# Lecture 11: Introduction to Spline Curves

Splines are used in graphics to represent smooth curves and surfaces. They use a small set of control points (knots) and a function that generates a curve through those points. This allows the creation of complex smooth shapes without the need for manipulating many short line segments or polygons at the cost of a little extra computation time when the objects of a scene are being designed. We will start with a simple, but not very useful spline. Taking the equation $y = f(x)$, we can express $f$ as a polynomial function, say:

$$y = a_2x^2 + a_1x + a_0$$

Let us choose any three points $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ at coordinates $(x_0, y_0)$, $(x_1, y_1)$ and $(x_2, y_2)$. We can substitute each set of coordinates into the equation to get three simultaneous equations which we can solve for the unknowns $a_2$, $a_1$ and $a_0$. We now have the equation of a curve that *interpolates* (passes through) the three chosen points. It is of course a parabola, or parabolic spline. Notice that we don't have any control over the curve. There is only one parabola that will fit the data as shown in Figure 1.

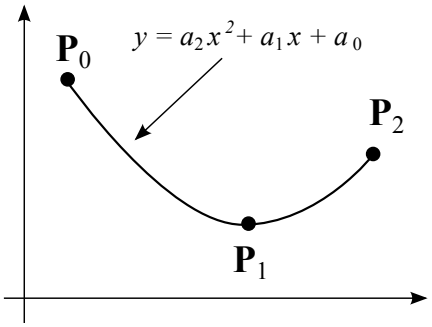


Figure 1: A non-parametric spline

## Parametric Splines

We can improve our choice by using a *parametric* spline. Let us consider first a quadratic polynomial spline written in vector notation as.

$$\mathbf{P} = \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0 \tag{1}$$

This equation gives the vector location of a point $\mathbf{P}$ in terms of constant vectors $\mathbf{a}_0$, $\mathbf{a}_1$ and $\mathbf{a}_2$ and a scalar parameter $\mu$. The locations of $\mathbf{a}_0$, $\mathbf{a}_1$ and $\mathbf{a}_2$ determine the shape of the spline. For two dimensional curves we now therefore have six unknowns (rather than the three in the parabola above). We can use these extra degrees of freedom to control the shape of the curve.

We will use the convention that $0 \leq \mu \leq 1$ over the range of interest. Hence at $\mu = 0$ the curve is at the first point to be interpolated, and at $\mu = 1$ it is at the last. Now consider interpolating the same three points as before: ($\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ at $(x_0, y_0)$, $(x_1, y_1)$ and $(x_2, y_2)$). When $\mu = 0$, the curve passes through the first point, say $\mathbf{P}_0$, and so, substituting $\mu = 0$ into Equation 1, we can write $\mathbf{P}_0 = \mathbf{a}_0$. Similarly, when $\mu = 1$ the point passes through the last point, say $\mathbf{P}_2$, and this gives us the equation:

$$\mathbf{P}_2 = \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0,$$

substituting for $\mathbf{a}_0$ we get

$$\mathbf{P}_2 - \mathbf{P}_0 = \mathbf{a}_2 + \mathbf{a}_1. \tag{2}$$

We have now met all the conditions except for the requirement that the curve must pass through $\mathbf{P}_1$. We can choose $\mu$ anywhere in the range $0 < \mu < 1$, and get a third equation to solve for the curve parameters. In other words we can now pick one of a whole *family* of different curves: Each one will interpolate the three points and have its own particular value of $\mu$ that we can choose for $\mathbf{P}_1$ . For example, choosing $\mu = 1/2$ in Equation 1 and substituting $\mathbf{P}_0$ for $\mathbf{a}_0$, we get

$$\mathbf{P}_1 - \mathbf{P}_0 = \mathbf{a}_2/4 + \mathbf{a}_1/2 \tag{3}$$

We can now solve equations 2 and 3 and find the values of $\mathbf{a}_1$ and $\mathbf{a}_2$. We can then draw the curve, for this choice of $\mu$ at $\mathbf{P}_1$, as shown on the left in Figure 2. Further possible curves using the same three points and parametric equation are also shown in Figure 2, each curve has its own set of choices of parameter $\mu$ for the given points..

