{i=0}^{n} \mathbf{b}_{i0} \, B_i^n(u). \qquad (8.6)$$



Each marginal set of control points defines a Bézier curve (the four border curves) and each of these curves is a boundary of the Bézier surface patch. Such a curve is shown at the forward edge of the patch in the picture above.

Properties of the Bézier Surface Patch

Many of the properties of the Bézier surface are derived directly from those of the Bézier curve, especially those curves that form the boundaries of the patch:

- Endpoint Interpolation

  The Bézier surface patch passes through all four corner control points. Formally, for the bicubic case:

  $$\mathbf{b}(0,0) = \mathbf{b}_{00}; \mathbf{b}(0,1) = \mathbf{b}_{03}; \mathbf{b}(1,0) = \mathbf{b}_{30}; \mathbf{b}(1,1) = \mathbf{b}_{33}.$$

- Tangent Conditions

  The four border curves of the Bézier surface patch are cotangent to the first and last segments of the each border control polygon, at the first and last control points. The normal to the surface patch at each vertex may be found from the cross product of the tangents.

- Convex Hull

  The Bézier surface patch is contained in the convex hull of its control mesh for $0 \leq u \leq 1$ and $0 \leq v \leq 1$.

- Affine Invariance

  The Bézier surface patch is affinely invariant with respect to its control mesh. This means that any linear transformation or translation of the control mesh defines a new patch which is just the transformation or translation of the original patch.

- Variation Diminishing

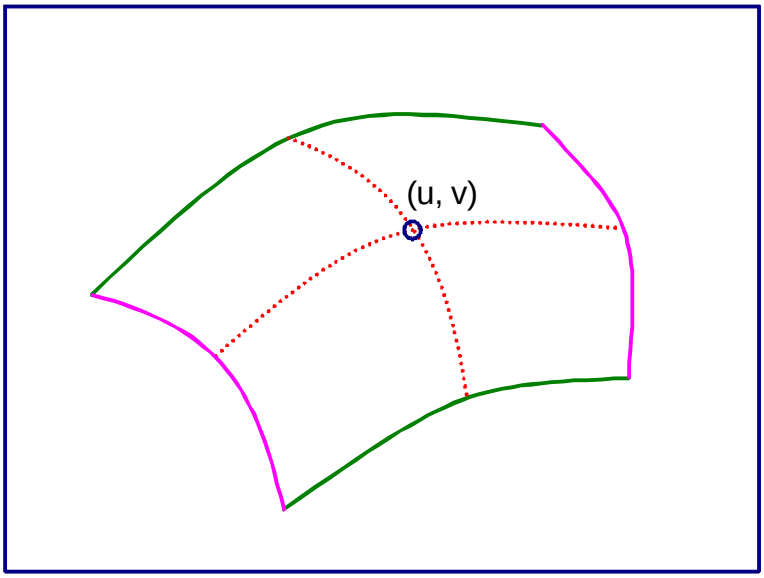
  Although this is difficult to define for surfaces, the control mesh suggests the shape of the patch.

- Planar Precision

  The Bézier surface patch has planar precision: if all the points in the control mesh lie in a plane, the surface patch will lie in the same plane; if all the points in the control mesh form a straight line, the surface is also reduced to a line.

