Lecture 9: 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 mainpulating 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:

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

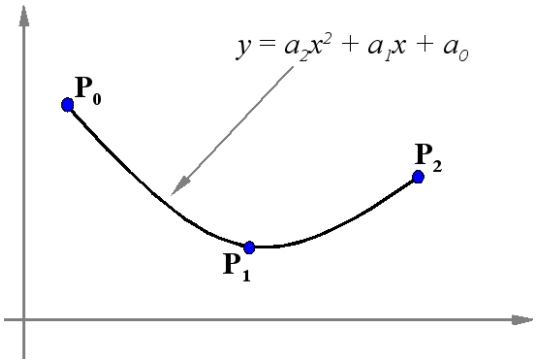

Figure 1: A non-parametric spline

If we now take any three points $[x_0, y_0]$, $[x_1, y_1]$ and $[x_2, y_2]$, we can substitute then 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 interpolating the three 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.

Parametric Splines

We can improve our choice by the simple expedient of using a parametric spline. Let us consider first a quadratic polynomial spline written in vector notation as:

$$\mathbf{P} = \mathbf{a_2}\mu^2 + \mathbf{a_1}\mu + \mathbf{a_0} \tag{1}$$

where $\mathbf{a_0}, \mathbf{a_1}$ and $\mathbf{a_2}$ are constant vectors whose values determine the shape of the spline. For two dimensional curves we now therefore have six unknowns (rather than the three previously). We can use these extra degrees of freedom to control the shape of the curve.

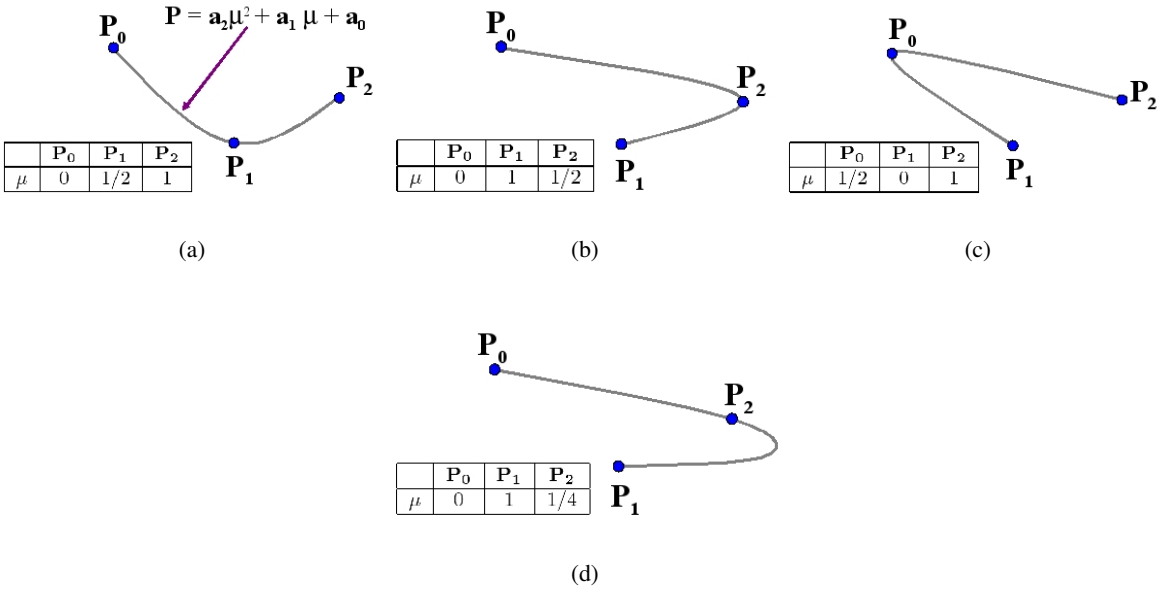

Figure 2: Possibilities using parametric splines

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 three points as before ($\mathbf{P_0} = [x_0, y_0], \mathbf{P_1} = [x_1, y_1]$ and $\mathbf{P_2} = [x_2, y_2]$). 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 our conditions except that the curve shall pass through $\mathbf{P_1}$. We can choose μ 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 family of curves interpolating the three points, by selecting the value of μ at $\mathbf{P_1}$. Choosing $\mu = 1/4$ we get

$$\mathbf{P_1} - \mathbf{P_0} = \mathbf{a_2}/16 + \mathbf{a_1}/4. \tag{3}$$

We can now solve equations 2 and 3 for the values of $\mathbf{a_1}$ and $\mathbf{a_2}$ and draw the curve as shown in Figure 2(d). Further possible curves using the same three points and parametric equation are shown in Figures 2(a), 2(b) and 2(c).