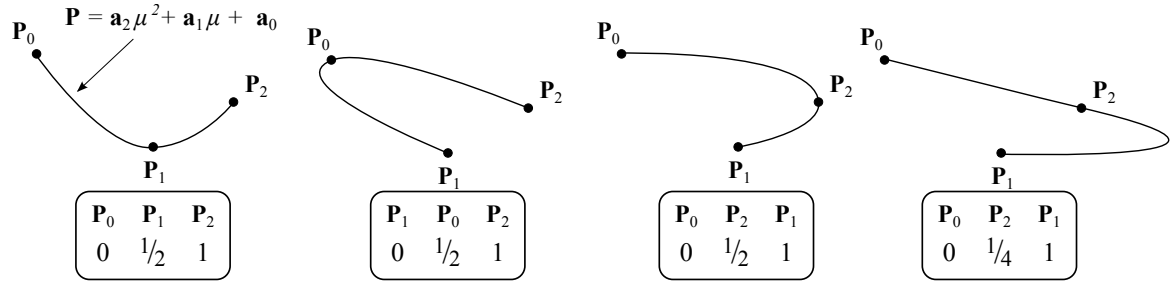


Figure 2: Four possible parametric splines that can pass through three points. The points are at the same locations each time but different choices of parameter values for the points can be made and this changes the shape of the curve. The parameter choices are shown in the boxes.

## Spline Patches

Although we have gained more freedom by using the parametric form, we do not have any intuitive way of using it. That is to say, we have no simple way to choose $\mu$ values for each point to get the type of interpolating spline we want. Moreover, we still face the problem of having to use ever higher degrees of polynomials for higher numbers of points. To overcome these difficulties we introduce a method based on *spline patches* that allows simple intuitive spline construction. With spline patches, we define a different curve between each pair of adjacent knots, as shown in Figure 3.

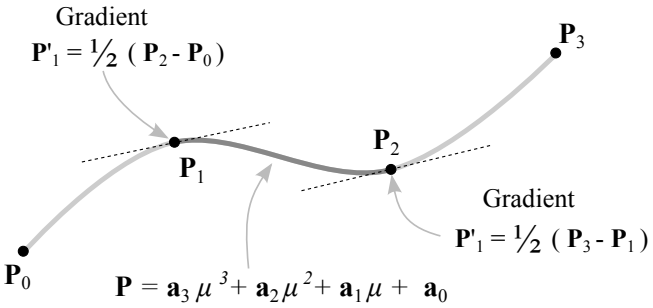


Figure 3: Using spline patches, each pair of adjacent knots is joined by a different curve. We then need to join them together. The above diagram shows one way of doing this.

It is common to implement spline patches using a cubic spline for each patch, rather than the quadratic splines formulated above. The reason for choosing a cubic form is so that we can *join* the patches smoothly together. The equation for a parmetric cubic spline patch has four unknowns, $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ and $\mathbf{a}_3$:

$$\mathbf{P} = \mathbf{a}_3\mu^3 + \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0 \tag{4}$$

We use the extra degrees of freedom provided by the cubic equation to control the gradient at the ends of the patch, and this makes it join seamlessly to its neighbours. Looking at Figure 3, we see that this can conveniently be done by making the gradient at a knot depend on the difference of the coordinate vectors on either side. This however is not the only way of setting this gradient, as we will see later. If we differentiate the curve equation we get:

$$\mathbf{P}' = 3\mathbf{a}_3\mu^2 + 2\mathbf{a}_2\mu + \mathbf{a}_1 \tag{5}$$

Now consider the two ends of a spline patch between $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$, where $\mu = 0$ at $\mathbf{P}_i$ and $\mu = 1$ $\mathbf{P}_{i+1}$. We can find the positions and gradients at each end by substituting $\mu = 0$ and $\mu = 1$ into Equations 4 and 5 which

gives:

$$\begin{aligned} \mathbf{P}_i &= \mathbf{a}_0 \\ \mathbf{P}'_i &= \mathbf{a}_1 \\ \mathbf{P}_{i+1} &= \mathbf{a}_3 + \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\ \mathbf{P}'_{i+1} &= 3\mathbf{a}_3 + 2\mathbf{a}_2 + \mathbf{a}_1 \end{aligned} \tag{6}$$

This is a system of vector equations as the left and right hand sides of each equation is a vector. For convenience, we can write this system of equations in matrix form if we form the matrices

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{P}_i \\ \mathbf{P}'_i \\ \mathbf{P}_{i+1} \\ \mathbf{P}'_{i+1} \end{pmatrix}$$

where the rows of the first matrix are the unknown coefficient vectors $\mathbf{a}_0, \ldots, \mathbf{a}_3$ and the rows of the second matrix are the known coordinates and gradients at $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$. In the 2D case, each of these is a $4 \times 2$ matrix. With these matrices, we can re-write the vector Equations 6 in the following matrix form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{P}_i \\ \mathbf{P}'_i \\ \mathbf{P}_{i+1} \\ \mathbf{P}'_{i+1} \end{pmatrix}$$

and since the values for $\mathbf{a}_0, \ldots, \mathbf{a}_3$ are unknown and the values of $\mathbf{P}$ and $\mathbf{P}'$ are known, we need to invert the matrix to solve for the parameters of the spline patch.