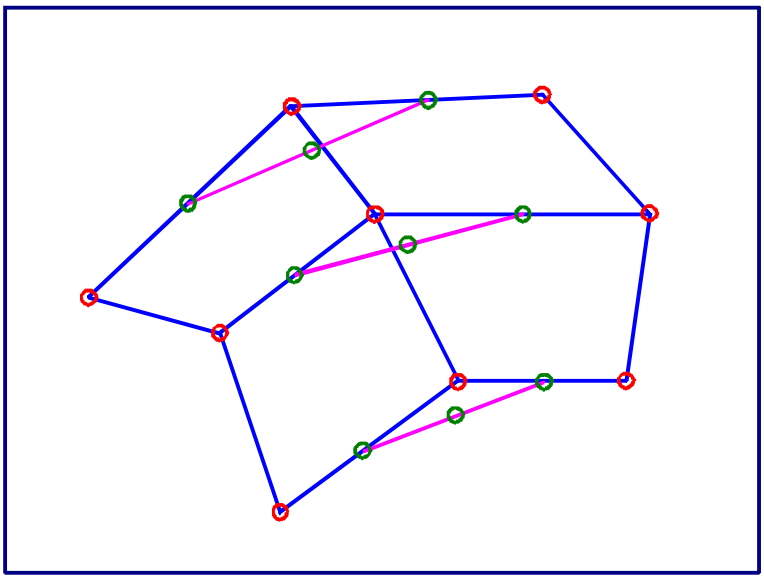
Subdivision of the Bézier Surface Patch

As with the Bézier curve, the de Casteljau algorithm can be applied to subdivide a Bézier surface patch. When a surface patch is subdivided, it yields four subpatches that share a corner at the (u, v) subdivision point.



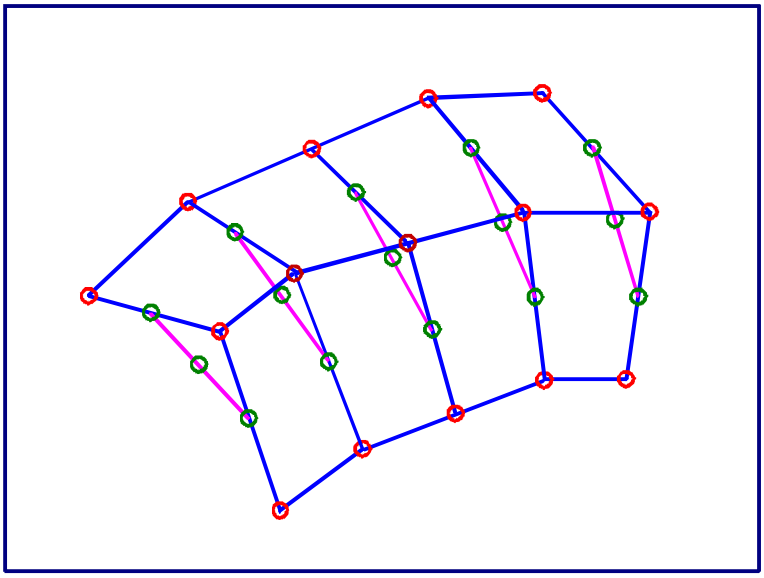
Recall that when a curve was subdivided, the new curve's control points appeared as the legs of a systolic array. In the surface case, subdividing each row of the control mesh produces points of the systolic array for each.

Each point on each leg of every row's systolic array now becomes a control point for a columnar set. A biquadratic case is shown here:

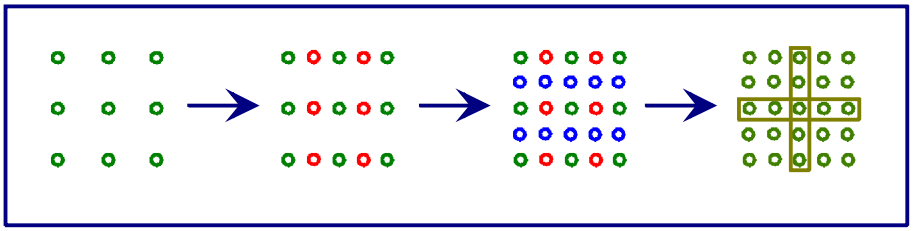


In this case, three points in each row produce five points after subdivision.