SplinePatches

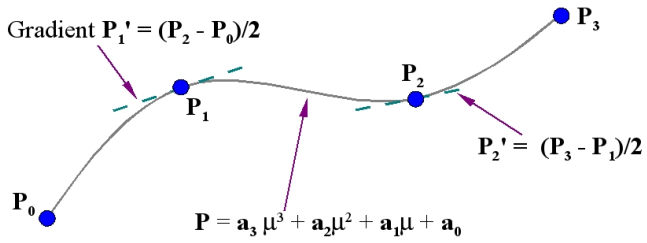

Figure 3: A simple way to join patches

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 μ 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. We define a different curve between each pair of adjacent knots, as shown in Figure 3. This is most commonly done by 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 set the gradient at either end of the patch, and thus make it join seamlessly to its neighbours. Looking at Figure 3, we see that this can conveniently be done by taking the difference of the coordinates on either side of the knot in question. 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 $\mu = 0$ or $\mu = 1$. We can find the positions and gradients at each end by substituting $\mu = 0$ and $\mu = 1$ into equations 4 and 5 which gives:

$\mathbf{P_i} = \mathbf{a_0}$
$\mathbf{P'_i} = \mathbf{a_1}$
$\mathbf{P_{i+1}} = \mathbf{a_3} + \mathbf{a_2} + \mathbf{a_1} + \mathbf{a_0}$
$\mathbf{P'_{i+1}} = \mathbf{3a_3} + \mathbf{2a_2} + \mathbf{a_1}$

We can write this system of equations in matrix form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{a_0} \\ \mathbf{a_1} \\ \mathbf{a_2} \\ \mathbf{a_3} \end{bmatrix} = \begin{bmatrix} \mathbf{P_i} \\ \mathbf{P'_i} \\ \mathbf{P_{i+1}} \\ \mathbf{P'_{i+1}} \end{bmatrix} \tag{6}$$

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

$$\begin{bmatrix} \mathbf{a_0} \\ \mathbf{a_1} \\ \mathbf{a_2} \\ \mathbf{a_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_i} \\ \mathbf{P'_i} \\ \mathbf{P_{i+1}} \\ \mathbf{P'_{i+1}} \end{bmatrix} \tag{7}$$

Notice that we have vector quantities, so, this formulation represents eight equations for the 2D case, and twelve for the 3D case. The matrix is the same for each dimension. 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. Here four knots $\mathbf{P_0}$, $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ are shown. The gradient at each end of the Bezier curve is the same as the gradient of the line joining the first two knots, thus: $\mathbf{P'_0} = k(\mathbf{P_1} - \mathbf{P_0})$, where k is the number of knots - 1. 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 Casteljau. The idea is illustrated by Figure 5. For a given value of μ, say 1/2 we first construct the points on the lines [$\mathbf{P_{0,0}}$, $\mathbf{P_{0,1}}$] [$\mathbf{P_{0,1}}$, $\mathbf{P_{0,2}}$] and [$\mathbf{P_{0,2}}$, $\mathbf{P_{0,3}}$] for the chosen value of μ. These are labelled as the first set of constructed points, $\mathbf{P_{1,0}}$, $\mathbf{P_{1,1}}$ and $\mathbf{P_{1,2}}$. The new points are joined up and the same procedure is followed to construct the second set of points $\mathbf{P_{2,0}}$ and $\mathbf{P_{2,1}}$. The process is repeated to find the point $\mathbf{P_{3,0}}$. This is a point on the Bezier curve. As μ 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 can be written very simply:

