TechNote TN-06-001

Catmull-Rom Splines

Jim Armstrong
Singularity
January 2006

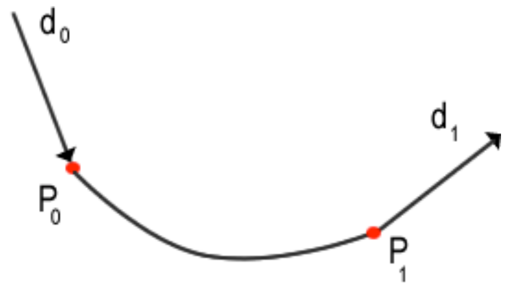
This is the fifth in a series of TechNotes on the subject of applied curve mathematics in Adobe Flash^{TM}. Each TechNote provides the mathematical foundation for a set of Actionscript examples.

Interpolating Splines and Animation

An important consideration in computer-generated tweening is smooth motion between irregularly spaced keyframes. Cubic splines are often an ideal solution. A natural cubic spline produces a C^2-continuous interpolation, although the parametric version is computationally expensive. Often, a C^1-continuous curve is adequate. Catmull-Rom splines, as often referred to in both online and printed literature, are actually a specific instance of a family of splines derived by Catmull and Rom [1]. These splines exhibit C^1 continuity and have a simple piecewise construction.

Relationship to Hermite Interpolation

It seems as if almost every cubic curve construction bears some resemblance to Hermite curves. Consider the case of fitting a cubic curve between two points. In addition to passing through the two points, two additional constraints are required to define the curve. Suppose derivative values are specified at each endpoint, as illustrated in the following diagram.



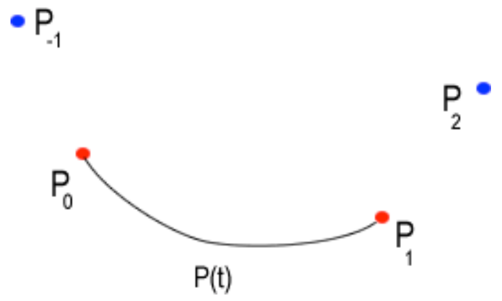
Cubic curve between two control points

This situation is very similar to the Hermite curve except that the above curve does not extend beyond the control points. As the tangents influence the shape of the curve, tangents may either be explicitly provided or inferred from additional control points.

Single-Curve Construction

Consider the derivation of a single curve, $P(t)$, that interpolates the points P_0 and P_1 above.

Suppose that additional control points, P_{-1} and P_2 are specified as shown below.



P(t) is a cubic curve with the conditions that *t* = 0 yields P_0 and *t* = 1 yields P_1. The auxiliary points P_{-1} and P_2 are used to adjust the shape of the curve by implicitly defining tangents,

$$D_0 = \alpha(P_1 - P_{-1})$$
$$D_2 = \alpha(P_2 - P_0)$$

where α is between 0 and 1. This definition makes the tangent at each endpoint parallel to the chord between adjacent control points. The general equation of the curve is

$$P(t) = at^3 + bt^2 + ct + d \qquad [1]$$

The coefficients can be determined from the geometric constraints provided by endpoint and tangent specifications. The process is similar to the derivation of Hermite interpolation,

$$\begin{aligned} P(0) &= P_0 \\ P(1) &= P_1 \\ P'(0) &= \alpha(P_1 - P_{-1}) \\ P'(1) &= \alpha(P_2 - P_0) \end{aligned} \qquad [2]$$

Since $P'(t) = 3at^2 + 2bt + c$

$$\begin{aligned} P(0) &= d \\ P(1) &= a + b + c + d \\ P'(0) &= c \\ P'(1) &= 3a + 2b + c \end{aligned} \qquad [3]$$

If $P'(0) = S_0$ and $P'(1) = S_1$, equation [3] yields,

$$\begin{bmatrix} P_0 \\ P_1 \\ S_0 \\ S_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad [4a]$$

From equation [2],

$$\begin{bmatrix} P_0 \\ P_1 \\ S_0 \\ S_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\alpha & 0 & \alpha & 0 \\ 0 & -\alpha & 0 & \alpha \end{bmatrix} \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix} \qquad [4b]$$