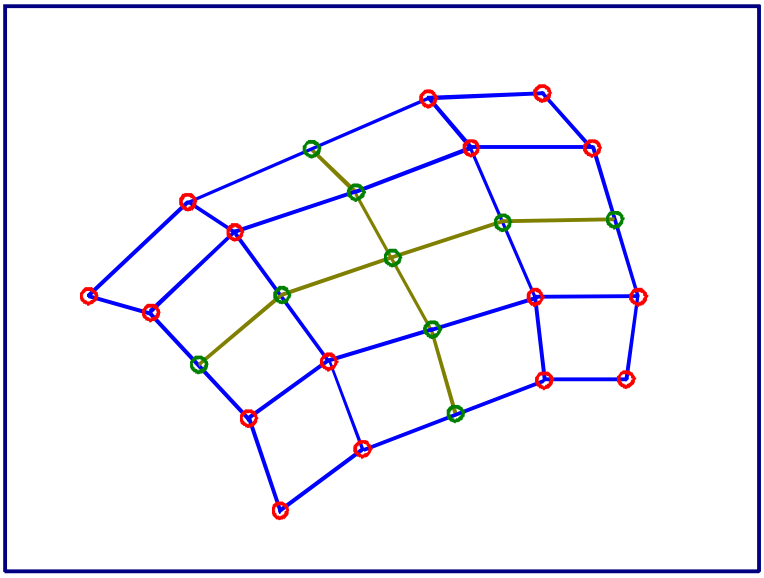
Now consider the points in columns, subdividing the columns with de Casteljau's algorithm The points in the legs of their systolic arrays become the control points of the new subpatches. In the example above, rows with three control points produce five "leg" points, that is, five columns of three points. Each column then produces five control points; so a 3 by 3 grid generates a 5 by 5 grid after subdivision:



The control meshes of the four new patches are produced as follows:



The central row and column of control points are shared by each 3 by 3 subpatch:



The order of the scheme does not matter. Columns may have been taken first, and then rows.

Subdivision is a basic operation of surfaces. Many "divide and conquer" algorithms are based on it. Imagine clipping a surface to a viewing window with the two properties of:

- Convex hull property
- Subdivision

Uniform B-Spline Surfaces

As with the Bézier surface, the B-spline surface is defined as a nested bundle of curves, thus yielding:

$$s(u,v) = \sum_{i=0}^{L+n-1} \sum_{j=0}^{M+m-1} d_{ij} N_i(u) N_j(v), \qquad (8.7)$$

where:

- N_i are the familiar B-spline basis functions;
- n, m are the degrees of the B-splines;
- L, M are the number of segments, so there are L by M patches.

All operations used for B-spline curves carry over to the surface via the nesting scheme, including knot insertion, de Boor's algorithm, and so on.

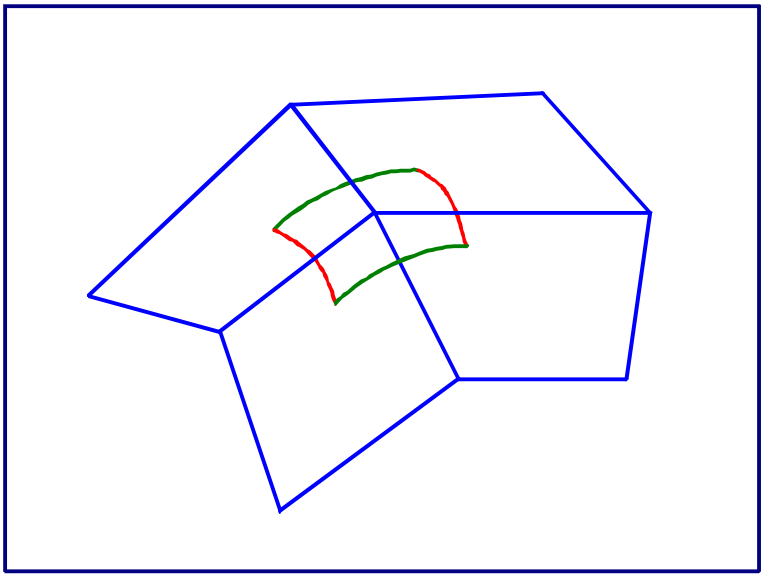
B-spline curves are especially convenient for obtaining continuity between polynomial segments. This convenience is even stronger in the case of B-spline surfaces:

- C^{m-1}, C^{n-1} continuity is maintained between the subpatches.
- B-splines define quilts of patches with corresponding design flexibility.

B-spline curves are more compact at representing a design than Bézier curves. This advantage is "squared" in the case of surfaces.

These advantages are tempered by the fact that operations are typically more efficient on Bézier curves. Conventional wisdom says that it is best to design and represent surfaces as B-splines, and then convert to Bézier form for operations.

The following figure shows a simple B-spline surface patch:



What Has Been Accomplished in this Topic

The majority of industrial design tasks require the creation of three-dimensional products. Bézier and B-spline surface patches supply the tools for such products.

