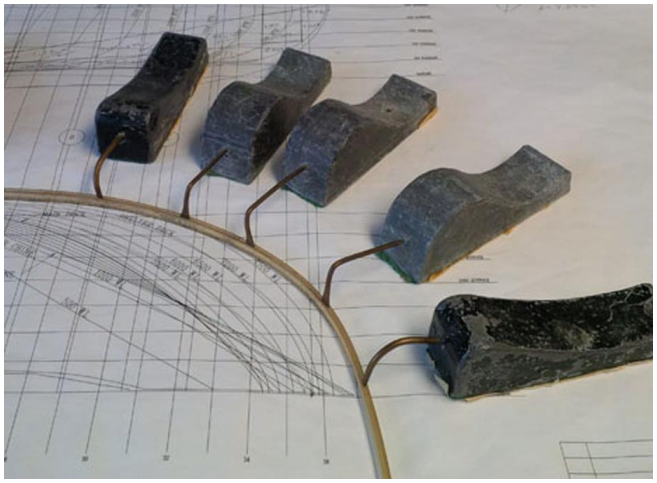
Chapter 18

Simple Spline Curves

18.1 Introduction

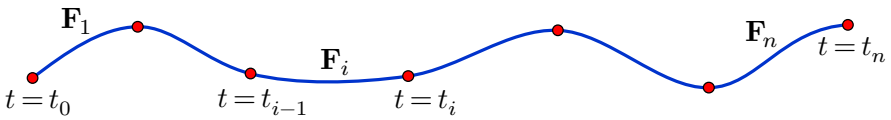
Originally, in the days before CAD, a "spline" was a thin flexible strip of wood that was used to help draw free-form curves. The spline was held in place by weights called "ducks", which the draftsman would move around until the spline shape matched a curve given on a drawing.



We will define a **spline** curve to be one that is formed by joining together polynomial or rational pieces, usually in way that makes the junctions fairly smooth. In other words, a spline is a curve that is defined piecewise, using different functions over different intervals

$$\mathbf{F}(t) = \begin{cases} \mathbf{F}_1(t) & \text{for } t_0 \le t < t_1 \\ \mathbf{F}_2(t) & \text{for } t_1 \le t < t_2 \\ \vdots & \\ \mathbf{F}_i(t) & \text{for } t_{i-1} \le t < t_i \\ \vdots & \\ \mathbf{F}_n(t) & \text{for } t_{n-1} \le t < t_n \end{cases}$$

The quadratic and cubic curves that we discussed in the previous three chapters can be used to build simple spline curves. So, in the picture below, each segment might be either a quadratic or a cubic curve:

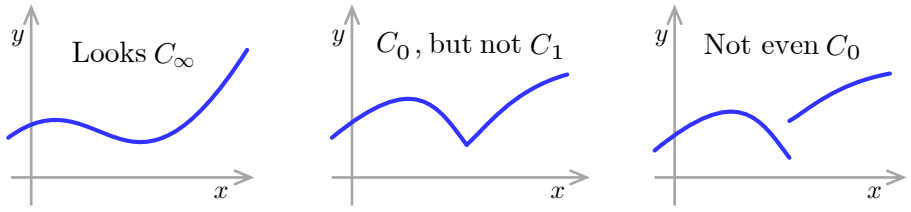


We will cover more general spline curves later, in chapter xxx, but the simple ones described in this chapter are adequate for many purposes.

18.2 Continuity

In mathematics, a function is said to be **continuous** if (roughly speaking) there are no abrupt jumps in its value. For short, we say that the function is C_0. If the function's first derivative exists and is also continuous, we say that the function itself is C_1. More generally, a function is said to be C_n if its n-th order derivative exists and is continuous. If all derivatives, of all orders, are continuous, then we say that the function is C_∞.

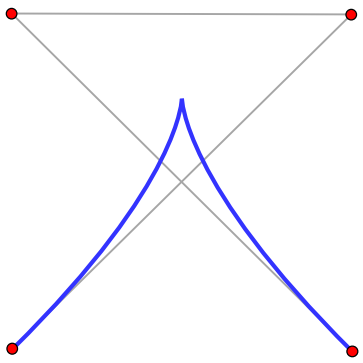
For a real-valued function $f : \mathbb{R} \to \mathbb{R}$, continuity is fairly easy to understand. The function f is C_0 if it's graph has no vertical jumps, and it is C_1 if its graph has no sharp corners.



So, we see that the continuity of a curve is a measure of its smoothness, in some sense.

However, the curves that we are studying are typically not described by real-valued functions of the form $y = f(x)$, but rather by vector-valued functions, i.e. parametric equations of the form $\mathbf{X} = \mathbf{F}(u)$. For these sorts of curves, the connections between continuity and geometric smoothness are rather less clear. One major problem is immediately obvious: continuity is a property of the functions used to parameterize the curve, whereas geometric smoothness is a property of the curve itself. For a given curve, there are always many different parameterizations, and it may be (for example) that some of these are C_1 while others are not.

So, suppose we have a function $\mathbf{F} : \mathbb{R} \to \mathbb{R}^3$ that provides a parameterization of a curve. Then saying that $\mathbf{F}$ is C_0 , certainly means that there are no gaps in the curve. However, C_1 continuity does not tell us much about the shape of the curve. It's quite possible that the function $\mathbf{F}$ is C_∞, and yet there will still be a sharp corner in the curve. Here is an example:



The equation of the curve is

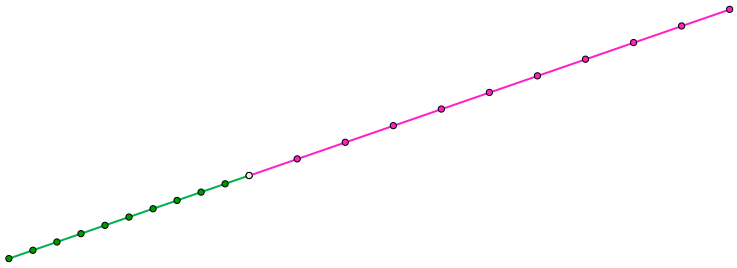
$$\mathbf{X}(t) = (3t - 6t^2 + 4t^3, 3t - 3t^2)$$

so you can easily check that it has derivatives of all orders. The anomaly arises because the derivative vector of the curve is zero at the corner point. As we pass through the corner point, the derivative vector shrinks in length, becomes zero, and then grows again, but pointing in a different direction, and this change is all perfectly "smooth" in the mathematical sense.