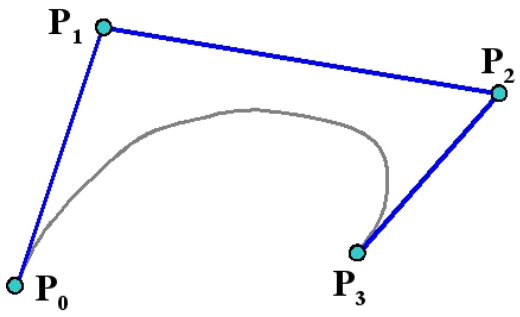

Figure 4: A typical Bezier Curve

```
Point Casteljau (knots P[] ; int N, int r, float μ)
begin
    if (r == 1)
    then Casteljau = μ*P[N+1] + (1-μ)*P[N]
    else Casteljau = μ*Casteljau(P,N+1,r-1,μ) + (1-μ)*Casteljau(P,N,r-1,μ)
end
```

and the curve can be drawn with:

```
for j=1 to L
begin
    locus=Casteljau(Knotarray,0,N,j/L)
    drawto(locus.x,locus.y)
end
```

Note that here, and for all subsequent treatment of splines we will use a set of N+1 knots, labelled 0 to N.

Blending

Another way to view a Bezier curve is to think of it as a blend of its knots. In the simplest case, if we apply the Casteljau algorithm to the degenerate case of two knots we get the parametric line equation:

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

We can think of this as linearly blending the two points to produce a third. The parameter μ may be thought of as measuring the distance along the line. Like the curve constructed using the 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:

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

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

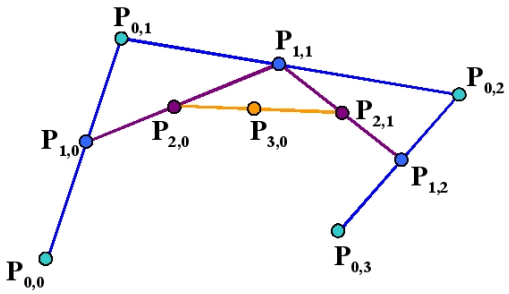

Figure 5: Using Casteljau's construction to draw a Bezier curve

$$W(N, I, \mu) = \binom{N}{i} \mu^i (1-\mu)^{N-i}$$

$$\binom{N}{i} = \frac{N!}{(N-i)!i!}$$

As before we are using a parameter μ to determine the distance along the curve, and it can be easily verified that when μ is 0 or 1 the spline interpolates the end points, $\mathbf{P(0)} = \mathbf{P_0}$ and $\mathbf{P(1)} = \mathbf{P_N}$, and that when $0 < \mu < 1$ then $\mathbf{P}(\mu)$ is a blend of all the knots $\mathbf{P_i}$.

The 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, Bezier curves have their end gradient clamped to the slope of the end line segments, and, beyond the ends they blend 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.

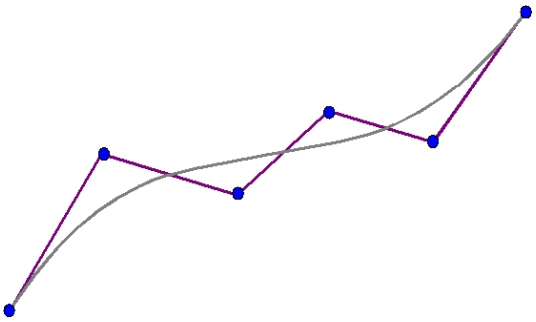

Figure 6: The lack of local detail in Bezier curves

Relation between Bezier Curves and Cubic Patches

We noted previously that the order of the Bezier curve is one less than the number of knots, so for four knots we have a cubic spline. This will be verified by expanding the iterative form of the Bezier curve:

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

$$\mathbf{P}(\mu) = \mathbf{P_0}(1-\mu)^3 + 3\mathbf{P_1}\mu(1-\mu)^2 + 3\mathbf{P_2}\mu^2(1-\mu) + \mathbf{P_3}\mu^3$$