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_i \\ \mathbf{P}'_i \\ \mathbf{P}_{i+1} \\ \mathbf{P}'_{i+1} \end{pmatrix} \tag{7}$$

Remember that we have vector quantities in these equations, so, this formulation represents eight equations for the 2D case (data matrices are $2 \times 4$), and twelve for the 3D case (data matrices are $3 \times 4$). However, the matrix that we invert and multiply to get the solution is always the same, whether we have 2D or 3D data. For any given set of knots, this cubic patch method gives a stable, practical solution.

## Bezier Curves

One of the simplest ways of approximating a curve was made popular by the French mathematician Pierre Bezier in the context of car body design. It was based on a mathematical formulation by another French mathematician Paul de Casteljau. A typical Bezier curve is shown in Figure 4.

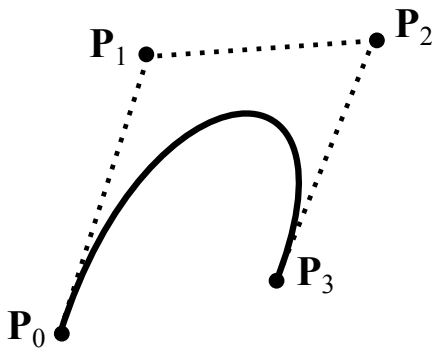


Figure 4: A typical Bezier Curve

The Bezier curve in Figure 4 is cubic and has four knots: $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$. The gradient of the Bezier curve at the start is obtained from the gradient of the line joining the first two knots, thus:

$$\mathbf{P}'_0 = k(\mathbf{P}_1 - \mathbf{P}_0) \tag{8}$$

where $k$ is the number of knots - 1, i.e. $k = 3$ in this example. A similar equation gives to the gradient at the end of the curve. This is an important property as it allows us to join Bezier patches together smoothly.

Computation of the Bezier Curve may be done in two ways. The first uses a recursive algorithm based on a method of de Casteljau. The idea is illustrated by Figure 5. The knot points, $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ are re-labelled $\mathbf{P}_{0,0}$, $\mathbf{P}_{0,1}$, $\mathbf{P}_{0,2}$ and $\mathbf{P}_{0,3}$, this will help represent the recursive nature of the method. First we have to choose a value of $\mu$, the illustration in Figure 5 uses a value of around 0.6.

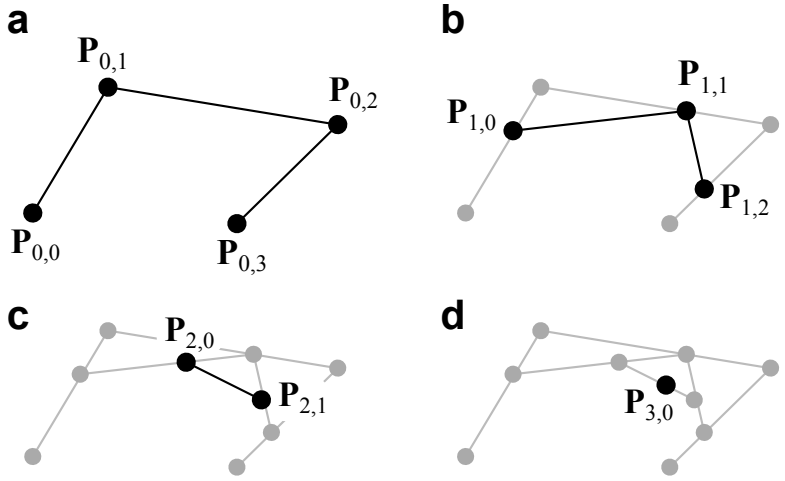


Figure 5: de Casteljau's construction to draw a point on a cubic Bezier curve (for a parameter value of approximately 0.6). a) The knot points. b), c) & d) The three steps to construct the point for the given parameter value.