Conversely, there are curves that are perfectly smooth, but which are not C_1. For example, consider the curve with equation

$$\mathbf{X}(t) = \begin{cases} (t, t) & \text{for } 0 \le t < 1 \\ (2t-1, 2t-1) & \text{for } 1 \le t \le 2 \end{cases}$$

We see that $\mathbf{X}'(t) = (1, 1)$ when $0 \le t < 1$ but $\mathbf{X}'(t) = (2, 2)$ when $1 \le t \le 1$, so there is an abrupt change in the first derivative vector at $t = 1$, which implies that the curve is not C_1 at that point. However, notice that the curve is just consists of two portions of the straight line $y = x$, which is certainly a very smooth curve, by any reasonable definition.



Of course, at $t = 1$, the derivative vector changes length, but it does not change direction. Or, saying this another way, the (unit) tangent vector is continuous. If a curve's unit tangent vector is continuous, we say that the curve is **geometrically continous** or visually continuous, or we simply say that the curve is G_1. As we have seen, there are curves that are C_1, but not G_1, and curves that are G_1, but not C_1. Similarly, we say that a curve is G_2 if its curvature is continuous. This is not the same as being C_2, of course. In fact, it can't possibly be the same, because a curve's second derivative depends on its parameterization, while its curvature does not.

Most of the curve equations we have seen so far (circles, conics, quadratics, cubics) are C_∞ and (usually) perfectly smooth, so we have not needed to worry about continuity. However, in this chapter and later ones, we will be forming composite curves by joining together simpler ones, and we'll need to consider continuity and smoothness at these junctions. Specifically, our curves will be typically be defined piecewise by equations of the form

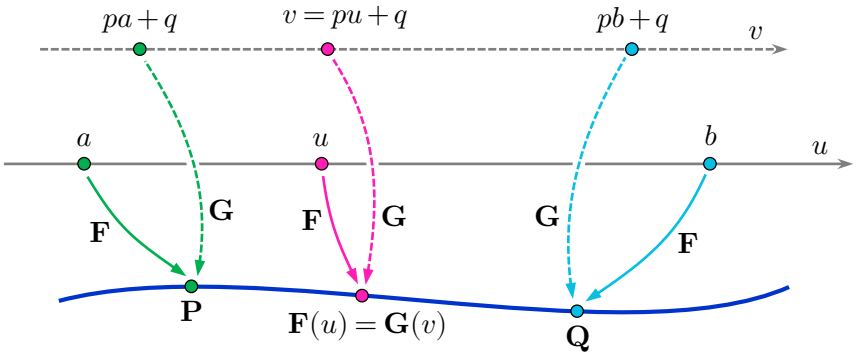
$$\mathbf{X}(t) = \begin{cases} \mathbf{F}(t) & \text{for } a \le t < b \\ \mathbf{G}(t) & \text{for } b \le t \le c \end{cases}$$

and we'll need to consider what happens at the junction where $t = b$. Some possible cases are

- If $\mathbf{F}(b) = \mathbf{G}(b)$, then the join is C_0
- If $\mathbf{F}(b) = \mathbf{G}(b)$ and $\mathbf{F}'(b) = \mathbf{G}'(b)$, then the join is C_1
- If $\mathbf{F}(b) = \mathbf{G}(b)$ and $\mathbf{F}'(b) = \alpha\mathbf{G}'(b)$, for some $\alpha > 0$, then the join is G_1
- If $\mathbf{F}(b) = \mathbf{G}(b)$, $\mathbf{F}'(b) = \mathbf{G}'(b)$, and $\mathbf{F}''(b) = \mathbf{G}''(b)$, then the join is C_2
- If $\mathbf{F}$ and $\mathbf{G}$ agree in position, unit tangent, and curvature at $t = b$, then the join is G_2

18.3 Change of Parameters

Suppose we have a curve with parametric equations $\mathbf{F}(u)$. We define a new parameter v by $v = pu + q$, where p and q are constants. Then we can regard u as a function of v: $u(v) = (v - q)/p$. Then, let's define a new parametric equation $\mathbf{G}$ by $\mathbf{G}(v) = \mathbf{F}(u(v))$. We want to understand how this change of parameterization affects the curve and its derivatives.



First, it is clear that the images of the two parameterization mappings are the same; given any interval $[a, b]$, we have

$$\{\mathbf{F}(u) : a \le u \le b\} = \{\mathbf{G}(v) : pa + q \le v \le pb + q\}$$

So, the two parameterizations trace out the same curve, but over different parameter intervals. Specifically, the parameter intervals $u \in [a, b]$ and $v \in [pa + q, pb + q]$ both correspond to the portion of the curve between the points $\mathbf{P}$ and $\mathbf{Q}$.

We note that the length of the new parameter interval is $p(b - a)$, which is p times the length of the original one. Also, if u_0 is some fixed value of u and $v_0 = pu_0 + q$, then, using the chain rule for differentiation, we have

$$\frac{d\mathbf{G}}{dv}(v_0) = \frac{du}{dv}\frac{d\mathbf{F}}{du}(u_0) = \frac{1}{p}\frac{d\mathbf{F}}{du}(u_0)$$

so the reparameterization has caused a scaling of the curve's first derivative vector. More precisely, the scale factor is $1/p$, where

$$p = \frac{\text{Length of new parameter interval}}{\text{Length of original parameter interval}}$$

This should make sense intuitively: if $p > 1$, then we have expanded the time interval required to traverse the curve, so there has been a reduction in the speed of traversal.

Of course, this reparameterization does not change the shape of the curve, only the "time interval" over which it is traced out. So all purely geometric properties of the curve, like tangent direction, curvature, and torsion, will remain unchanged.

18.4 Making a G_1 Junction C_1

Suppose we have a parametric curve $\mathbf{X}(t)$ that is defined piecewise, say

$$\mathbf{X}(t) = \begin{cases} \mathbf{F}(t) & \text{for } a \le t < b \\ \mathbf{G}(t) & \text{for } b \le t \le c \end{cases}$$

We will assume that the junction is G_1, and will find a way to make it C_1. Since the junction is $\mathbf{G}_1$, we can find a positive number p such that

$$\frac{d\mathbf{F}}{dt}(b) = p\frac{d\mathbf{G}}{dt}(b)$$

According to the results from the previous section, we simply have to scale the parameter interval of $\mathbf{G}$ by a factor p, making its length $p(c-b)$ instead of $c-b$. In other words, we should express $\mathbf{G}$ as a function of a new parameter that ranges over the interval $[b, b+p(c-b)]$, instead of over $[b, c]$. Then, by the results of the previous section, the first derivative of $\mathbf{G}$ will be scaled by $1/p$, so it will match the first derivative of $\mathbf{F}$, and the joint will be C_1.

Using this technique, we can walk along a curve that has G_1 junctions, and make them all C_1, just by adjusting the lengths of parameter intervals. Making a composite curve C_1 instead of just G_1, does not really impact the user directly, since it does not change the shape of the curve. However, it might improve the reliability of internal algorithms that have trouble handling non-C_1 curves. Also, curve parameterization affects many surface constructions, and C_1 curves often work better, as we will see in chapter xxx.