This topic has introduced descriptions for these surface patches, and given their advantages and behavior. Subdivision of the Bézier patch was also presented.

Bibliography

Bibliography

[Bajaj95] C. Bajaj and G. Xu, Modeling with cubic A patches, ACM Trans. on Graphics, 1995.

[Barnhill74] R. Barnhill. Smooth interpolation over triangles. In R. Barnhill and R. Riesenfeld, editors, Computer Aided Geometric Design, pages 45-70, Academic Press, 1974.

[Barnhill82] R. Barnhill. Coons patches. Computers in Industry, 3: 37-43, 1982.

[Barnhill74] R. Barnhill and R. F. Riesenfeld, editors. Computer Aided Geometric Design. Academic Press, 1974. This contains the proceedings of the first conference on CAGD.

[Barsky81] B. Barsky. The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures. Ph.D. thesis, Dept. of Computer Science, University of Utah, 1981.

[Bartels87] R. Bartels, J. Beatty, and B. Barsky, An Introduction of Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaufmann, 1987.

[Blinn78] Simulation of wrinkled surfaces. Computer Graphics, 12(3): 286-292.

[Bloor91] M. Bloor and M. Wilson, Generating blending surfaces with partial differential equations, CAD 25 (1991) pp. 251-256.

[Boehm83] W. Boehm and G. Farin. Concerning subdivision of Bézier triangles. Computer Aided Design, 15(5): 260-261, 1983. Letter to the editor.

[Boehm84] 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.

[Catmull78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer Aided Design, 10(6):350-355, 1978.

[Catmull & Rom74] E. Catmull and R. Rom. A class of local interpolating splines. In R. Barnhilll and R. Riesenfeld, editors, Computer Aided Geometric Design, pages 317-326, Academic Press, 1974.

[Catmull74] E. Catmull. A Subdivision Algorithm for the Computer Display of Curved Surfaces, Ph.D. thesis, Dept. of Computer Science, University of Utah, 1974.

[Charrot84] P. Charrot and J. Gregory. A pentagonal surface patch for computer aided geometric design. Computer Aided Geometric Design, 1(1): 87-94, 1984.

[Chandru89] V. Chandru, D. Dutta, and C. M. Hoffmann. On the geometry of dupin cyclides. The Visual Computer, 5(5): 277-290, 1989.

[Chiyokura83] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. Computer Graphics, 17(3): 289-298, 1983.

[Chiyokura86] H. Chiyokura. Solid Modeling with Designbase, Theory and implementation, Addison Wesley, New York, 1986.

[Coons64] 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. This is the first of Coons' papers on his patch.

[Dahman89] W. Dahman. Smooth Piecewise Quadric Shapes. Math. Methods in CAGD. T. Lyche, L. Schumaker, editors, Academic Press, Boston, 1989.

[deBoor78] C. de Boor. A Practical Guide to Splines. Springer, 1978. This is the classical reference to splines written for the numerical analyst.

[Doo78] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. Computer Aided Design, 10(6): 356-360, 1978.

[Farin93] G. Farin. Curves and Surfaces for Computer Aided Geometric Design. Third Edition. Academic Press, 1993. This is the popular work specifically on parametric curves and surfaces.

[Farin86] G. Farin. Triangular Bernstein-Bézier patches. Computer Aided Geometric Design, 3(2):83-128, 1986.

[Faux79] I. Faux and M. Pratt. Computational Geometry for Design and Manufacture. Ellis Horwood, 1979. This is the first text on the subject, still an interesting read.

[Filip86] D. Filip. Adaptive subdivision algorithms for a set of Bézier triangles. Computer Aided Design, 18(2): 74-78, 1986.

[Foley92] J. Foley and A. Van Dam. Fundamentals of Interactive Computer Graphics. Addison-Wesley, 1992. A popular text book on computer graphics, this text has a few introductory chapters on CAGD.

[Forrest72] A. Forrest. On Coons' and other methods for the representation of curved surfaces. Computer Graphics and Image Processing, 1(4): 341-359, 1972.

[Goldman88] R. Goldman, Urn Models, Approximations and Splines. Journal of Approximation Theory, 54: 1-66, 1988.

