B.4 Interpolating and Approximating Curves

This section covers many of the basic terms and concepts needed to interpolate values in computer animation. It is not a complete treatise of curves but an overview of the important ones. While many of the terms and concepts discussed are applicable to functions in general, they are presented as they relate to functions having to do with the practical interpolation of points in Euclidean space as typically used in computer animation applications. For more complete discussions of the topics contained here, see, for example, Mortenson [14], Rogers and Adams [18], Farin [4], and Bartels, Beatty, and Barsky [1].

B.4.1 Equations: Some Basic Terms

For present purposes, there are three types of equations: *explicit*, *implicit*, and *parametric*. *Explicit equations* are of the form $y = f(x)$. The explicit form is good for generating points because it generates a value of y for any value of x put into the function. The drawback of the explicit form is that it is dependent on the choice of coordinate axes and it is ambiguous if there is more than one y for a given x (such as $y = \sqrt{x}$, in which an input value of 4 would generate values of either 2 or –2). *Implicit equations* are of the form $f(x, y) = 0$. The implicit form is good for testing to see if a point is on a curve because the coordinates of the point can easily be put into the equation for the curve and checked to see if the equation is satisfied. The drawback of the implicit form is that generating a series of points along a curve is often desired, and implicit forms are not generative. *Parametric equations* are of the form $x = f(t)$, $y = g(t)$. For any given value of t, a point (x, y) is generated. This form is good for generating a sequence of points as ordered values of t are given. The parametric form is also useful because it can be used for multivalued functions of x, which are problematic for explicit equations.

Equations can be classified according to the terms contained in them. Equations that contain only variables raised to a power are *polynomial* equations. If the highest power is one, then the equation is *linear*. If the highest power is two, then the equation is *quadratic*. If the highest power is three, then it is *cubic*. If the equation is not a simple polynomial but rather contains sines, cosines, log, or a variety of other functions, then it is called *transcendental*. In computer graphics, the most commonly encountered type of function is the cubic polynomial.

Continuity refers to how well behaved the curve is in a mathematical sense. For a value arbitrarily close to a x_0 if the function is arbitrarily close to $f(x_0)$, then it has *positional*, or *zeroth-order*, continuity (C^0) at that point. If the slope of the curve (or the first derivative of the function) is continuous, then the function has *tangential*, or *first-order*, continuity (C^1). This is extended to all of the function's

that correspond to the principal axes The rotated vectors are the columns of the equivalent rotation matrix (Equation B 57)

$$\begin{bmatrix} 1-2\ y^2-2\ z^2 & 2\ x\ y-2\ s\ z & 2\ x\ z+2\ s\ y \\ 2\ x\ y+2\ s\ z & 1-2\ x^2-2\ z^2 & 2\ y\ z-2\ s\ x \\ 2\ x\ z-2\ s\ y & 2\ y\ z+2\ s\ x & 1-2\ x^2-2\ y^2 \end{bmatrix} \quad \text{(Eq B.57)}$$

Given a rotation matrix, one can use the definitions for the terms of the matrix in Equation B 57 to solve for the elements of the equivalent unit quaternion The fact that the unit quaternion has a magnitude of one ($s^2 + x^2 + y^2 + z^2 = 1$), makes it easy to see that the diagonal elements sum to $4\ s^2 - 1$ Summing the diagonal elements of the matrix in Equation B 58 results in Equation B 59. The diagonal elements can also be used to solve for the remaining terms (Equation B 60) The square roots of these last equations can be avoided if the off-diagonal elements are used to solve for x, y, and z at the expense of testing for a divide by an s that is equal to zero (in which case Equation B 60 can be used)

$$\begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1\,0} & m_{1,1} & m_{1,2} \\ m_{2\,0} & m_{2,1} & m_{2,2} \end{bmatrix} \quad \text{(Eq. B.58)}$$

$$s = \frac{\sqrt{m_{0,0} + m_{1,1} + m_{2,2} + 1}}{2} \quad \text{(Eq. B.59)}$$

$$m_{0,0} = 1-2\ y^2-2\ z^2 = 1-2\ (y^2+z^2)$$

$$= 1-2\ (1-x^2-s^2) = -1+2\ x^2+2\ s$$

$$x = \sqrt{\frac{m_{0,0}+1-2\ s^2}{2}}$$

$$y = \sqrt{\frac{m_{1,1}+1-2\ s^2}{2}}$$

$$z = \sqrt{\frac{m_{2,2}+1-2\ s^2}{2}} \quad \text{(Eq B.60)}$$

B.4 Interpolating and Approximating Curves

This section covers many of the basic terms and concepts needed to interpolate values in computer animation. It is not a complete treatise of curves but an overview of the important ones. While many of the terms and concepts discussed are applicable to functions in general, they are presented as they relate to functions having to do with the practical interpolation of points in Euclidean space as typically used in computer animation applications. For more complete discussions of the topics contained here, see, for example, Mortenson [14], Rogers and Adams [18], Farin [4], and Bartels, Beatty, and Barsky [1].

B.4.1 Equations: Some Basic Terms

For present purposes, there are three types of equations: *explicit*, *implicit*, and *parametric*. *Explicit equations* are of the form $y = f(x)$. The explicit form is good for generating points because it generates a value of y for any value of x put into the function. The drawback of the explicit form is that it is dependent on the choice of coordinate axes and it is ambiguous if there is more than one y for a given x (such as $y = \sqrt{x}$, in which an input value of 4 would generate values of either 2 or -2). *Implicit equations* are of the form $f(x, y) = 0$. The implicit form is good for testing to see if a point is on a curve because the coordinates of the point can easily be put into the equation for the curve and checked to see if the equation is satisfied. The drawback of the implicit form is that generating a series of points along a curve is often desired, and implicit forms are not generative. *Parametric equations* are of the form $x = f(t)$, $y = g(t)$. For any given value of t, a point (x, y) is generated. This form is good for generating a sequence of points as ordered values of t are given. The parametric form is also useful because it can be used for multivalued functions of x, which are problematic for explicit equations.