Then we take the three lines $\mathbf{P}_{0,0}\,\mathbf{P}_{0,1}$, $\mathbf{P}_{0,1}\,\mathbf{P}_{0,2}$ and $\mathbf{P}_{0,2}\,\mathbf{P}_{0,3}$ and construct a point in each one a factor of $\mu$ along each line. These are labelled as the first set of constructed points, $\mathbf{P}_{1,0}$, $\mathbf{P}_{1,1}$ and $\mathbf{P}_{1,2}$ (Fig. 5b). The new points are joined up and the same process is followed to construct the second set of constructed points $\mathbf{P}_{2,0}$ and $\mathbf{P}_{2,1}$ (Fig. 5c). The process is repeated for a final time to find the point $\mathbf{P}_{3,0}$ and this is a point on the Bezier curve. As $\mu$ varies from 0 to 1, the locus of $\mathbf{P}_{3,0}$ traces out the Bezier Curve.

Using a functional pseudocode which allows us such liberties as scalar and vector multiplications and typed functions, this algorithm and another for drawing the curve are illustrated below. Note that here, and for all subsequent treatment of splines we will use a set of $N + 1$ knots, labelled 0 to $N$.

```
Point casteljau (knots P[ ] ; int k, int r, float μ)
begin
    if (r > 0)
    then casteljau = (1 − μ) ∗ casteljau(P, k, r - 1, μ) + μ ∗ casteljau(P, k + 1, r - 1, μ)
    else  casteljau = (1 − μ) ∗ P[ k ] + μ ∗ P[ k + 1 ]
end


void drawCurve (knots knotArray[], int L)
for j=0 to L
begin
    let μ = j/L
    point = casteljau(knotArray, 0, 3, μ)
    plot(point.x, point.y)
end
```

## Blending

Another way to compute a Bezier curve is by thinking of it as a *blend* of its knots. In the simplest case, if we apply the de Casteljau algorithm to the degenerate case, when the 'curve' is actually a straight line defined by two knots, we get the parametric line equation:

$$\mathbf{P} = \mu\mathbf{P}_{0,1} + (1-\mu)\mathbf{P}_{0,0}$$

This can be seen as linearly blending the two points to produce a third. We can think of the parameter $\mu$ as a measure of distance along the line. Like the curve constructed using the de Casteljau algorithm, most spline formulations consist of a blend of the positions of the knots. For more than two points the blend is expressed by the iterative formulation of the Bezier Curve:

$$\mathbf{P}(\mu) = \sum_{i=0}^{N} W(N,i,\mu)\,\mathbf{P}_i \tag{9}$$

where $W(N,i,\mu)$ is called the Bernstein blending function:

$$W(N,i,\mu) = \binom{N}{i}\mu^i(1-\mu)^{N-i} \qquad \text{and} \qquad \binom{N}{i} = \frac{N!}{(N-i)!i!}$$

As before we are using a parameter $\mu$ to determine the distance along the curve.

The general Bernstein blending function contains both the terms $\mu^i$ and $(1-\mu)^{N-i}$ but if $i=0$ or $i=N$ we have

$$\begin{aligned} W(N,0,\mu) &= \binom{N}{0}\mu^0(1-\mu)^{N-0} = (1-\mu)^N \\ W(N,N,\mu) &= \binom{N}{N}\mu^N(1-\mu)^{N-N} = \mu^N \end{aligned}$$

From these properties, it can be easily verified that when $\mu = 0$ or $\mu = 1$ the spline interpolates the end points, $\mathbf{P}(\mathbf{0}) = \mathbf{P}_0$ and $\mathbf{P}(\mathbf{1}) = \mathbf{P_N}$, and that when $0 < \mu < 1$ then $\mathbf{P}(\mu)$ is a blend of all the knots $\mathbf{P}_i$.

The first few expansions of the blending function for different values of $N$ are

| $N$ | Expansion |
|---|---|
| 1 | $(1-\mu)\mathbf{P}_0 + \mu\mathbf{P}_1$ |
| 2 | $(1-\mu)^2\mathbf{P}_0 + 2\mu(1-\mu)\mathbf{P}_1 + \mu^2\mathbf{P}_2$ |
| 3 | $(1-\mu)^3\mathbf{P}_0 + 3\mu(1-\mu)^2\mathbf{P}_1 + 3\mu^2(1-\mu)\mathbf{P}_2 + \mu^3\mathbf{P}_3$ |
| $\vdots$ | $\vdots$ |

The iterative equation for the Bezier curve can be computed slightly more efficiently in terms of space than the recursive form, though for most applications this is not likely to be significant, since it is rare to use Bezier curves for more than a few points. The iterative solution, though less elegant, generalises to surface construction more easily, and so tends to be used in preference.