18.5 Bézier Cubics on Arbitrary Intervals

To construct spline curves, we must be able to parameterize a Bézier-Bernstein curves over some arbitrary interval $[a, b]$, rather than the unit interval $[0, 1]$. We let $h = b - a$ and then

$$u = \frac{t-a}{b-a} = \frac{t-a}{h} \quad ; \quad 1-u = \frac{b-t}{b-a} = \frac{b-t}{h}$$

Then the equations of the cubic Bernstein basis functions on $[a, b]$ are

$$\begin{aligned}
\phi_0(t) &= (1-u)^3 = \frac{1}{h^3}(b-t)^3 \\
\phi_1(t) &= 3u(1-u)^2 = \frac{3}{h^3}(t-a)(b-t)^2 \\
\phi_2(t) &= 3u^2(1-u) = \frac{3}{h^3}(t-a)^2(b-t) \\
\phi_3(t) &= u^3 = \frac{1}{h^3}(t-a)^3
\end{aligned}$$

and the corresponding Bézier curve is

$$\mathbf{X}(t) = \phi_0(t)\mathbf{P}_0 + \phi_1(t)\mathbf{P}_1 + \phi_2(t)\mathbf{P}_2 + \phi_3(t)\mathbf{P}_3 \qquad (a \le t \le b)$$

Then $\mathbf{X}(a) = \mathbf{P}_0$ and $\mathbf{X}(b) = \mathbf{P}_3$, and straightforward differentiation shows that

$$\begin{aligned}
\frac{d\mathbf{X}}{dt}(t=a) &= \frac{3}{h}(\mathbf{P}_1 - \mathbf{P}_0) \\
\frac{d\mathbf{X}}{dt}(t=b) &= \frac{3}{h}(\mathbf{P}_3 - \mathbf{P}_2)
\end{aligned} \tag{18.1}$$

Of course, this is exactly the scaling of first derivative vectors that we would expect, based on the results from **section (18.3)**. By differentiating again, it is easy to show that the second derivative at the start of the curve is given by

$$\frac{d^2\mathbf{X}}{dt^2}(t=a) = \frac{6}{h^2}\{(\mathbf{P}_2 - \mathbf{P}_1) - (\mathbf{P}_1 - \mathbf{P}_0)\}$$

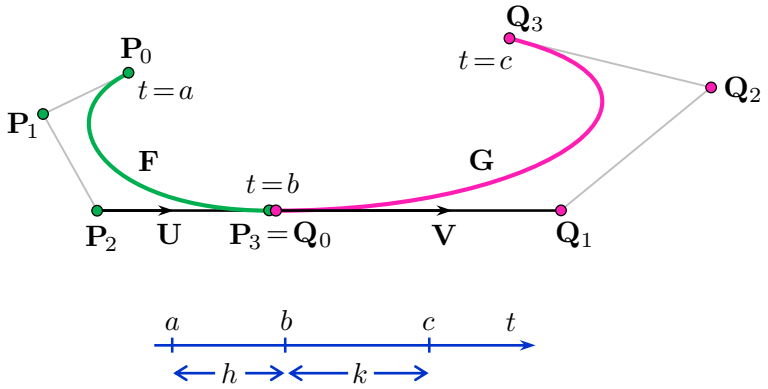
and its curvature is

$$\kappa(a) = \frac{(m-1)\,\|(\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_1)\|}{m\,\|\mathbf{P}_1 - \mathbf{P}_0\|^3}$$

As we would expect, $\kappa(a)$ does not depend on h, because curvature is a purely geometric property that does not depend on parameterization.

18.6 Making C_1 Junctions Between Bézier Curves

Let $\mathbf{F}$ be a cubic curve with Bézier poles $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and let $\mathbf{G}$ be a cubic with poles $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$. Suppose $\mathbf{F}$ uses the parameter interval $[a, b]$, and $\mathbf{G}$ uses the interval $[b, c]$, and let $h = b - a$ and $k = c - b$ be the lengths of the parameter intervals. Also, let $\mathbf{U} = \mathbf{P}_3 - \mathbf{P}_2$ and $\mathbf{V} = \mathbf{Q}_1 - \mathbf{Q}_0$ be the relevant legs of the two curves' control polygons:



We assume that the vectors $\mathbf{U}$ and $\mathbf{V}$ are parallel, so that the junction between $\mathbf{F}$ and $\mathbf{G}$ is already G_1; we will use reparameterization to make it C_1. By (18.1), we know that

$$\mathbf{F}'(b) = \frac{3}{h}\mathbf{U} \quad ; \quad \mathbf{G}'(b) = \frac{3}{k}\mathbf{V} \tag{18.2}$$

so, to obtain C_1 continuity, we just have to choose h and k so that

$$\frac{1}{h}\mathbf{U} = \frac{1}{k}\mathbf{V}$$

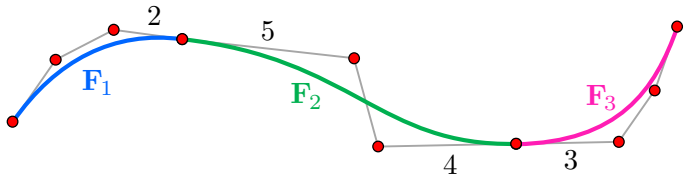
or equivalently, we need to choose h and k so that

$$\frac{h}{k} = \frac{\|\mathbf{U}\|}{\|\mathbf{V}\|} \tag{18.3}$$

Or, in words, we get C_1 continuity when the ratio of parameter interval lengths is the same as the ratio of control polygon legs.

18.7 Example

The picture below shows three Bézier curves, with the lengths of some polygon legs marked.