Equations [4a] and [4b] together imply

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\alpha & 0 & \alpha & 0 \\ 0 & -\alpha & 0 & \alpha \end{bmatrix} \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix}$$

[4c]

$$\Rightarrow \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\alpha & 0 & \alpha & 0 \\ 0 & -\alpha & 0 & \alpha \end{bmatrix}$$

Every now and then, it is useful to work one of these out in full. If

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

then $MM^{-1} = I$. Let $M^{-1} = C = c_{ij}$ from which

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [5a]$$

Although 16 equations in 16 unknowns seem intimidating by hand, half the equations are trivial.

$$\begin{aligned} c_{41} &= 0 \\ c_{42} = c_{43} = c_{44} &= 0 \\ c_{31} = c_{32} = c_{34} &= 0 \\ c_{33} &= 1 \end{aligned}$$

The remaining equations are

$$\begin{aligned} c_{11} + c_{21} + c_{31} + c_{41} &= 0 \\ c_{12} + c_{22} + c_{32} + c_{42} &= 1 \\ c_{13} + c_{23} + c_{33} + c_{43} &= 0 \\ c_{14} + c_{24} + c_{34} + c_{44} &= 0 \\ 3c_{11} + 2c_{21} + c_{31} &= 0 \\ 3c_{12} + 2c_{22} + c_{32} &= 0 \\ 3c_{13} + 2c_{23} + c_{33} &= 0 \\ 3c_{14} + 2c_{24} + c_{34} &= 1 \end{aligned}$$

Cranking through the algebra yields

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

$$\Rightarrow \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\alpha & 0 & \alpha & 0 \\ 0 & -\alpha & 0 & \alpha \end{bmatrix} = \begin{bmatrix} -\alpha & 2-\alpha & \alpha-2 & \alpha \\ 2\alpha & \alpha-3 & 3-2\alpha & -\alpha \\ -\alpha & 0 & \alpha & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad [5b]$$

Equation [5b] represents the basis matrix for the so-called cardinal splines. The parameter, α, represents the spline's 'tension'. As the tension parameter approaches 1, the bend at each knot is less, as if the spline was a rope that was being tightened while still passing through all the knots.

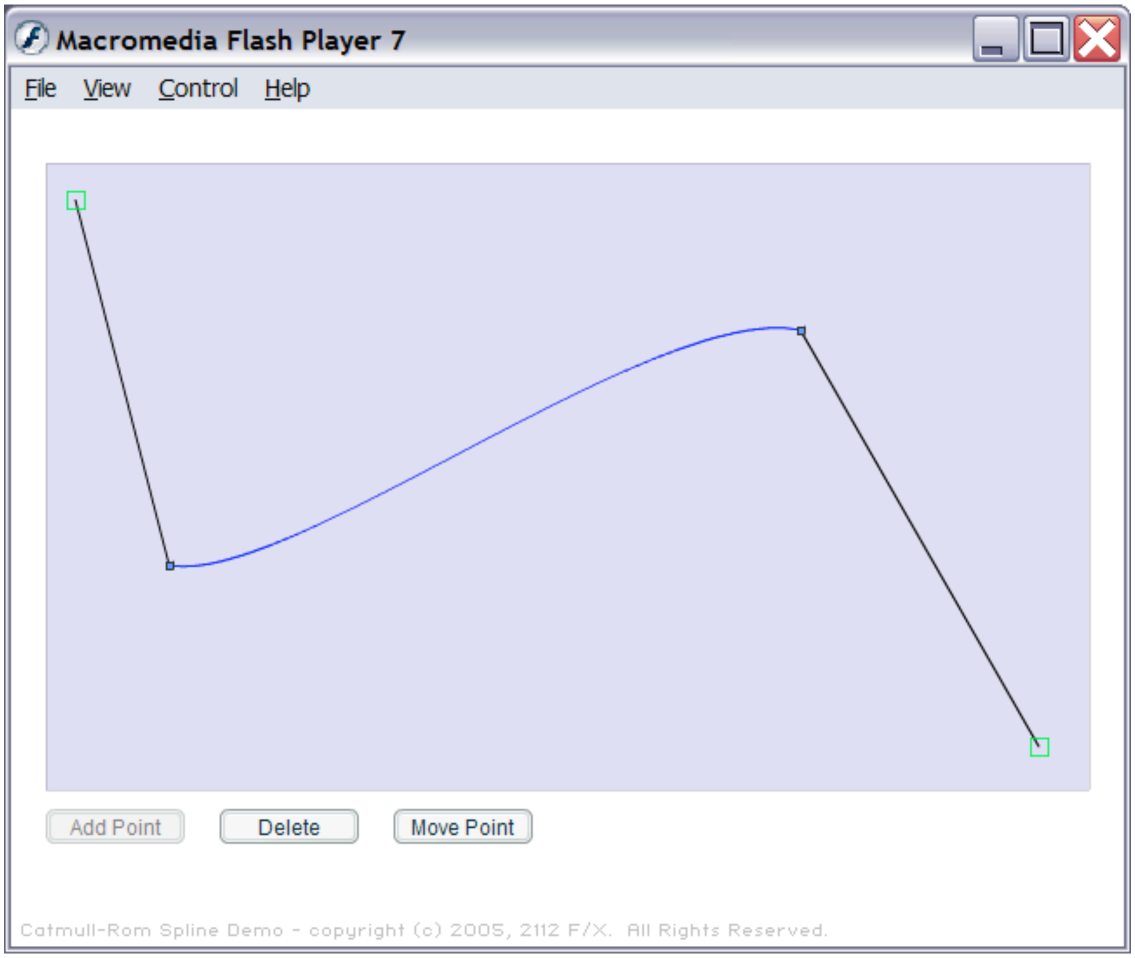
A tension value of ½ is commonly used to represent the Catmull-Rom spline. Substituting into [5b] yields the Catmull-Rom basis