## Characteristics of Bezier Curves

As previously mentioned, the gradient of a Bezier curve at each end is determined by the slope of the end line segments. At other points of the curve (not at the ends) it is a blend of the positions of all the points. Since Bezier Curves are a blend of all their control points there is little local control over a part of the curve. Figure 6 shows how a large number of control points tends to be ineffectual with Bezier curves. Moving the intermediate points has little effect, and it is not possible to create a curve which wiggles with any degree of complexity. This problem can be offset to some degree by pieceing together a number of sections, as we did for the cubic patches.

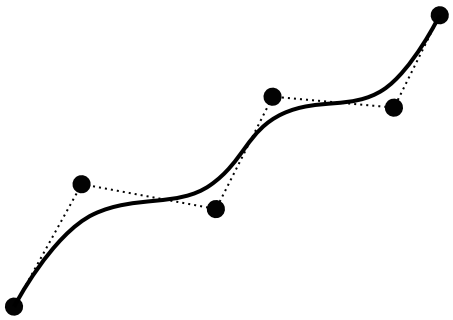

Figure 6: The lack of local detail in Bezier curves

## The Gradients at the spline ends

To find the gradient at any point of a parametric spline curve we differentiate it with respect to the parameter. Thus if we have the blending function for a four knot Bezier curve:

$$\mathbf{P}(\mu) = (1-\mu)^3\,\mathbf{P}_0 + 3\mu(1-\mu)^2\,\mathbf{P}_1 + 3\mu^2(1-\mu)\,\mathbf{P}_2 + \mu^3\,\mathbf{P}_3$$

we can differentiate it to get:

$$\mathbf{P}'(\mu) = -3(1-\mu)^2\,\mathbf{P}_0 + 3(1-\mu)^2\,\mathbf{P}_1 - 6\mu(1-\mu)\,\mathbf{P}_1 + 6\mu(1-\mu)\,\mathbf{P}_2 - 3\mu^2\,\mathbf{P}_2 + 3\mu^2\,\mathbf{P}_3$$

and by grouping the terms

$$\mathbf{P}'(\mu) = (1-\mu)^2\,(3\mathbf{P}_1 - 3\mathbf{P}_0) + \mu(1-\mu)\,(6\mathbf{P}_2 - 6\mathbf{P}_1) + \mu^2\,(3\mathbf{P}_3 - 3\mathbf{P}_2)$$

i.e.

$$\mathbf{P}'(\mu) = 3(\mathbf{P}_1 - \mathbf{P}_0)(1-\mu)^2 + 6(\mathbf{P}_2 - \mathbf{P}_1)\mu(1-\mu) + 3(\mathbf{P}_3 - \mathbf{P}_2)\mu^2$$

from which we get

$$\mathbf{P}'(0) = 3(\mathbf{P}_1 - \mathbf{P}_0) \qquad \text{and} \qquad \mathbf{P}'(1) = 3(\mathbf{P}_3 - \mathbf{P}_2)$$

So at the start, where $\mu = 0$, the gradient is in the same direction as $\mathbf{P}_1 - \mathbf{P}_0$ and at the end, where $\mu = 1$, the gradient is parallel to $\mathbf{P}_3 - \mathbf{P}_2$. This confirms that the curve is tangential to $\mathbf{P}_1 - \mathbf{P}_0$ at the start and $\mathbf{P}_3 - \mathbf{P}_2$ at the end as illustrated in Figure 4. Each gradient is a multiple of 3 (one fewer than the number of knots) times the corresponding vector between the adjacent points as described in Equation 8.

## Bezier Curves are Cubic Spline Patches

### The iterative form

We will now show that Bezier curves are cubic spline patches by modifying the iterative form of the Bezier curve. You may have already noticed that the *order* of a Bezier curve is one less than the number of its knots. In the earlier example there were four knots and this gives a cubic (order 3) spline. This can be verified by expanding the iterative form of the Bezier curve. Assume we have four knots $\mathbf{P}_i$, for $i = 0, 1, 2, 3$

$$\mathbf{P}(\mu) = \sum_{i=0}^{3} W(3, i, \mu)\mathbf{P}_i$$

$$\mathbf{P}(\mu) = (1-\mu)^3\mathbf{P}_0 + 3\mu(1-\mu)^2\mathbf{P}_1 + 3\mu^2(1-\mu)\mathbf{P}_2 + \mu^3\mathbf{P}_3$$