We want to construct a C_1 curve from these three segments. To do this, we will use the reparameterization technique described in the previous section. So, let h_1, h_2 and h_3 be the parameter ranges that we will assign to the three segments. By (18.3), we know that the junctions will be C_1 if

$$\frac{h_1}{h_2} = \frac{2}{5} \quad ; \quad \frac{h_2}{h_3} = \frac{4}{3}$$

We can arbitrarily set $h_1 = 1$. Then the equations above tell us that $h_2 = \frac{5}{2}$ and $h_3 = \frac{3}{4}h_2 = \frac{15}{8}$. If we choose $t = 0$ as the start parameter of $\mathbf{F}_1$, then the parameter values at the segment end-points are

$$t_0 = 0$$

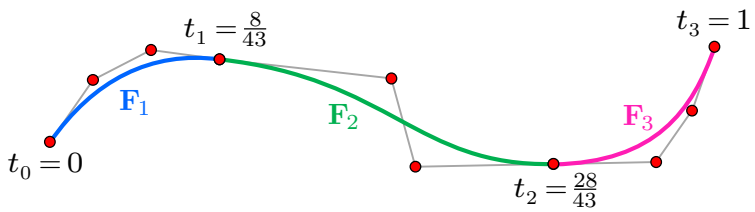
$$t_1 = t_0 + h_1 = 1$$

$$t_2 = t_1 + h_2 = 1 + \tfrac{5}{2} = \tfrac{7}{2}$$

$$t_3 = t_2 + h_3 = \tfrac{7}{2} + \tfrac{15}{8} = \tfrac{43}{8}$$

If we want, we can rescale these parameter values so that $t_3 = 1$. The rescaling will not change the length ratios, so the curve will still be C_1. With rescaling, we get

$$t_0 = 0 \quad ; \quad t_1 = \tfrac{8}{43} \quad ; \quad t_2 = \tfrac{28}{43} \quad ; \quad t_3 = 1$$



18.8 Hermite Cubics on Arbitrary Intervals

We can represent a cubic curve in Hermite form over over some arbitrary interval $[a, b]$, too. In **section (??)** we described the Hermite cubic functions for the interval $[0, 1]$. These were

$$\phi_0(u) = 1 - 3u^2 + 2u^3 \quad ; \quad \phi_1(u) = 3u^2 - 2u^3$$

$$\psi_0(u) = u - 2u^2 + u^3 \quad ; \quad \psi_1(u) = u^3 - u^2$$

We recall that these functions were constructed to give us the properties

$$\phi_0(0) = 1 \quad ; \quad \phi_0(1) = 0 \quad ; \quad \phi_0'(0) = 0 \quad ; \quad \phi_0'(1) = 0$$

$$\phi_1(0) = 0 \quad ; \quad \phi_1(1) = 1 \quad ; \quad \phi_1'(0) = 0 \quad ; \quad \phi_1'(1) = 0$$

$$\psi_0(0) = 0 \quad ; \quad \psi_0(1) = 0 \quad ; \quad \psi_0'(0) = 1 \quad ; \quad \psi_0'(1) = 0$$

$$\psi_1(0) = 0 \quad ; \quad \psi_1(1) = 0 \quad ; \quad \psi_1'(0) = 0 \quad ; \quad \psi_1'(1) = 1$$

Suppose we are given two points $\mathbf{P}_0$ and $\mathbf{P}_1$ and two vectors $\mathbf{V}_0$ and $\mathbf{V}_1$, and we want to construct the corresponding Hermite cubic over the interval $[a, b]$. We let $h = b - a$ and $\lambda = 1/h = 1/(b - a)$ and

$$u = \frac{t - a}{b - a} = \lambda(t - a)$$

and then the equation of the desired Hermite cubic is

$$\mathbf{X}(t) = \phi_0(u)\mathbf{P}_0 + \phi_1(u)\mathbf{P}_1 + \psi_0(u)\mathbf{V}_0 + \psi_1(u)\mathbf{V}_1 \qquad (a \le t \le b) \tag{18.4}$$

It's clear that $\mathbf{X}(a) = \mathbf{P}_0$ and $\mathbf{X}(b) = \mathbf{P}_1$, and differentiating gives

$$\begin{aligned} \frac{d\mathbf{X}}{dt}(t=a) &= \frac{d\psi_0}{du}(u=0)\frac{du}{dt}h\mathbf{V}_0 = \lambda h\mathbf{V}_0 = \mathbf{V}_0 \\ \frac{d\mathbf{X}}{dt}(t=b) &= \frac{d\psi_1}{du}(u=1)\frac{du}{dt}h\mathbf{V}_1 = \lambda h\mathbf{V}_1 = \mathbf{V}_1 \end{aligned} \tag{18.5}$$

For later reference, we also note the values of the second and third derivatives at the curve end-points:

$$\mathbf{X}''(a) = 6\lambda^2(\mathbf{P}_1 - \mathbf{P}_0) - 2\lambda(2\mathbf{V}_0 + \mathbf{V}_1) \tag{18.6}$$

$$\mathbf{X}''(b) = 6\lambda^2(\mathbf{P}_0 - \mathbf{P}_1) + 2\lambda(\mathbf{V}_0 + 2\mathbf{V}_1) \tag{18.7}$$

$$\mathbf{X}'''(t) = 12\lambda^3(\mathbf{P}_1 - \mathbf{P}_0) + 6\lambda^2(\mathbf{V}_0 + \mathbf{V}_1) \tag{18.8}$$

18.9 Spline Input Data

There are many different situations where we need to construct and interpolating spline, with many different combinations of input data. Input data might include any combination of the following values:

- Points to be interpolated (i.e. data points)
- Parameter values to be assigned to the data points
- First derivative vectors at the data points
- Direction vectors at the data points
- Second derivative values at the data points (rarely)
- Curvature values at the data points

There are two basic scenarios, which are quite different. The first one is the case where a user is interactively creating a curve. The user will certainly want to specify the data points to be interpolated, but often nothing more than this. Sometimes tangent directions or curvatures are specified, too, but this is comparatively rare. It certainly does not make sense for the user to specify quantities that depend on parameterization, such as first or second derivative vectors.

The second scenario is an algorithm that is approximating a given curve. Here, a great deal of information can be obtained from the curve, and can be used to help construct the spline. If we are trying to replicate the parameterization of the curve (not just its shape), then it makes sense to use parameterization-dependent quantities like derivatives. Otherwise, parameterization-independent quantities like tangent direction and curvature should be used.

The following sections describe some of the most commonly used algorithms, but there are many other possibilities.

18.10 Interpolating Points and Derivatives

Suppose we are given n points $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ and n derivative vectors $\mathbf{V}_1, \ldots, \mathbf{V}_n$, that we want to interpolate at parameter values $t_1, \ldots, t_n$. In other words, we want to construct a curve $\mathbf{X}(t)$

such that $\mathbf{X}(t_i) = \mathbf{Q}_i$ and $\mathbf{X}'(t_i) = \mathbf{V}_i$ for $i = 1, 2, \ldots, n$. We can easily solve this problem using the results from **section (18.8)**. For each i, we let $h_i = t_{i+1} - t_i$ and $\lambda_i = 1/(t_{i+1} - t_i)$, and we define the Hermite cubic segment:

$$\mathbf{X}_i(t) = \phi_0(u)\mathbf{P}_i + \phi_1(u)\mathbf{P}_{i+1} + \psi_0(u)h_i\mathbf{V}_i + \psi_1(u)h_i\mathbf{V}_{i+1} \qquad (t_i \le t \le t_{i+1})$$