$$B_c = \begin{bmatrix} -1/2 & 3/2 & -3/2 & 1/2 \\ 2/2 & -5/2 & 4/2 & -1/2 \\ -1/2 & 0 & 1/2 & 0 \\ 0 & 2/2 & 0 & 0 \end{bmatrix} = 1/2 \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \qquad [6]$$

Substituting [6] into the standard matrix equation for a cubic curve yields

$$P(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix} \qquad [7]$$

Single Curve Demo

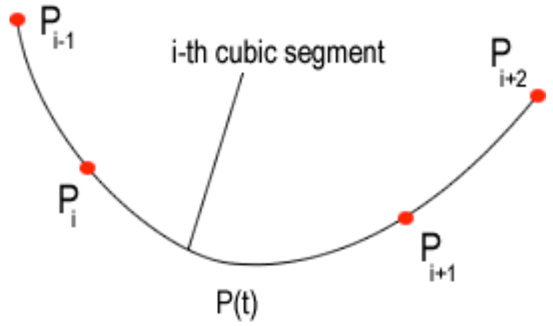

Macromedia Flash Player 7
File View Control Help
Add Point
Delete
Move Point
Catmull-Rom Spline Demo - copyright (c) 2005, 2112 F/X. All Rights Reserved.

This demo illustrates the use of the *CRBasic* class (Flash source no longer online). This class generates a single cubic curve from P_0 to P_1 . The auxiliary control points, P_{-1} and P_2 are interactively generated using tangent handles.

The demo allows two control points to be added to the drawing area. Tangent handles are automatically generated. Drag the handles to new locations to study the effect on the curve. Load the *CRBasic* class and study the __*coef()* method to see the direct application of equation [7].

Interpolating Multiple Points and Parameterization

Once the generation of a single cubic curve is understood, the next step is to apply the process in a piecewise manner to fit a number of datapoints. The *i*-th curve segment is generated between points P_i and P_{i+1} using the sequence of points, P_{i-1}, P_i, P_{i+1}, and P_{i+2} , as illustrated below.



As each new segment is generated, the slopes match at the interior control points, providing C^1 continuity. There are two new issues, however, in generating the complete curve.

Auxiliary Control Points

If the curve is to interpolate all user-supplied knots, additional control points must be supplied at each end of the curve. There are several approaches for resolving this issue. The user may wish interactive control over setting the points. If the spline is to be used in animation, it is valuable to have the control points automatically generated. Some implementations duplicate the end knots. This can cause kinks in the generated curve.

Another method is reflection. For example, the segment $P_1 - P_0$ is reflected about P_0 to generate P_{-1}. While this ensures the curve is 'moving' in an intuitive direction at each endpoint, it can sometimes result in 'pinching', as will be seen in the examples.