[Goldman85] 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.

[Goshtasby93] A. Goshtasby. Design and recovery of 2D and 3D shapes using rational Gaussian curves and surfaces. Int. J. of Computer Vision, 10(3): 233-256, 1993.

[Gregory80] 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.

[Hoffman89] C. Hoffmann. Geometric & Solid Modeling. Morgan Kaufmann, 1989.

[Hosaka84] M. Hosaka and F. Kimura. Non-four-sided patch expressions with control points. Computer Aided Geometric Design, 1(1): 75-86, 1984.

[Hoschek89] J. Hoschek and K. Lasser. Grundlagen der Geometrischen Datenverarbeiturng. B.G. Teubner, Stuttgart, 1989. English translation: Fundamentals of Computer Aided Geometric Design, Jones and Bartlett, publishers. This is a text that is very broad and comprehensive.

[Loop90] C. Loop and T. DeRose. Generalized B-spline surfaces of arbitrary topology. Computer Graphics, 24(4): 347-356, 1990.

[Nasri91] A. Nasri. Boundary-corner control in recursive-subdivision surfaces. Computer Aided Design, 23(6): 405-410, 1991.

[Nielson79] G. Nielson. The side-vertex method for interpolation in triangles. Journal of Approximation Theory, 25: 318-336, 1979.

[Nielson86] G. Nielson. A rectangular nu-spline for interactive surface design. IEEE Computer Graphics and Applications, 6(2): 35-41. 1986.

[Nielson87] G. Nielson. A transfinite, visually continuous, triangular interpolant. In G. Farin, editor, Geometric Modeling: Algorithms and New Trends, pages 235-246, SIAM, Philadelphia, 1987.

[Pavlidis83] T. Pavlidis. Curve fitting with conic splines. ACM Transactions on Graphics, 2(1): 1-31, 1983.

[Plowman95] D. Plowman and P. Charrot. A practical implementation of vertex blend surfaces using an n-sided patch. The Mathematics of Surfaces, VI, G. Mullineux, editor, Oxford University Press, 1995.

[Powell77] M. Powell and M. Sabin. Piecewise quadratic approximations on triangles. ACM Transactions on Mathematical Software, 3: 316-325, 1977.

[Pratt90] M. Pratt. Cyclides in computer aided geometric design. Computer Aided Geometric Design, 7(1-4): 221-242, 1990.

[Rockwood89] A. Rockwood. The displacement method for implicit blending surfaces in solid models. ACM TOGS, 8(4): 279-297, Oct. 1989.

[Sabin76] M. Sabin. The use of piecewise forms for the numerical representation of shape. Ph.D. thesis, Hungarian Academy of Sciences, Budapest, Hungary, 1976.

[Sabin86] M. Sabin. Some negative results in n-sided patches. Computer Aided Design, 18(1): 38-44, 1986.

[Salmon1885] G. Salmon. Modern Higher Algebra. Chelsea, New York, 5th edition, pp. 83-86, 1885. An excellent Victorian text on geometry. Be careful how you name books-notice the date!

[Sarraga90] R. F. Sarraga. Computer modeling of surfaces with arbitrary Shapes. IEEE Computer Graphics and Applications (1990), 67-77.

[Sederberg83] T. Sederberg. Implicit and parametric curves and surfaces for computer aided geometric design. Ph.D. thesis, Purdue University, 1983.

[Sederberg95] T. Sederberg. Implicitization using moving curves and surfaces. Proc. of ACM SIGGRAPH ‘95, August 1995.

[Sederberg85] T. Sederberg. Piecewise algebraic surface patches. CAGD 2, 1985.

[Varady87] T. Varady. 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.

[Varady91] T. Varady. Overlap patches: a new scheme for interpolating curve networks with n-sided regions. CAGD 8 (1991), 7-27.

[Varady95] T. Varady and A. Rockwood. Vertex blending based on the setback split. Math. Methods in CAGD III, M. Daehlen et al, editors, Vanderbilt University Press, 1995.

[Zhao94] Y. Zhao and A. Rockwood. A convolution approach to N-sided patches and vertex blending, designing fair curves and surfaces. N. Sapidis, editor, SIAM, Philadelphia, 1994.