Then, from (18.5), we have

$$\frac{d\mathbf{X}_i}{dt}(t = t_i) = \lambda_i h_i \mathbf{V}_i = \mathbf{V}_i \quad ; \quad \frac{d\mathbf{X}_i}{dt}(t = t_{i+1}) = \lambda_i h_i \mathbf{V}_{i+1} = \mathbf{V}_{i+1}$$

So, at the junction where $t = t_i$, we have

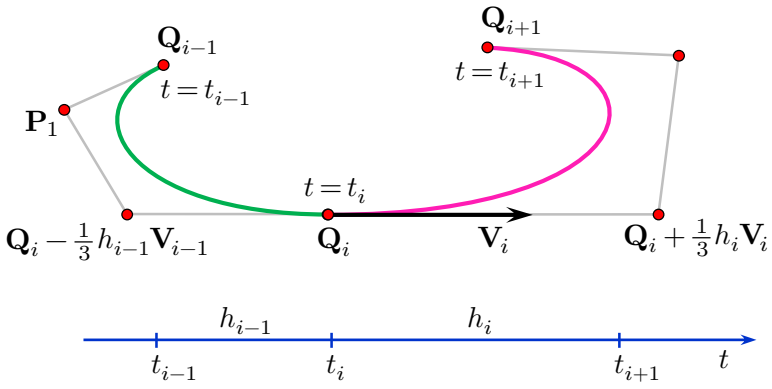
$$\frac{d\mathbf{X}_i}{dt}(t = t_i) = \frac{d\mathbf{X}_{i-1}}{dt}(t = t_i) = \mathbf{V}_i$$

which means that the junction is C_1.

An alternative approach is to construct a cubic in Bézier form on each interval. So, on the interval $t_i \le t \le t_{i+1}$ we define Bézier control points $\mathbf{P}_0$, $\mathbf{P}_a$, $\mathbf{P}_b$, $\mathbf{P}_1$ by

$$\begin{aligned}
\mathbf{P}_0 &= \mathbf{Q}_i \\
\mathbf{P}_a &= \mathbf{Q}_i + \tfrac{1}{3}h_i\mathbf{V}_i \\
\mathbf{P}_b &= \mathbf{Q}_{i+1} - \tfrac{1}{3}h_i\mathbf{V}_{i+1} \\
\mathbf{P}_1 &= \mathbf{Q}_{i+1}
\end{aligned}$$

and we define $\mathbf{X}_i$ to be the corresponding Bézier curve parameterized over the interval $[t_i, t_{i+1}]$.



Then, by (18.1), we know that

$$\frac{d\mathbf{X}_i}{dt}(t = t_i) = 3\lambda_i(\mathbf{P}_a - \mathbf{P}_0) = \mathbf{V}_i$$

$$\frac{d\mathbf{X}_i}{dt}(t = t_{i+1}) = 3\lambda_i(\mathbf{P}_1 - \mathbf{P}_b) = \mathbf{V}_{i+1}$$

So, at the junction where $t = t_i$, we again have

$$\frac{d\mathbf{X}_i}{dt}(t = t_i) = \frac{d\mathbf{X}_{i-1}}{dt}(t = t_i) = \mathbf{V}_i$$

This is what we would expect, since we have constructed the same curve as before, only by different reasoning.

18.11 Interpolating Points and Directions

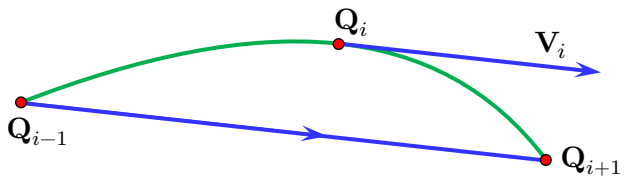
Suppose again that we are given n points $\mathbf{Q}_1, \dots, \mathbf{Q}_n$ and n vectors $\mathbf{U}_1, \dots, \mathbf{U}_n$, that we want to interpolate at parameter values $t_1, \dots, t_n$. But now suppose that only the directions of the vectors are important, not their magnitudes. In other words, we want to construct a curve $\mathbf{X}(t)$ such that $\mathbf{X}(t_i) = \mathbf{Q}_i$ and $\mathbf{X}'(t_i)$ is parallel to $\mathbf{U}_i$ for $i = 1, 2, \dots, n$. We can solve this problem by using a sequence of cubic curves, each constructed from two points and two tangent directions. We saw several ways to perform this construction in **chapter 17**. For example:

- In **section (17.14)**, there are simple explicit formulas telling us what lengths to use for first derivative vectors
- In **section (16.16)**, we saw a way to construct a conic with minimum eccentricity, which is (in some sense) the "nicest" conic that interpolates two points and two tangent directions. We can obtain the rho (ρ) value of this conic, and use it to construct a rho cubic, as in **section (17.13)**
- If we are approximating a curve, then we can calculate another point on the interior of each segment, and then use the construction described in **section (17.18)**
- Again, if we are approximating a curve, we can obtain curvature information, and use this in the construction described in **section (17.16)**

Any one of these constructions will give us a string of cubic curves that have G_1 joins. Then (if we want), we can make the joins C_1 by adjusting the lengths of parameter intervals, as described in **section (18.4)** and **section (18.6)**.

18.12 Catmull-Rom Splines

Suppose, as usual, that we are given n points $\mathbf{Q}_1, \dots, \mathbf{Q}_n$ that we want to interpolate, but this time no derivative or direction vectors. If we can somehow calculate some derivative vectors, $\mathbf{V}_1, \dots, \mathbf{V}_n$, then we can use the technique from **section (18.10)** to construct a spline curve. There are many ways to invent derivative vectors, but they are all somewhat heuristic, and no one method is clearly the best in all situations. One common method is due to Catmull and Rom. The idea is to obtain the derivative vector $\mathbf{V}_i$ from a quadratic curve drawn through the given points $\mathbf{Q}_{i-1}$, $\mathbf{Q}_i$, $\mathbf{Q}_{i+1}$.