If we multiply out the brackets and collect the terms we get:

$$\mathbf{P}(\mu) = \mathbf{a}_3\mu^3 + \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0 \tag{10}$$

where

$$\begin{aligned} \mathbf{a}_0 &= \mathbf{P}_0 \\ \mathbf{a}_1 &= 3\mathbf{P}_1 - 3\mathbf{P}_0 \\ \mathbf{a}_2 &= 3\mathbf{P}_2 - 6\mathbf{P}_1 + 3\mathbf{P}_0 \\ \mathbf{a}_3 &= \mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0 \end{aligned} \tag{11}$$

If we substitute $\mu = 0$ and $\mu = 1$ into Equation 10 we get the end points, so

$$\mathbf{P}_0 = \mathbf{P}(0) = \mathbf{a}_0 \qquad \text{and} \qquad \mathbf{P}_3 = \mathbf{P}(1) = \mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3$$

As shown above, we have expressions for the gradient at the endpoints of

$$\mathbf{P}'(0) = 3\mathbf{P}_1 - 3\mathbf{P}_0 \qquad \text{and} \qquad \mathbf{P}'(1) = 3\mathbf{P}_3 - 3\mathbf{P}_2$$

With these choices of points and gradients at each end it is possible to show that the matrix form for the cubic spline patch in Equation 7 is equivalent to the set of equations (11) for the parameters by writing the following and simplifying

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ 3\mathbf{P}_1 - 3\mathbf{P}_0 \\ \mathbf{P}_3 \\ 3\mathbf{P}_3 - 3\mathbf{P}_2 \end{pmatrix}$$

So, we see that the four point Bezier curve is just a cubic spline patch between $\mathbf{P}_0$ and $\mathbf{P}_3$. The remaining points, $\mathbf{P}_1$ and $\mathbf{P}_2$, are two shape control points which can be manipulated by the designer.

**The de Casteljau form**

We now show that the de Casteljau construction can also be shown to be equivalent to a cubic spline patch. To do this we expand the the recursion backwards:

$$\begin{aligned} \mathbf{P}_{3,0} &= (1-\mu)\mathbf{P}_{2,0} + \mu\mathbf{P}_{2,1} \\ &= (1-\mu)\left\{(1-\mu)\mathbf{P}_{1,0} + \mu\mathbf{P}_{1,1}\right\} + \mu\left\{(1-\mu)\mathbf{P}_{1,1} + \mu\mathbf{P}_{1,2}\right\} \\ &= (1-\mu)^2\mathbf{P}_{1,0} + 2\mu(1-\mu)\mathbf{P}_{1,1} + \mu^2\mathbf{P}_{1,2} \\ &= (1-\mu)^2\left\{(1-\mu)\mathbf{P}_{0,0} + \mu\mathbf{P}_{0,1}\right\} \\ &\quad +2\mu(1-\mu)\left\{(1-\mu)\mathbf{P}_{0,1} + \mu\mathbf{P}_{0,2}\right\} \\ &\quad +\mu^2\left\{(1-\mu)\mathbf{P}_{0,2} + \mu\mathbf{P}_{0,3}\right\} \end{aligned}$$

To make things a little easier, we drop the first subscript, which indicated the construction recursion level of the de Casteljau algorithm, e.g. $\mathbf{P}_{0,2} \rightarrow \mathbf{P}_2$. This means we get:

$$\begin{aligned} \mathbf{P}(\mu) &= (1-\mu)^2\left\{(1-\mu)\mathbf{P}_0 + \mu\mathbf{P}_1\right\} \\ &\quad +2\mu(1-\mu)\left\{(1-\mu)\mathbf{P}_1 + \mu\mathbf{P}_2\right\} \\ &\quad +\mu^2\left\{(1-\mu)\mathbf{P}_2 + \mu\mathbf{P}_3\right\} \\ &= (1-\mu)^3\mathbf{P}_0 + 3\mu(1-\mu)^2\mathbf{P}_1 + 3\mu^2(1-\mu)\mathbf{P}_3 + \mu^3\mathbf{P}_3 \end{aligned}$$

This is the same as the blending function for four knot points which was shown above to be equivalent to the cubic spline patch.

## In summary . . .

So we can think of a four point Bezier curve as a cubic spline patch with shape control. The curve goes through points $\mathbf{P}_0$ and $\mathbf{P}_3$ and by picking up points $\mathbf{P}_1$ and $\mathbf{P}_2$ with a mouse and moving them we can control the shape as desired.