Equations can be classified according to the terms contained in them. Equations that contain only variables raised to a power are *polynomial* equations. If the highest power is one, then the equation is *linear*. If the highest power is two, then the equation is *quadratic*. If the highest power is three, then it is *cubic*. If the equation is not a simple polynomial but rather contains sines, cosines, log, or a variety of other functions, then it is called *transcendental*. In computer graphics, the most commonly encountered type of function is the cubic polynomial.

Continuity refers to how well behaved the curve is in a mathematical sense. For a value arbitrarily close to a x_0 if the function is arbitrarily close to $f(x_0)$, then it has *positional*, or *zeroth-order*, continuity (C^0) at that point. If the slope of the curve (or the first derivative of the function) is continuous, then the function has *tangential*, or *first-order*, continuity (C^1). This is extended to all of the function's

derivatives, although for purposes of computer animation the concern is with first-order continuity or, possibly, *second-order*, or *curvature*, continuity (C^2) Polynomials are infinitely continuous

If a curve is pieced together from individual curve segments, one can speak of *piecewise properties*—the properties of the individual pieces For example, a sequence of straight line segments, sometimes called a *polyline* or a *wire*, is piecewise linear A major concern regarding piecewise curves is the continuity conditions at the junctions of the curve segments If one curve segment begins where the previous segment ends, then there is *zeroth-order*, or *positional*, continuity at the junction If the beginning tangent of one curve segment is the same as the ending tangent of the previous curve segment, then there is *first-order*, or *tangential*, continuity at the junction If the beginning curvature of one curve segment is the same as the ending curvature of the previous curve segment, then there is *second-order*, or *curvature*, continuity at the junction Typically, computer animation is not concerned with continuity beyond second order

Sometimes in discussions of the continuity at segment junctions, a distinction is made between *parametric continuity* and *geometric continuity* (e g , [14]) So far the discussion has concerned parametric continuity Geometric continuity is less restrictive First-order parametric continuity, for example, requires that the ending tangent vector of the first segment be the same as the beginning tangent vector of the second First-order geometric continuity, on the other hand, requires that only the direction of the tangents be the same, and it allows the magnitudes of the tangents to be different Similar definitions exist for higher-order geometric continuity One distinction worth mentioning is that parametric continuity is sensitive to the rate at which the parameter varies relative to the length of the curve traced out Geometric continuity is not sensitive to this rate

When a curve is constructed from a set of points and the curve passes through the points, it is said to *interpolate* the points However, if the points are used to control the general shape of the curve, with the curve not necessarily passing through them, then the curve is said to *approximate the points* *Interpolation* is also used generally to refer to all approaches for constructing a curve from a set of points For a given interpolation technique, if the resulting curve is guaranteed to lie within the convex hull of the set of points, then it is said to have the *convex hull property*

B.4.2 Simple Linear Interpolation: Geometric and Algebraic Forms

Simple linear interpolation is given by Equation B 61 and shown in Figure B 23 Notice that the interpolants, $1 - u$ and u, sum to one This property ensures that the interpolating curve (in this case a straight line) falls within the convex hull of

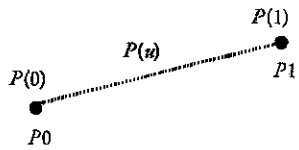


Figure B.23 Linear interpolation

the geometric entities being interpolated (in this simple case the convex hull is the straight line itself)

$$P(u) = (1-u)\ P0 + u\ P1 \qquad \text{(Eq. B.61)}$$

Using more general notation, one can rewrite the equation above as in Equation B.62. Here $F0$ and $F1$ are called *blending* functions. This is referred to as the *geometric* form because the geometric information, in this case $P0$ and $P1$, is explicit in the equation

$$P(u) = F_0(u)\ P0 + F_1(u)\ P1 \qquad \text{(Eq. B.62)}$$

The linear interpolation equation can also be rewritten as in Equation B.63. This form is typical of polynomial equations in which the terms are collected according to coefficients of the variable raised to a power. It is more generally written as Equation B.64. In this case there are only linear terms. This way of expressing the equation is referred to as the *algebraic* form.

$$P(u) = (P1 - P0)\ u + P0 \qquad \text{(Eq. B.63)}$$

$$P(u) = a1\ u + a0 \qquad \text{(Eq. B.64)}$$

Alternatively, both of these forms can be put in a *matrix representation*. The geometric form becomes Equation B.65 and the algebraic form becomes Equation B.66. The geometric form is useful in situations in which the geometric information (the points defining the curve) needs to be frequently updated or replaced. The algebraic form is useful for repeated evaluation of a single curve for different values of the parameter. The fully expanded form is shown in Equation B.67. The curves discussed below can all be written in this form. Of course, depending on the actual curve type, the U (variable), M (coefficient), and B (geometric information) matrices will contain different values.

$$P(u) = \begin{bmatrix} F_0(u) \\ F_1(u) \end{bmatrix} \begin{bmatrix} P0 & P1 \end{bmatrix} = FB^T \qquad \text{(Eq. B.65)}$$

$$P(u) = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} a1 \\ a0 \end{bmatrix} = U^T A \qquad \text{(Eq. B.66)}$$

$$P(u) = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P0 \\ P1 \end{bmatrix} = U^T MB = FB = U^T A \qquad \text{(Eq. B.67)}$$

B.4.3 Parameterization by Arc Length

It should be noted that in general there is not a linear relationship between changes in the parameter u and the distance traveled along a curve (its *arc length*) It happens to be true in the example above concerning a straight line and the parameter u. However, as Mortenson [14] points out, there are other equations that trace out a straight line in space that are fairly convoluted in their relationship between changes in the parameter and distance traveled. For example, consider Equation B.68, which is linear in $P0$ and $P1$. That is, it traces out a straight line in space between $P0$ and $P1$. However, it is nonlinear in u. As a result, the curve is not traced out in a nice monotonic, constant-velocity manner. The nonlinear relationship is evident in most parameterized curves unless special care is taken to ensure constant velocity. (See Chapter 3, "Controlling the Motion Along a Curve.")