We discussed the construction of a quadratic curve through 3 points in **chapter 15**. Specifically, in **section (15.15)**, we saw that the curve

$$\mathbf{Q}(t) = \frac{(t-t_i)(t-t_{i+1})}{(t_{i-1}-t_i)(t_{i-1}-t_{i+1})}\mathbf{Q}_{i-1} + \frac{(t-t_{i-1})(t-t_{i+1})}{(t_i-t_{i-1})(t_i-t_{i+1})}\mathbf{Q}_i + \frac{(t-t_{i-1})(t-t_i)}{(t_{i+1}-t_{i-1})(t_{i+1}-t_i)}\mathbf{Q}_{i+1}$$

gives us $\mathbf{Q}(t_{i-1}) = \mathbf{Q}_{i-1}$, $\mathbf{Q}(t_i) = \mathbf{Q}_i$, and $\mathbf{Q}(t_{i-1}) = \mathbf{Q}_{i+1}$. The Catmull-Rom algorithm then uses $\mathbf{V}_i = \mathbf{Q}'(t_i)$ as the derivative of the spline curve at the point $\mathbf{Q}_i$. From **equation (15.7)**,

we know that this derivative is

$$\mathbf{Q}'(t_i) = \frac{t_i - t_{i+1}}{(t_i - t_{i-1})(t_{i+1} - t_{i-1})}\mathbf{Q}_{i-1} + \frac{2t_i - t_{i-1} - t_{i+1}}{(t_{i-1} - t_i)(t_{i+1} - t_i)}\mathbf{Q}_i + \frac{t_i - t_{i-1}}{(t_{i-1} - t_{i+1})(t_i - t_{i+1})}\mathbf{Q}_{i+1}$$

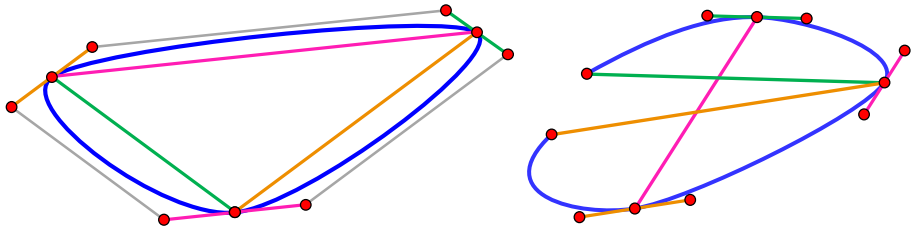
As we know, the shape of the quadratic curve (and hence its first derivative vector) will depend on the parameter values t_{i-1}, t_i, t_{i+1} that we assign to the three points. In the simplest form of the Catmull-Rom algorithm, the values used are $-1, 0, 1$. Then the quadratic curve becomes

$$\mathbf{Q}(t) = \tfrac{1}{2}(t^2 - t)\mathbf{Q}_{i-1} + (1 - t^2)\mathbf{Q}_i + \tfrac{1}{2}(t^2 + t)\mathbf{Q}_{i+1} \tag{18.9}$$

The derivative vector of this curve at the point $\mathbf{Q}_i$ (where $t = 0$) is simply

$$\mathbf{V}_i = \tfrac{1}{2}(\mathbf{Q}_{i+1} - \mathbf{Q}_{i-1}) \tag{18.10}$$

The Catmull-Rom algorithm is used in Powerpoint, so you can experiment with it there. Here are two examples, with color coding to show how the derivative vectors are obtained:



On a closed curve, like the one on the left in the picture above, the Catmull-Rom calculation (18.10) can be applied at every point. However, on an open curve, special handling is required at the first and last points. At the first point, we use the first three points to construct a quadratic curve. In other words, we set $i = 1$ in (18.9). But then we use the first derivative vector of this curve at $t = -1$. This gives us a vector $\mathbf{V}_1 = -\frac{3}{2}\mathbf{Q}_1 + 2\mathbf{Q}_2 - \frac{1}{2}\mathbf{Q}_3$. We use a similar technique at the end of the curve.

The main benefit of the Catmull-Rom algorithm is its simplicity — given the input points, the construction of the curve (in Bézier form, say) involves very little programming, and very little arithmetic. Also, the curve definition is local, in the sense that a given section of the curve depends on only four points. So, when the user moves a point, recalculating the curve is easy, and can be done in real time.

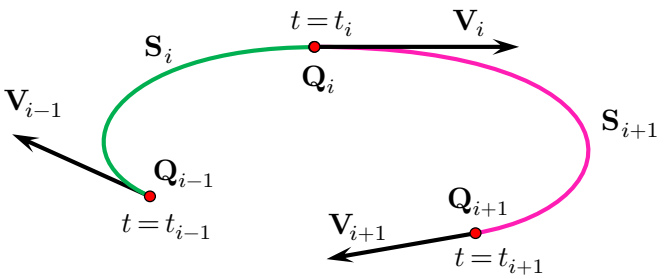
18.13 C_2 Cubic Splines

Again, suppose that we are given $n + 1$ points $\mathbf{Q}_0, \ldots, \mathbf{Q}_n$ that we want to interpolate at parameter values $t_0, \ldots, t_n$, but we have no derivative or direction vectors, so we need to calculate these somehow. The Catmull-Rom algorithm described in the previous section is one approach; another is one based on the idea of enforcing C_2 continuity at each junction.

For $i = 1, 2, \ldots, n$, let $\mathbf{S}_i$ be the Hermite cubic segment satisfying

$$\mathbf{S}_i(t_{i-1}) = \mathbf{Q}_{i-1} \quad ; \quad \mathbf{S}_i'(t_{i-1}) = \mathbf{V}_{i-1}$$
$$\mathbf{S}_i(t_i) = \mathbf{Q}_i \quad ; \quad \mathbf{S}_i'(t_i) = \mathbf{V}_i$$

where $\mathbf{V}_{i-1}$ and $\mathbf{V}_i$ are first derivative vectors that are unknown, as yet.



Then, for $i = 1, 2, \ldots, n-1$, we have

$$\mathbf{S}_i(t_i) = \mathbf{S}_{i+1}(t_i) = \mathbf{Q}_i$$

$$\mathbf{S}_i'(t_i) = \mathbf{S}_{i+1}'(t_i) = \mathbf{V}_i$$

and so the string of cubics is C_1 continuous at the parameter values $t_1, \ldots, t_n$ where the joins occur. For each i, we let $h_i = t_i - t_{i-1}$ and $\lambda_i = 1/(t_i - t_{i-1})$. Then from **equation (18.7)**, we have

$$\mathbf{S}_i''(t_i) = 6\lambda_i^2(\mathbf{Q}_{i-1} - \mathbf{Q}_i) + 2\lambda_i(\mathbf{V}_{i-1} + 2\mathbf{V}_i)$$

Similarly, from **equation (18.6)**, we have

$$\mathbf{S}_{i+1}''(t_i) = 6\lambda_{i+1}^2(\mathbf{Q}_{i+1} - \mathbf{Q}i) - 2\lambda_{i+1}(2\mathbf{V}_i + \mathbf{V}_{i+1})$$

To get C_2 continuity at $t = t_i$, we need to have $\mathbf{S}_i''(t_i) = \mathbf{S}_{i+1}''(t_i)$. Using the expressions from the two equations above, this means that

$$6\lambda_i^2(\mathbf{Q}_{i-1} - \mathbf{Q}_i) + 2\lambda_i(\mathbf{V}_{i-1} + 2\mathbf{V}_i) = 6\lambda_{i+1}^2(\mathbf{Q}_{i+1} - \mathbf{Q}i) - 2\lambda_{i+1}(2\mathbf{V}_i + \mathbf{V}_{i+1})$$