The *CatmullRom* class allows the user to either explicitly specify auxiliary control points or have them automatically computed. For illustrative purposes, the latter option is performed with a simple reflection. Either mode can be selected through a setter function, i.e.

```
var __myCurve:CatmullRom = new CatmullRom();
__myCurve.tangent = CatmullRom.EXPLICIT;
```

or

```
__myCurve.tangent = CatmullRom.AUTO;
```

The default tangent mode is AUTO.

Parameterization

The curve is piecewise cubic. Global parameter values of t between 0 and 1 must be mapped to the appropriate segment and a local parameter value between 0 and 1. For purposes of this discussion, t is used for the normal (global) curve parameter and t^* is used for the local parameter. If there are a total of N knots, then

$$t = 0 \Rightarrow t^* = 0, \quad P(t) = P_0$$
$$t = 1 => t^* = 1, \quad P(t) = P_{N-1}$$

are the trivial mappings. As the user varies t between 0 and 1, the appropriate segment index, i, and local parameter value, t^* must be computed. The specific computation determines how the curve is parameterized.

A common parameterization is chord-length. The cumulative distance between each knot is divided by the total length to produce a parameter that varies from 0 to 1. The user-specified value of t is compared segment-by-segment to determine the appropriate curve segment. The parameter value is mapped into a local range of [0,1] in order to be evaluated by equation [7].

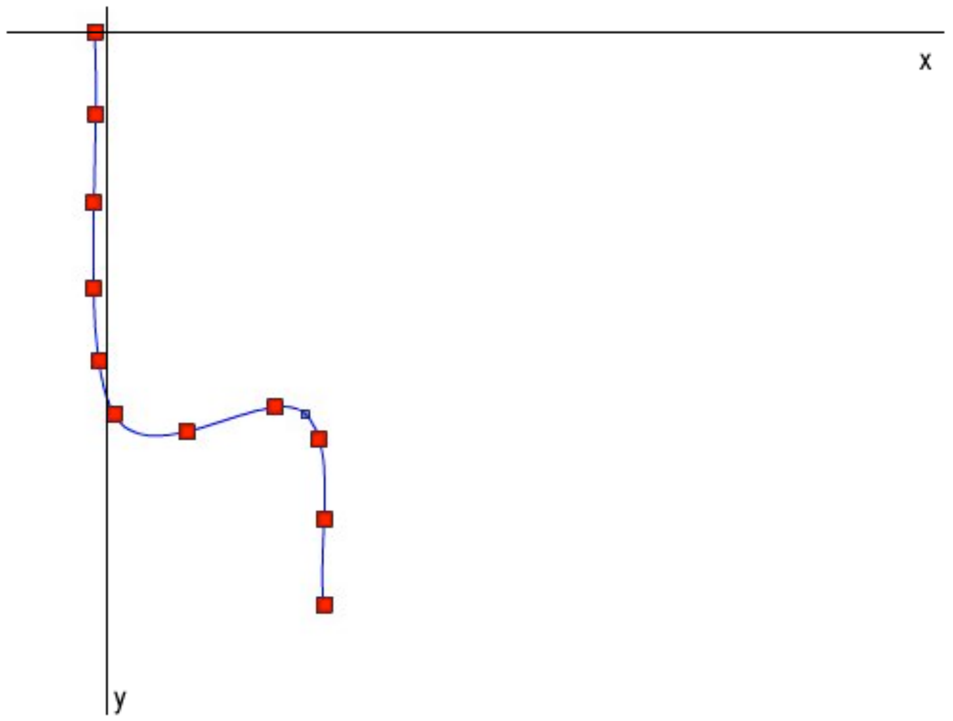
A different parameterization is the uniform formula,

$$i = \min(floor((N-1)t + 1, N-1)$$
$$t^* = (N-1)t - floor((N-1)t)$$

The latter parameterization is the default (*__myCurve.parameterize* = *UNIFORM*). For comparative purposes, the chord-length parameterization can be selected with *__myCurve.parameterize* = *CHORD_LENGTH*. The two parameterizations apply to the same set of geometric constraints, so the same curve plot is generated.