$$P(u) = P0 + ((1-u) \quad u+u) \quad (P1 - P0) \qquad \text{(Eq. B.68)}$$

B.4.4 Computing Derivatives

One of the matrix forms for parametric curves, as shown in Equation B.67 for linear interpolation, is $U^T MB$. Parametric curves of any polynomial order can be put into this matrix form. Often, it is useful to compute the derivatives of a parametric curve. This can be done easily by taking the derivative of the U vector. For example, the first two derivatives of a cubic curve, shown in Equation B.69, are easily evaluated for any value of u.

$$P(u) = U^T MB = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} MB$$

$$P'(u) = U'^T MB = \begin{bmatrix} 3 & u^2 & 2 & u & 1 & 0 \end{bmatrix} MB$$

$$P''(u) = U''^T MB = \begin{bmatrix} 6 & u & 2 & 0 & 0 \end{bmatrix} MB \qquad \text{(Eq. B.69)}$$

B.4.5 Hermite Interpolation

Hermite interpolation generates a cubic polynomial from one point to another. In addition to specifying the beginning and ending points (P_i, P_{i+1}), the user needs to supply beginning and ending tangent vectors (P'_i, P'_{i+1}) as well (Figure B.24). The general matrix form for a curve is repeated in Equation B.70, and the Hermite matrices are given in Equation B.71.

$$P(u) = U^T M B \qquad \text{(Eq. B.70)}$$

$$U^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} P_i \\ P_{i+1} \\ P'_i \\ P'_{i+1} \end{bmatrix} \qquad \text{(Eq. B.71)}$$

Continuity between beginning and ending tangent vectors of connected segments is ensured by merely using the ending tangent vector of one segment as the beginning tangent vector of the next. A composite Hermite curve (piecewise cubic with first-order continuity at the junctions) is shown in Figure B.25.

Trying to put a Hermite curve through a large number of points, which requires the user to specify all of the needed tangent vectors, can be a burden. There are several techniques to get around this. One is to enforce second-degree continuity.

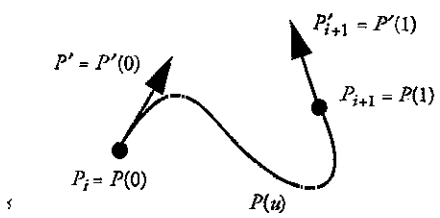


Figure B.24 Hermite interpolation

Figure B.25 Composite Hermite curve

This requirement provides enough constraints so that the user does not have to provide interior tangent vectors; they can be calculated automatically. See Rogers and Adams [18] or Mortenson [14] for alternative formulations. A more common technique is the Catmull-Rom spline.

B.4.6 Catmull-Rom Spline

The Catmull-Rom curve can be viewed as a Hermite curve in which the tangents at the interior control points are automatically generated according to a relatively simple geometric procedure (as opposed to the more involved numerical techniques referred to above). For each interior point, P_i, the tangent at that point, P_i', is computed as one-half the vector from the previous control point, P_{i-1}, to the following control point, P_{i+1} (Equation B.72), as shown in Figure B.26.[1] The matrices for the Catmull-Rom curve in general matrix form are given in Equation B.73. A Catmull-Rom spline is a specific type of cardinal spline.

$$P_i' = (1/2)\ (P_{i+1} - P_{i-1}) \qquad \text{(Eq. B.72)}$$

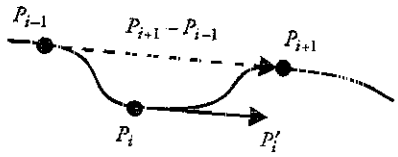


Figure B.26 Catmull-Rom spline

1 Farin [4] describes the Catmull-Rom spline curve in terms of a cubic Bezier curve by defining interior control points. Placement of the interior control points is determined by use of an auxiliary knot vector. With a uniform distance between knot values, the control points are displaced from the point to be interpolated by one-sixth of the vector from the previous interpolated point to the following interpolated point. Tangent vectors are three times the vector from an interior control point to the interpolated point. This results in the Catmull-Rom tangent vector described here.

$$U^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$M = \frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \qquad \text{(Eq. B.73)}$$

For the end conditions, the user can provide tangent vectors at the very beginning and at the very end of the cubic curve. Alternatively, various automatic techniques can be used. For example, the beginning tangent vector can be defined as follows. The vector from the second point (P_1) to the third point (P_2) is subtracted from the second point and used as a virtual point to which the initial tangent is directed. This tangent is computed by Equation B.74. Figure B.27 shows the formation of the initial tangent curve according to the equation, and Figure B.28 shows a curve that uses this technique.

$$P'(0.0) = \frac{1}{2} \cdot (P_1 - (P_2 - P_1) - P_0) = \frac{1}{2} \quad (2 \quad P_1 - P_2 - P_0) \qquad \text{(Eq. B.74)}$$

A drawback of the Catmull-Rom formulation is that an internal tangent vector is not dependent on the position of the internal point relative to its two neighbors. In Figure B.29, all three positions (Q_i, P_i, R_i) for the ith point would have the same tangent vector.