Rearranging this, we obtain

$$\lambda_i\mathbf{V}_{i-1} + 2(\lambda_i + \lambda_{i+1})\mathbf{V}_i + \lambda_{i+1}\mathbf{V}_{i+1} = 3\lambda_{i+1}^2(\mathbf{Q}_{i+1} - \mathbf{Q}_i) + 3\lambda_i^2(\mathbf{Q}_i - \mathbf{Q}_{i-1}) \tag{18.11}$$

for $i = 1, 2, \ldots, n-1$. So, we have a system of $n-1$ linear equations involving the $n+1$ unknown derivative vectors $\mathbf{V}_0, \ldots, \mathbf{V}_n$. To simplify the notation, we write

$$a_i = \lambda_i$$

$$b_i = 2(\lambda_i + \lambda_{i+1})$$

$$c_i = \lambda_{i+1}$$

$$\mathbf{D}_i = 3\lambda_{i+1}^2(\mathbf{Q}_{i+1} - \mathbf{Q}_i) + 3\lambda_i^2(\mathbf{Q}_i - \mathbf{Q}_{i-1})$$

Then (18.11) becomes

$$a_i\mathbf{V}_{i-1} + b_i\mathbf{V}_i + c_i\mathbf{V}_{i+1} = \mathbf{D}_i$$

for $i = 1, 2, \ldots, n-1$. Writing these equations in matrix form gives us:

$$\begin{bmatrix} a_1 & b_1 & c_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_2 & b_2 & c_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_3 & b_3 & c_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & a_{n-1} & b_{n-1} & c_{n-1} \end{bmatrix} \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \\ \vdots \\ \mathbf{V}_{n-2} \\ \mathbf{V}_{n-1} \\ \mathbf{V}_n \end{bmatrix} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \mathbf{D}_3 \\ \vdots \\ \mathbf{D}_{n-2} \\ \mathbf{D}_{n-1} \end{bmatrix} \tag{18.12}$$

Note that we have $n+1$ unknowns, but only $n-1$ equations, so the matrix on the left is not square. To get a linear system that has a unique solution, we must somehow construct two additional equations. We will do this by imposing end conditions on the curve, as explained in the next few sections.

18.14 End Conditions

We use "end conditions" to obtain two further linear equations that we can add to the system (18.13) to make it uniquely solvable.

Natural End Conditions

A cubic spline is meant to model the behaviour of a thin piece of wood (also called a spline). If an end of the piece of wood is not subject to any torque, then it's reasonable to assume that its shape will be close to linear, so its curvature will be zero. So, we may use the so-called "natural" end conditions, which set $\mathbf{S}_1''(t_0) = 0$ and $\mathbf{S}_n''(t_n) = 0$. Using (18.6) and (18.7), we get

$$2\mathbf{V}_0 + \mathbf{V}_1 = 3\lambda_1(\mathbf{Q}_1 - 2\mathbf{Q}_0)$$

$$\mathbf{V}_{n-1} + 2\mathbf{V}_n = 3\lambda_n(2\mathbf{Q}_n - \mathbf{Q}_{n-1})$$

We can combine these two equations with (18.13) to obtain a system that we can solve.

Clamped End Conditions

In some situations, the starting and ending first derivative vectors might be given. In other words, the values of $\mathbf{V}_0$ and $\mathbf{V}_n$ are already known, and these known values serve to "clamp" the ends of the spline. So, we may either remove $\mathbf{V}_0$ and $\mathbf{V}_n$ as unknowns in the linear system (18.13), or we may include two additional equations that specify the known values. Either way, we end up with a square matrix.

If $\mathbf{V}_0$ and $\mathbf{V}_n$ are not known, we can estimate their values. For example, we could use the first derivative of the circle or the parabola through the first three points. In **section (15.15)**, we saw how we can interpolate the points $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$ at parameter values t_0, t_1, t_2. The first derivative of this parabola at $t = t_0$ is given by **equation (15.6)**. If we use this derivative to set the value of $\mathbf{V}_0$, we get the equation

$$\mathbf{V}_0 = \frac{2t_0 - t_1 - t_2}{(t_1 - t_0)(t_2 - t_0)}\mathbf{Q}_0 + \frac{t_0 - t_2}{(t_0 - t_1)(t_2 - t_1)}\mathbf{Q}_1 + \frac{t_0 - t_1}{(t_0 - t_2)(t_1 - t_2)}\mathbf{Q}_2$$

which we can add as the first equation in the linear system (18.13). Similarly, we can use **equation (15.8)**, to find the derivative at $t = t_n$ of the parabola through the points $\mathbf{Q}_{n-2}$, $\mathbf{Q}_{n-1}$ and $\mathbf{Q}_n$, and we may set this equal to $\mathbf{V}_n$. This gives us another equation that we can add to (18.13).

Not-A-Knot End Conditions

Yet another way to obtain two additional equations is to assume that $\mathbf{S}_1 \equiv \mathbf{S}_2$ and $\mathbf{S}_{n-1} \equiv \mathbf{S}_n$, so there are really no joints at $t = t_1$ and $t = t_{n-1}$. If $\mathbf{S}_1 \equiv \mathbf{S}_2$, then this means that the third derivatives of the two curves match at $t = t_1$, so we have $\mathbf{S}_1'''(t_1) = \mathbf{S}_2'''(t_1)$. Using **equation (18.8)**, this gives us

$$12\lambda_1^3(\mathbf{Q}_1 - \mathbf{Q}_0) + 6\lambda_1^2(\mathbf{V}_0 + \mathbf{V}_1) = 12\lambda_2^3(\mathbf{Q}_2 - \mathbf{Q}_1) + 6\lambda_2^2(\mathbf{V}_1 + \mathbf{V}_2)$$

and rearranging gives

$$\lambda_1^2\mathbf{V}_0 + (\lambda_1^2 - \lambda_2^2)\mathbf{V}_1 - \lambda_2^2\mathbf{V}_2 = 2\lambda_2^3(\mathbf{Q}_2 - \mathbf{Q}_1) - 2\lambda_1^3(\mathbf{Q}_1 - \mathbf{Q}_0)$$

Equating third derivatives of $\mathbf{S}_{n-1}$ and $\mathbf{S}_n$ gives a similar equation involving $\mathbf{V}_{n-2}$, $\mathbf{V}_{n-1}$, and $\mathbf{V}_n$. These two equations can then be appended to the linear system (18.13).