While this parameterization is useful for drawing the spline, it has a drawback in terms of animation. Ideally, an object moving along the curve should do so with constant velocity. Equal increments in parameter, t, should result in equal distance along the curve. This is not achieved with chord-length or similar parameterization, as illustrated in the following example. Consider the control points,

(-10,0) (0,200) (100,200) and (110,300). The Catmull-Rom spline fitting these control points is shown below (in the Flash Stage coordinate space) with markers placed at a distance of 0.1 in t .



Notice that the distance between markers is not constant near the bends at the second and third control points, even though they appear pretty close to constant elsewhere. As t is varied uniformly between 0 and 1, an object moving along the curve will move faster and then slower around those bends than it would at other points on the curve.

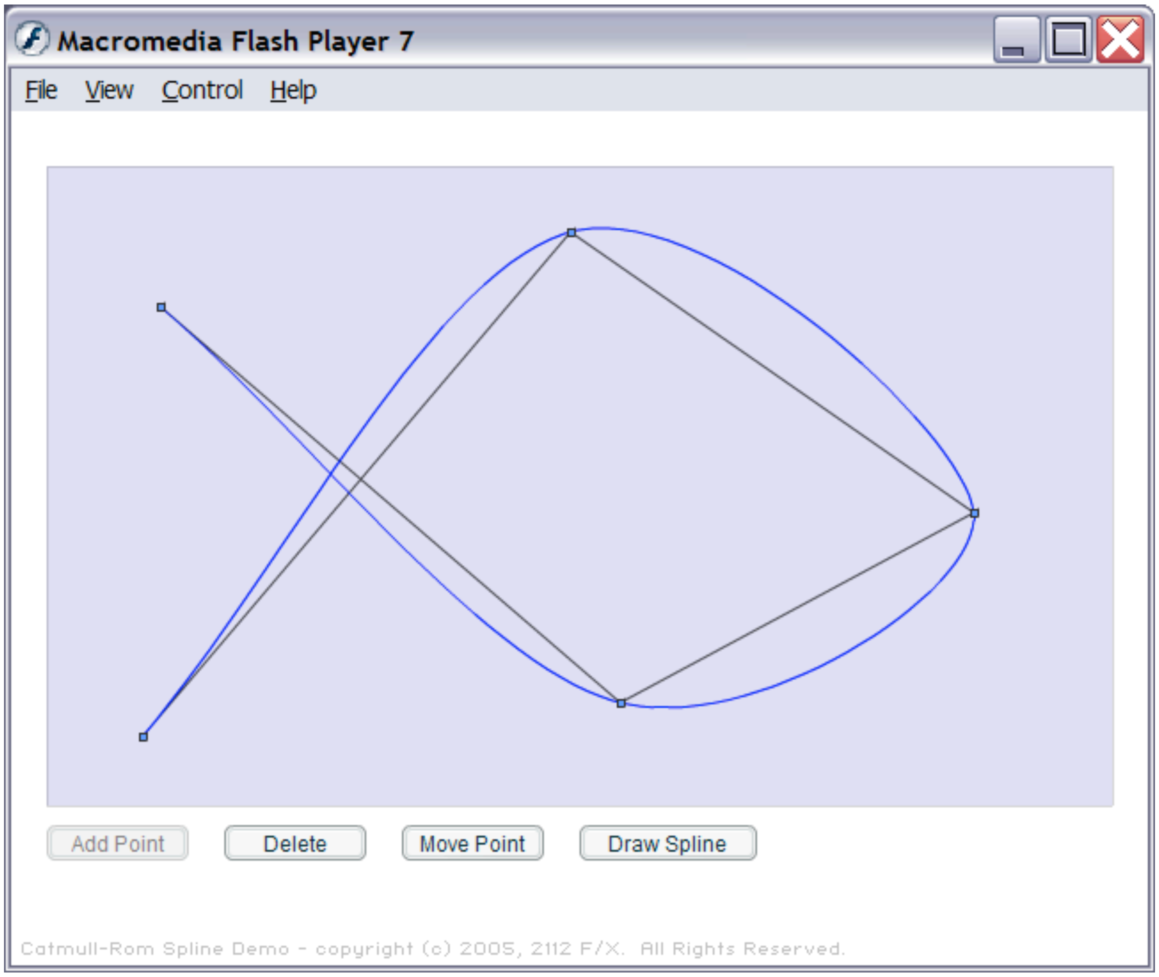
The cure to this situation is an arc-length parameterization, which is a topic that will have to wait for a future TechNote.