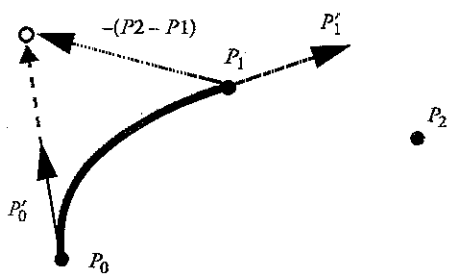


Figure B.27 Automatically forming the initial tangent of a Catmull-Rom spline

Figure B 28 Catmull-Rom spline with end conditions using Equation B 74

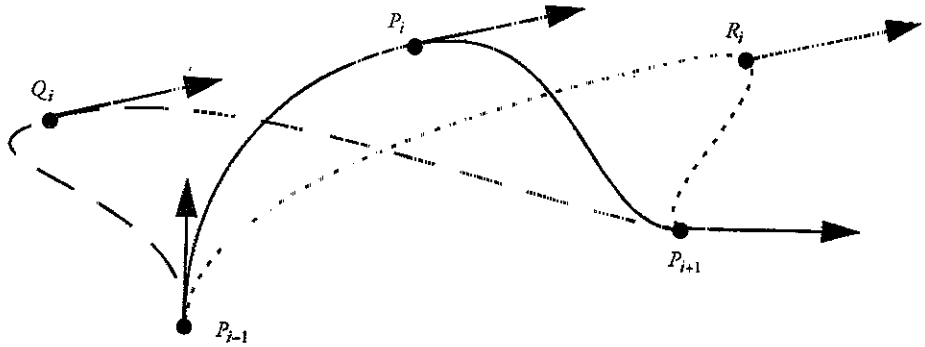


Figure B 29 Three curve segments, (P_{i-1}, P_i, P_{i+1}), (P_{i-1}, Q_i, P_{i+1}), (P_{i-1}, R_i, P_{i+1}), using the standard Catmull-Rom form for computing the internal tangent

An advantage of Catmull-Rom is that the calculation to compute the internal tangent vectors is extremely simple and fast However, for each segment the tangent computation is a one-time-only cost It is then used repeatedly in the computation for each new point in that segment Therefore, it often makes sense to spend a little more time computing more appropriate internal tangent vectors to obtain a better set of points along the segment One alternative is to use a vector perpendicular to the plane that bisects the angle made by $P_{i-1} - P_i$ and $P_{i+1} - P_i$ (Figure B 30). This can be computed easily by adding the normalized vector from P_{i-1} to P_i with the normalized vector from P_i to P_{i+1}

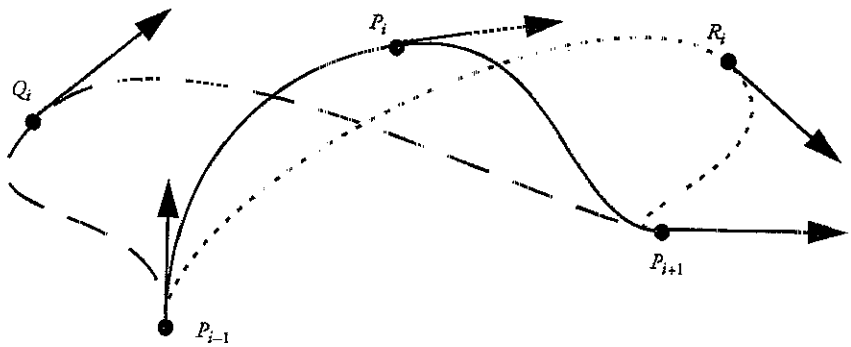


Figure B.30 Three curve segments, (P_{i-1}, P_i, P_{i+1}), (P_{i-1}, Q_i, P_{i+1}), (P_{i-1}, R_i, P_{i+1}) using the perpendicular to the angle bisector for computing the internal tangent

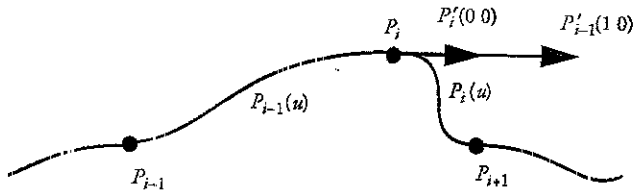


Figure B 31 Interior tangents based on relative segment lengths

Another modification, which can be used with the original Catmull-Rom tangent computation or with the bisector technique above, is to use the relative position of the internal point (P_i) to independently determine the length of the tangent vector for each segment it is associated with. Thus, a point P_i has an ending tangent vector associated with it for the segment from P_{i-1} to P_i as well as a beginning tangent vector associated with it for the segment P_i to P_{i+1}. These tangents have the same direction but different lengths. This relaxes the C^1 continuity of the Catmull-Rom spline and uses G^1 continuity instead. For example, an initial tangent vector at an interior point is determined as the vector from P_{i-1} to P_{i+1}. The ending tangent vector for the segment P_{i-1} to P_i is computed by scaling this initial tangent vector by the ratio of the distance between the points P_i and P_{i-1} to the distance between points P_{i-1} and P_{i+1}. Referring to the segment between P_{i-1} and P_i as $P_{i-1}(u)$ results in Equation B.75. A similar calculation for the beginning tangent vector of the segment between P_i and P_{i+1} results in Equation B.76. These tangents can be seen in Figure B.31. The computational cost of this approach is only a little more than the standard Catmull-Rom spline and seems to give more intuitive results.

$$P'_{i-1}(1\ 0) = \frac{|P_i - P_{i-1}|}{|P_{i+1} - P_{i-1}|}\ (P_{i+1} - P_{i-1}) \qquad \text{(Eq. B.75)}$$

$$P'_i(0\ 0) = \frac{|P_{i+1} - P_i|}{|P_{i+1} - P_{i-1}|}\ (P_{i+1} - P_{i-1}) \qquad \text{(Eq. B.76)}$$

B.4.7 Four-Point Form

Fitting a cubic segment to four points (P_0, P_1, P_2, P_3), assigned to user-specified parametric values (u_0, u_1, u_2, u_3), can be accomplished by setting up the linear system of equations for the points (Equation B.77) and solving for the unknown coefficient matrix. In the case of parametric values of 0, 1/3, 2/3, and 1, the matrix

is given by Equation B 78 However, with this form it is difficult to join segments with C^1 continuity

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} u_0^3 & u_0^2 & u_0 & 1 \\ u_1^3 & u_1^2 & u_1 & 1 \\ u_2^3 & u_2^2 & u_2 & 1 \\ u_3^3 & u_3^2 & u_3 & 1 \end{bmatrix} \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1\,1} & m_{1,2} & m_{1\,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad \text{(Eq. B.77)}$$

$$M = \frac{1}{2} \begin{bmatrix} -9 & 27 & -27 & 9 \\ 18 & -45 & 36 & -9 \\ -11 & 18 & -9 & 2 \\ 2 & 0 & 0 & 0 \end{bmatrix} \quad \text{(Eq. B.78)}$$