18.15 Solving the Linear System

In the previous section, we saw three different ways to append two further equations to the system (18.13). Using any one of these three techniques, we eventually get a system of $n+1$ linear equations for the $n+1$ unknowns $\mathbf{V}_0, \ldots, \mathbf{V}_n$.

Regardless of which end conditions we use, we obtain a linear system whose matrix $\mathbf{M}$ is invertible. We can prove this by observing that the matrix is "strictly diagonally dominant". Since $\mathbf{M}$ is invertible, the linear system has a unique solution. So, given the points $\mathbf{Q}_0, \ldots, \mathbf{Q}_n$, the parameter values $t_0, \ldots, t_n$, and two suitable end conditions, there is a *unique* cubic spline that solves our interpolation problem.

Furthermore, the matrix $\mathbf{M}$ is banded — the only non-zero elements occur in a band along the diagonal. A linear system with a banded matrix is especially easy to solve, and there are special numerical methods for this purpose. Using a general-purpose linear system solver will work, too, of course, but it may be significantly less efficient. This may or may not matter, depending on the number of points and the performance requirements.

Finally, note that (18.13) is actually a linear system in which the unknowns $\mathbf{V}_0, \ldots, \mathbf{V}_n$ are 3D vectors. So, in a sense, it is a collection of three linear systems, one for the x-components, one for y, and one for z. In practice, these three systems will be solved in three operations. But note that the matrix $\mathbf{M}$ on the left-hand-side is the same in all three cases, which means that some of the solving calculations can be shared. For example, if we solved by computing the inverse of $\mathbf{M}$ (which is a very bad idea, as always), this computation would only need to be done once.

Even though various shortcuts are available, as outlined above, solving a large linear system is a lot of work, compared with the simple computations required in the Catmull-Rom algorithm from **section (18.12)**. What we get, in return for all our hard work, is C_2 continuity, which may or may not be valuable, depending on the application.

18.16 Editing Behaviour

After adding the two equations obtained from end conditions, the linear system (18.13) becomes

$$\mathbf{M}\begin{bmatrix}\mathbf{V}_0 & \mathbf{V}_1 & \cdots & \mathbf{V}_n\end{bmatrix}^\mathsf{T} = \begin{bmatrix}\mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_n\end{bmatrix}^\mathsf{T}$$

where $\mathbf{M}$ is an invertible matrix with size $(m+1)\times(m+1)$. Though this is not a good approach for computational purposes, the solution can be written as

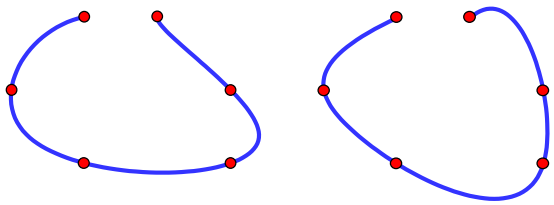
$$\begin{bmatrix}\mathbf{V}_0 & \mathbf{V}_1 & \cdots & \mathbf{V}_n\end{bmatrix}^\mathsf{T} = \mathbf{M}^{-1}\begin{bmatrix}\mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_n\end{bmatrix}^\mathsf{T}$$

Changing any point $\mathbf{Q}_i$ will change three right-hand side values $\mathbf{D}_{i-1}$, $\mathbf{D}_i$, $\mathbf{D}_{i+1}$. Generally, $\mathbf{M}^{-1}$ might have non-entries anywhere, so changing any value $\mathbf{D}_i$ might potentially change *any* (or even all) of $\mathbf{V}_0, \ldots, \mathbf{V}_n$.

So, moving a point $\mathbf{Q}_i$ might change the shape of the entire curve, and will require that the entire curve be recomputed. This global change of shape is very different from the local behaviour of a spline computed by the Catmull-Rom algorithm.

18.17 Choosing Parameter Values

In the example in **section (??)**, we saw that the parameter values the parameter values that we assign to the data points can have a significant effect on the shape of the resulting interpolating curve. Two examples with the same data points but different parameter values are shown below:



Computationally, the simplest approach is to set $t_i = i$, so the parameter values are just consecutive integers. This is called a **uniform** parameterization. If we do this, then the matrix $\mathbf{M}$ in our linear system is basically just

$$\begin{bmatrix} 1 & 4 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 4 & 1 \end{bmatrix} \tag{18.13}$$

with some adjustments for end conditions. This matrix is fixed, for any given n, so it can be inverted symbolically and saved. Then, we use

$$\begin{bmatrix} \mathbf{V}_0 & \mathbf{V}_1 & \cdots & \mathbf{V}_n \end{bmatrix}^{\mathsf{T}} = \mathbf{M}^{-1} \begin{bmatrix} \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_n \end{bmatrix}^{\mathsf{T}}$$

to obtain $\mathbf{V}_0, \ldots, \mathbf{V}_n$ for any given points $\mathbf{Q}_0, \ldots, \mathbf{Q}_n$.

However, using equally spaced parameter values does not work very well unless the points $\mathbf{Q}_0, \ldots, \mathbf{Q}_n$ are spaced fairly evenly. For points with uneven separation distances, we need to use parameter values that are somehow related to these distances. A common technique is to set

$$t_i - t_{i-1} = h_i = \|\mathbf{P}_i - \mathbf{P}_{i-1}\|$$

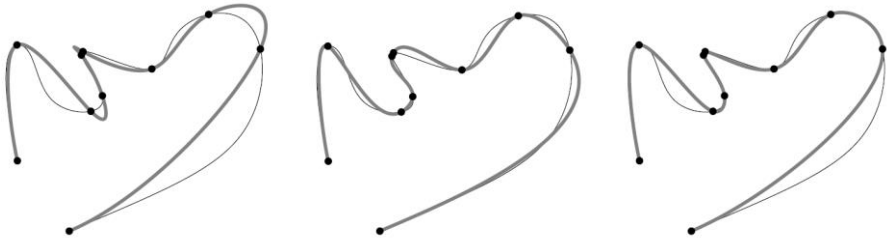
In other words, the parameter distance h_i is equal to the length of the chord $\mathbf{P}_{i-1}\mathbf{P}_i$. This is called **chordal** parameterization. Another common option is to set

$$t_i - t_{i-1} = h_i = \sqrt{\|\mathbf{P}_i - \mathbf{P}_{i-1}\|}$$

which is called the **centripetal** parameterization. In fact, all three of these parameterization schemes are special cases of

$$t_i - t_{i-1} = h_i = \|\mathbf{P}_i - \mathbf{P}_{i-1}\|^r$$

where $r = 0$, or $r = 1$ or $r = \frac{1}{2}$. There is no "right" or "best" parameterization, so it is impossible to make an intelligent choice. The following picture shows some sample results



(a) uniform (b) chordal (c) centripetal