Piecewise Cubic Implementation

The cubic coefficients for each segment are precomputed, based on the specified method for endpoint tangents. For the i-th control point, the direct application of equation [7] for the *x*-coordinate yields

$$\begin{aligned} a_x &= -x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2} \\ b_x &= 2x_{i-1} - 5x_i + 4x_{i+1} - x_{i+2} \\ c_x &= x_{i+1} - x_{i-1} \\ d_x &= 2x_i \end{aligned}$$

An example is illustrated below.


Macromedia Flash Player 7
File
View
Control
Help
Add Point
Delete
Move Point
Draw Spline
Catmull-Rom Spline Demo - copyright (c) 2005, 2112 F/X. All Rights Reserved.

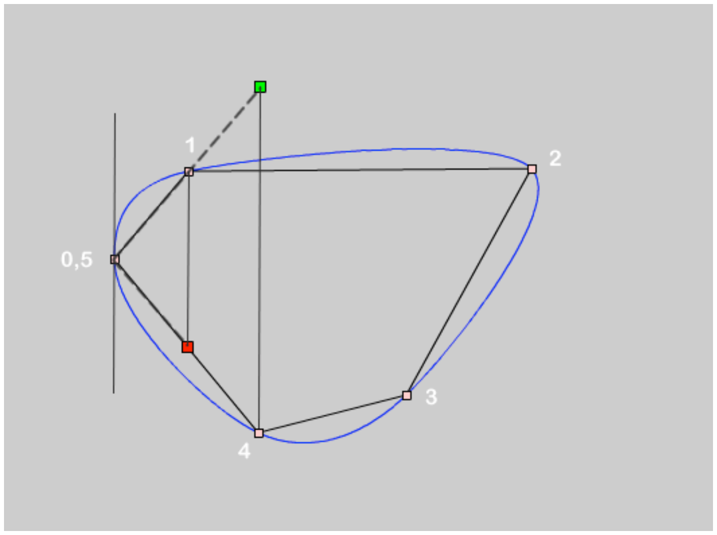
Notice how the curve 'pinches' slightly against the chord at each endpoint as a result of the pure reflection of the first and last segments about the endpoints.

The curve moves very smoothly through each knot. The Catmull-Rom spline has the additional benefit that modification of a single knot only affects the curve near that knot (local control). Since the Catmull-Rom blending functions do not sum to unity, the convex hull property is lost.

An arc-length parameterization and a little more flexibility with the auxiliary control points will provide the basis for a general path-tweening class.

Closed Loops

The auxiliary control points at the beginning and end of the Catmull-Rom spline provide flexibility for a variety of curve shapes, including closed loops. A common misconception among beginning students is that these endpoints somehow form 'tangent handles'. Recall that the tangent at any knot is parallel to the chord between previous and successive knots. One of the simplest strategies to create a smooth, closed loop is to place the outermost control points along the chords into and out of the first knot. This is illustrated in the following diagram.



The knots are numbered 0-5 (the last knot is automatically added when the *closed* property is set). The red marker indicates the first auxiliary control point. The green marker indicates the outermost auxiliary control point. The direction of the vector from initial knot to each control point is along the chord emanating from the first knot. The distance from first knot to both the second and next-to-last knots determines the distance along each chord to place the auxiliary control points. The first auxiliary control point is placed along the chord from the first knot to the next-to-last knot, but at a distance equal to that from the first to second knot. The process is reversed for the outermost auxiliary control point.

The method illustrated here was chosen for ease of illustration. The graphic illustration should convince you of at least G-1 continuity. What would have to be modified for C-1?

References

1) Catmull, E. and Rom, R., "A Class of Local Interpolating Splines," Computer Aided Geometric Design, R.E. Barnhill and R.F. Reisenfeld, Editors, Academic Press, NY, 1974, pp. 317-326.

2) Kochanek, D.H., and Bartels, R., "Interpolating Splines with Local Tension, Continuity, and Bias," SIGGRAPH Conference Proceedings, July 1984, pp. 33-41.