B.4.8 Blended Parabolas

Blending overlapping parabolas to define a cubic segment is another approach to interpolating a curve through a set of points In addition, the end conditions are handled by parabolic segments, which is consistent with how the interior segments are defined Blending parabolas results in a formulation that is very similar to Catmull-Rom in that each segment is defined by four points, it is an interpolating curve, and local control is provided Under the assumptions used here for Catmull-Rom and the blended parabolas, the interpolating matrices are identical

For each overlapping triple of points, a parabolic curve is defined by the three points A cubic curve segment is created by linearly interpolating between the two overlapping parabolic segments More specifically, take the first three points, P_0, P_1, P_2, and fit a parabola, $P(u)$, through them using the following constraints: $P(0\ 0) = P_0$, $P(0\ 5) = P_1$, $P(1\ 0) = P_2$ Take the next group of three points, P_1, P_2, P_3, which partially overlap the first set of three points, and fit a parabola, $R(u)$, through them using similar constraints: $R(0\ 0) = P_1$, $R(0\ 5) = P_2$, $R(1\ 0) = P_3$ Between points P_1 and P_2 the two parabolas overlap Reparameterize this region into the range [0 0, 1 0] and linearly blend the two parabolic segments (Figure

Figure B.32 Parabolic blend segment

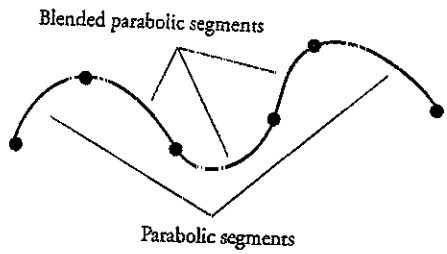


Figure B.33 Multiple parabolic blend segments

B 32). The result can be put in matrix form for a cubic curve using the four points as the geometric information together with the coefficient matrix shown in Equation B 79. To interpolate a list of points, calculate interior segments using this equation. End conditions can be handled by constructing parabolic arcs at the very beginning and very end (Figure B.33)

$$M = \frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \qquad \text{(Eq. B.79)}$$

This form assumes that all points are equally spaced in parametric space. Often it is the case that even spacing is not present. In such cases, relative cord length can be used to estimate parametric values. The derivation is a bit more involved [18], but the final result can still be formed into a 4x4 matrix and used to produce a cubic polynomial in the interior segments.

B.4.9 Bezier Interpolation/Approximation

A cubic Bezier curve is defined by the beginning point and the ending point, which are interpolated, and two interior points, which control the shape of the curve. The cubic Bezier curve is similar to the Hermite form. The Hermite form uses beginning and ending tangent vectors to control the shape of the curve; the Bezier form uses auxiliary control points to define tangent vectors. A cubic curve is defined by four points: P_0, P_1, P_2, P_3. The beginning and ending points of the curve are P_0 and P_3, respectively. The interior control points used to control the shape of the curve and define the beginning and ending tangent vectors are P_1 and P_2. See Figure B.34. The coefficient matrix for a single cubic Bezier curve is shown in Equation B.80. In the cubic case, $P'(0) = 3\ (P_1 - P_0)$ and $P'(1) = 3\ (P_3 - P_2)$

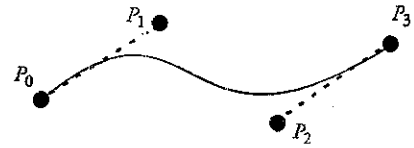


Figure B.34 Cubic Bezier curve segment

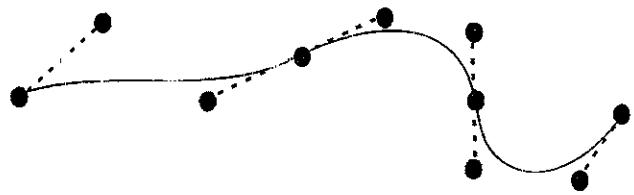


Figure B.35 Composite cubic Bezier curve showing tangents and colinear control points

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{(Eq B.80)}$$

Continuity between adjacent Bezier segments can be controlled by colinearity of the control points on either side of the shared beginning/ending point of the two curve segments where they join (Figure B 35) In addition, the Bezier curve form allows one to define a curve of arbitrary order If three interior control points are used, then the resulting curve will be quartic; if four interior control points are used, then the resulting curve will be quintic See Mortenson [14] for a more complete discussion

B.4.10 De Casteljau Construction of Bezier Curves

The de Casteljau method is a way to geometrically construct a Bezier curve Figure B 36 shows the construction of a point at $u = 1/3$ This method constructs a point u along the way between paired control points (identified by a "1" in Figure B 36) Then points are constructed u along the way between points just previously constructed These new points are marked "2" in Figure B 36 In the cubic case, in which there were four initial points, there are two newly constructed points The point on the curve is constructed by going u along the way between these two points This can be done for any values of u and for any order of curve Higher-order Bezier curves require more iterations to produce the final point on the curve

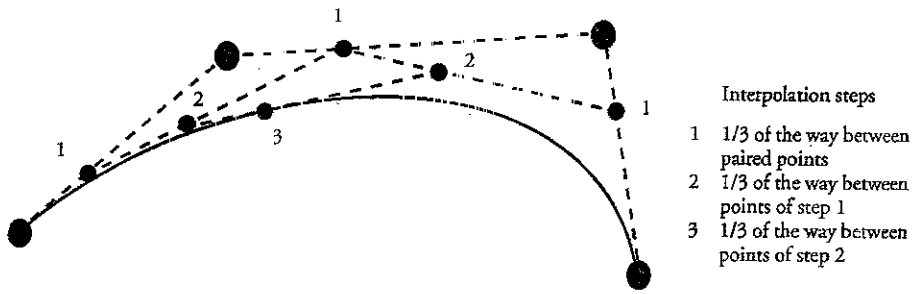


Figure B.36 De Casteljau construction of a point on a cubic Bezier curve

B.4.11 Tension, Continuity, and Bias Control

Often an animator wants better control over the interpolation of key frames than the standard interpolating splines provide. For better control of the shape of an interpolating curve, Kochanek [11] suggests a parameterization of the internal tangent vectors based on the three values tension, continuity, and bias. The three parameters are explained by decomposing each internal tangent vector into an incoming part and an outgoing part. These tangents are referred to as the left and right parts, respectively, and are notated by T_i^L and T_i^R for the tangents at P_i.