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

$$\mathbf{P}(\mu) = \mathbf{a_3}\mu^3 + \mathbf{a_2}\mu^2 + \mathbf{a_1}\mu + \mathbf{a_0}$$

where
$$\mathbf{a_3} = \mathbf{P_3} - 3\mathbf{P_2} + 3\mathbf{P_1} - \mathbf{P_0}$$
$$\mathbf{a_2} = 3\mathbf{P_2} - 6\mathbf{P_1} + 3\mathbf{P_0}$$
$$\mathbf{a_1} = 3\mathbf{P_1} - 3\mathbf{P_0}$$
$$\mathbf{a_0} = \mathbf{P_0}$$

So, we see that the four point Bezier curve is just a cubic spline patch between $\mathbf{P_0}$ and $\mathbf{P_3}$, with two shape control points, $\mathbf{P_1}$ and $\mathbf{P_2}$, which are manipulated by the designer.

We can also show that the Casteljau construction results in the creation of a cubic spline patch. To do this we expand the the recursion backwards:

$$\begin{aligned}
\mathbf{P_{3,0}} &= \mu\mathbf{P_{2,1}} + (1-\mu)\mathbf{P_{2,0}} \\
&= \mu[\mu\mathbf{P_{1,2}} + (1-\mu)\mathbf{P_{1,1}}] + (1-\mu)[\mu\mathbf{P_{1,1}} + (1-\mu)\mathbf{P_{1,0}}] \\
&= \mu^2\mathbf{P_{1,2}} + 2\mu(1-\mu)\mathbf{P_{1,1}} + (1-\mu)^2\mathbf{P_{1,0}} \\
&= \mu^2[\mu\mathbf{P_{0,3}} + (1-\mu)\mathbf{P_{0,2}}] + 2\mu(1-\mu)[\mu\mathbf{P_{0,2}} + (1-\mu)\mathbf{P_{0,1}}] \\
&\quad +(1-\mu)^2[\mu\mathbf{P_{0,1}} + (1-\mu)\mathbf{P_{0,0}}]
\end{aligned}$$

if we drop the first subscript, which indicated the construction level of the Casteljau algorithm we get:

$$\begin{aligned}
\mathbf{P}(\mu) &= \mu^2[\mu\mathbf{P_3} + (1-\mu)\mathbf{P_2}] + 2\mu(1-\mu)[\mu\mathbf{P_2} + (1-\mu)\mathbf{P_1}] \\
&\quad +(1-\mu)^2[\mu\mathbf{P_1} + (1-\mu)\mathbf{P_0}] \\
&= \mu^3\mathbf{P_3} + 3\mu^2(1-\mu)\mathbf{P_3} + 3\mu(1-\mu)^2\mathbf{P_1} + (1-\mu)^3\mathbf{P_0}
\end{aligned}$$

which is the same as the blending formulation.

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.

The Gradients at the spline ends

To determine the gradient at any point of a parametric spline curve we differentiate it with respect to the parameter. Thus if:

$$\mathbf{P}(\mu) = \mathbf{P_0}(1-\mu)^3 + 3\mathbf{P_1}\mu(1-\mu)^2 + 3\mathbf{P_2}\mu^2(1-\mu) + \mathbf{P_3}\mu^3$$

we differentiate to get:

$$\mathbf{P}'(\mu) = -3\mathbf{P_0}(1-\mu)^2 + 3\mathbf{P_1}(1-\mu)^2 - 6\mathbf{P_1}\mu(1-\mu) + 6\mathbf{P_2}\mu(1-\mu) - 3\mathbf{P_2}\mu^2 + 3\mathbf{P_3}\mu^2$$

grouping the terms

$$\mathbf{P}'(\mu) = (3\mathbf{P_1} - 3\mathbf{P_0})(1-\mu)^2 + (6\mathbf{P_2} - 6\mathbf{P_1})\mu(1-\mu) + (3\mathbf{P_3} - 3\mathbf{P_2})\mu^2$$

So at the start, where $\mu = 0$, the gradient is $3\mathbf{P_1} - 3\mathbf{P_0}$ and at the end, where $\mu = 1$, the gradient is $3\mathbf{P_3} - 3\mathbf{P_2}$. 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.