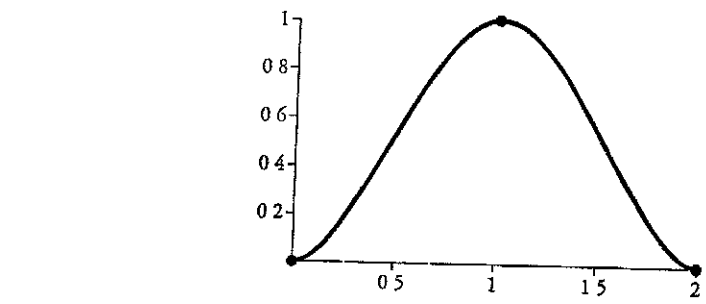
Tension controls the sharpness of the bend of the curve at P_i. It does this by means of a scale factor that changes the length of both the incoming and outgoing tangents at the control point (Equation B.81). In the default case, $t = 0$ and the tangent vector is the average of the two adjacent chords or, equivalently, half of the cord between the two adjacent points, as in the Catmull-Rom spline. As the tension parameter, t, goes to one, the tangents become shorter until they reach zero. Shorter tangents at the control point mean that the curve is pulled closer to a straight line in the neighborhood of the control point. See Figure B.37.

$$T_i^L = T_i^R = (1-t)\ \frac{1}{2}\ ((P_{i+1} - P_i) + (P_i - P_{i-1})) \qquad \text{(Eq B.81)}$$

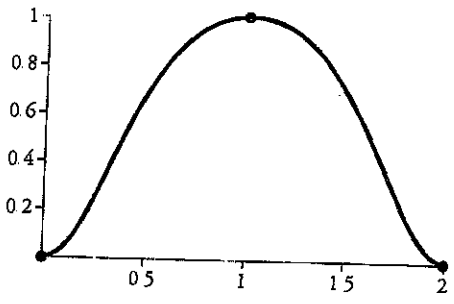
The continuity parameter, c, gives the user control over the continuity of the curve at the control point where the two curve segments join. The incoming (left) and outgoing (right) tangents at a control point are defined symmetrically with respect to the chords on either side of the control point. Assuming default tension, c blends the adjacent chords to form the two tangents, as shown in Equation B.82.

$$T_i^L = \frac{1-c}{2}(P_i - P_{i-1}) + \frac{1+c}{2}(P_{i+1} - P_i)$$

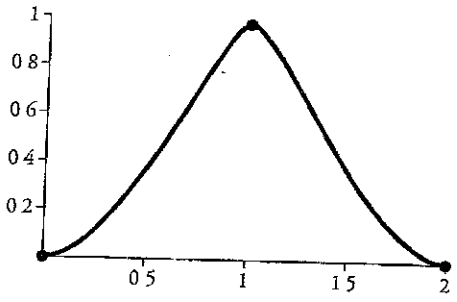
$$T_i^R = \frac{1+c}{2}(P_i - P_{i-1}) + \frac{1-c}{2}(P_{i+1} - P_i) \qquad \text{(Eq B.82)}$$



Default tension, $t = 0$



Low tension, $t = -3/4$



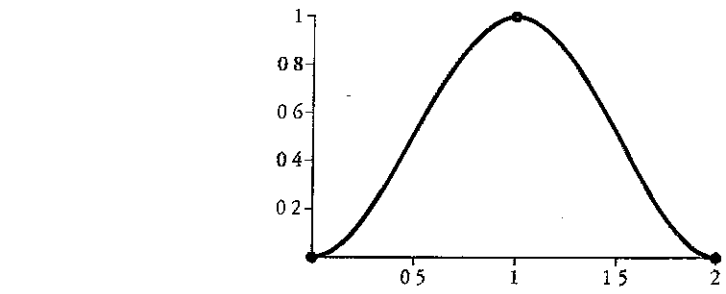
High tension, $t = 3/4$

Figure B 37 The effect of varying the tension parameter

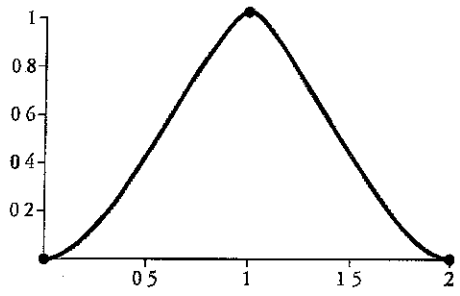
The default value for continuity is $c = 0$, which produces equal left and right tangent vectors, resulting in continuity at the joint As c approaches -1, the left tangent approaches equality with the chord to the left of the control point and the right tangent approaches equality with the chord to the right of the control point As c approaches $+1$, the definitions of the tangents reverse themselves, and the left tangent approaches the right chord and the right tangent approaches the left chord See Figure B 38

Bias, b, defines a common tangent vector, which is a blend between the chord left of the control point and the chord right of the control point (Equation B 83) At the default value ($b = 0$), the tangent is an even blend of these two, resulting in a Catmull-Rom type of internal tangent vector Values of b approaching -1 bias the tangent toward the chord to the left of the control point, while values of b approaching $+1$ bias the tangent toward the chord to the right See Figure B 39

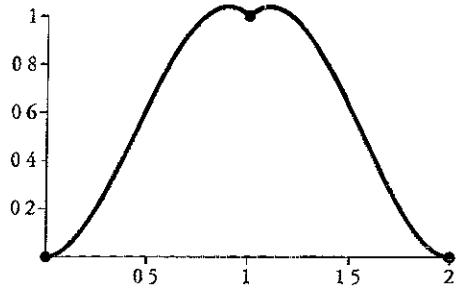
$$T_i^R = T_i^L = \frac{1+b}{2}(P_i - P_{i-1}) + \frac{1-b}{2}(P_{i+1} - P_i) \qquad \text{(Eq B.83)}$$



Default continuity, $c = 0$



Low continuity, $c = -3/4$



High continuity, $c = +3/4$

Figure B.38 The effect of varying the continuity parameter (with default tension)

The three parameters tension, continuity, and bias are combined in Equation B.84

$$T_i^I = \frac{((1-t)(1-c)(1+b))}{2}(P_i - P_{i-1}) + \frac{((1-t)(1+c)(1-b))}{2}(P_{i+1} - P_i)$$

$$T_i^R = \frac{((1-t)(1+c)(1+b))}{2}(P_i - P_{i-1}) + \frac{((1-t)(1-c)(1-b))}{2}(P_{i+1} - P_i) \quad \text{(Eq B.84)}$$

B.4.12 B-Splines

B-splines are the most flexible and useful type of curve, but they are also more difficult to grasp intuitively. The formulation includes Bezier curves as a special case. The formulation for B-spline curves decouples the number of control points from

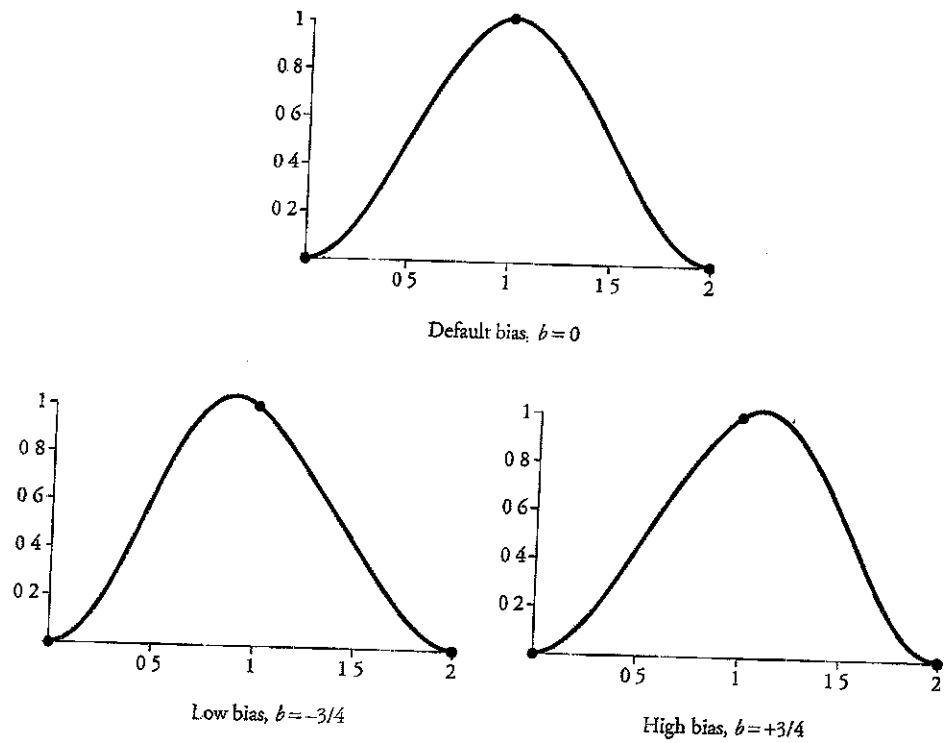
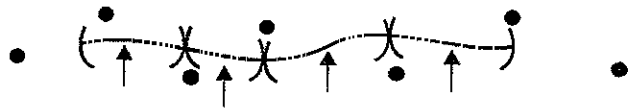


Figure B 39 The effect of varying the bias parameter (with default tension and continuity)

the degree of the resulting polynomial It accomplishes this with additional information contained in the *knot vector.* An example of a *uniform knot vector* is [0, 1, 2, 3, 4, 5, 6, , $n+k-1$], in which the knot values are uniformly spaced apart In this knot vector, n is the number of control points and k is the degree of the B-spline curve The parametric value varies between the first and last values of the knot vector The knot vector establishes a relationship between the parametric value and the control points With replication of values in the knot vector, the curve can be drawn closer to a particular control point up to the point where the curve actually passes through the control point

A particularly simple, yet useful, type of B-spline curve is a uniform cubic B-spline curve It is defined using four control points over the interval zero to one (Equation B 85) A compound curve is generated from an arbitrary number of control points by constructing a curve segment from each four-tuple of adjacent control points: (P_i, P_{i+1}, P_{i+2}, P_{i+3}) for $i = 1, 2,$, $n-3$, where n is the total number of control points (Figure B 40) Each section of the curve is generated by



Segments of the curve defined by different sets of four points

Figure B.40 Compound cubic B-spline curve

multiplying the same 4x4 matrix by four adjacent control points with an interpolating parameter between zero and one. In this case, none of the control points is interpolated

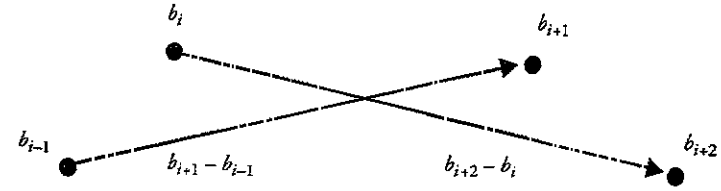
$$P(u) = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix} \quad \text{(Eq B.85)}$$

NURBS, *Nonuniform rational B-splines*, are even more flexible than basic B-splines. NURBS allow for exact representation of circular arcs, whereas Bezier and nonrational B-splines do not. This is often important in modeling, but for purposes of animation, the basic periodic, uniform cubic B-spline is usually sufficient.

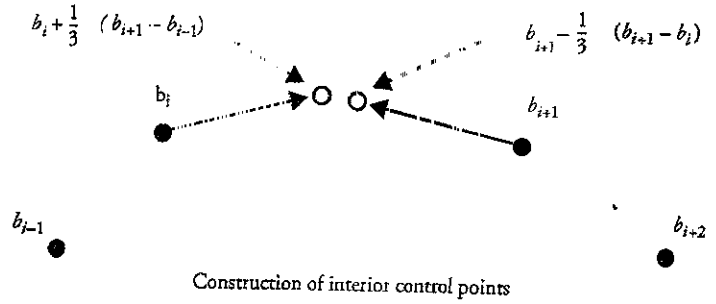
B.4.13 Fitting Curves to a Given Set of Points

Sometimes it is desirable to interpolate a set of points using a Bezier formulation. The points to be interpolated can be designated as the endpoints of the Bezier curve segments, and the interior control points can be constructed by forming tangent vectors at the vertices, as with the Catmull-Rom formulation. The interior control points can be constructed by displacing the control points along the tangent lines just formed. For example, for the segment between given points b_i and b_{i+1}, the first control point for the segment, c_i^1, can be positioned at $b_i + 1/3\ (b_{i+1} - b_{i-1})$. The second control point for the segment, c_i^2, can be positioned at $b_{i+1} - 1/3\ (b_{i+2} - b_i)$. See Figure B.41.

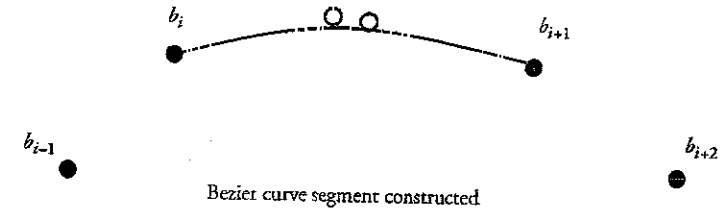
Other methods exist. Farin [4] presents a more general method of constructing the Bezier curve and, from that, constructing the B-spline control points. Both Farin [4] and Rogers and Adams [18] present a method of constructing a composite Hermite curve through a set of points that automatically calculates internal tangent vectors by assuming second-order continuity at the segment joints.



Four initial points with vectors drawn between pairs of points adjacent to interior points (e.g., b_{i-1} and b_{i+1} are adjacent to b_i)



Construction of interior control points



Bezier curve segment constructed

Figure B.41 Constructing a Bezier segment that interpolates points

B.5 Randomness

Introducing controlled randomness in both modeling and animation can often produce more interesting, realistic, natural-looking imagery. The use of noise and turbulence functions are often used in textures but also can be used in modeling natural phenomena such as smoke and clouds. The code for noise and turbulence that follows is from Peachey's chapter in Ebert [3]. Random perturbations are also

useful in human-figure animation to make the motion less "robotic" looking There are various algorithms proposed in the literature for generating random numbers; Gasch's [6] is presented at the end of this section

B.5.1 Noise

The *noise* function uses a table of pseudorandom numbers between −1 and +1 to represent the integer lattice values The table is created by *valueTableInit* the first time that *noise* is called Lattice coordinates are used to index into a table of pseudorandom numbers A simple function of the coordinates, such as their sum, is used to compute the index However, this can result in unwanted patterns To help avoid these artifacts, a table of random permutation values is used to modify the index before it is used A four-point spline is used to interpolate among the lattice pseudorandom numbers (*FPspline*)

```
#define TABSIZE              256
#define TABMASK              (TABSIZE-1)
#define PERM(x)              perm[(x)&TABMASK]
#define INDEX(ix,iy,iz)      PERM((ix)+PERM((iy)+PERM(iz)))
#define FLOOR(x)             (int)(x)

/* PERMUTATION TABLE */
static unsigned char perm[TABSIZE] = {
225, 155, 210, 108, 175, 199, 221, 144, 203, 116, 70, 213, 69, 158, 33,
252, 5, 82, 173, 133, 222, 139, 174, 27, 9, 71, 90, 246, 75, 130, 91,
191, 169, 138, 2, 151, 194, 235, 81, 7, 25, 113, 228, 159, 205, 253,
134, 142, 248, 65, 224, 217, 22, 121, 229, 63, 89, 103, 96, 104, 156,
17, 201, 129, 36, 8, 165, 110, 237, 117, 231, 56, 132, 211, 152, 20,
181, 111, 239, 218, 170, 163, 51, 172, 157, 47, 80, 212, 176, 250, 87,
49, 99, 242, 136, 189, 162, 115, 44, 43, 124, 94, 150, 16, 141, 247, 32,
10, 198, 223, 255, 72, 53, 131, 84, 57, 220, 197, 58, 50, 208, 11, 241,
28, 3, 192, 62, 202, 18, 215, 153, 24, 76, 41, 15, 179, 39, 46, 55, 6,
128, 167, 23, 188, 106, 34, 187, 140, 164, 73, 112, 182, 244, 195, 227,
13, 35, 77, 196, 185, 26, 200, 226, 119, 31, 123, 168, 125, 249, 68,
183, 230, 177, 135, 160, 180, 12, 1, 243, 148, 102, 166, 38, 238, 251,
37, 240, 126, 64, 74, 161, 40, 184, 149, 171, 178, 101, 66, 29, 59, 146,
61, 254, 107, 42, 86, 154, 4, 236, 232, 120, 21, 233, 209, 45, 98, 193,
114, 78, 19, 206, 14, 118, 127, 48, 79, 147, 85, 30, 207, 219, 54, 88,
234, 190, 122, 95, 67, 143, 109, 137, 214, 145, 93, 92, 100, 245, 0,
216, 186, 60, 83, 105, 97, 204, 52
};
```