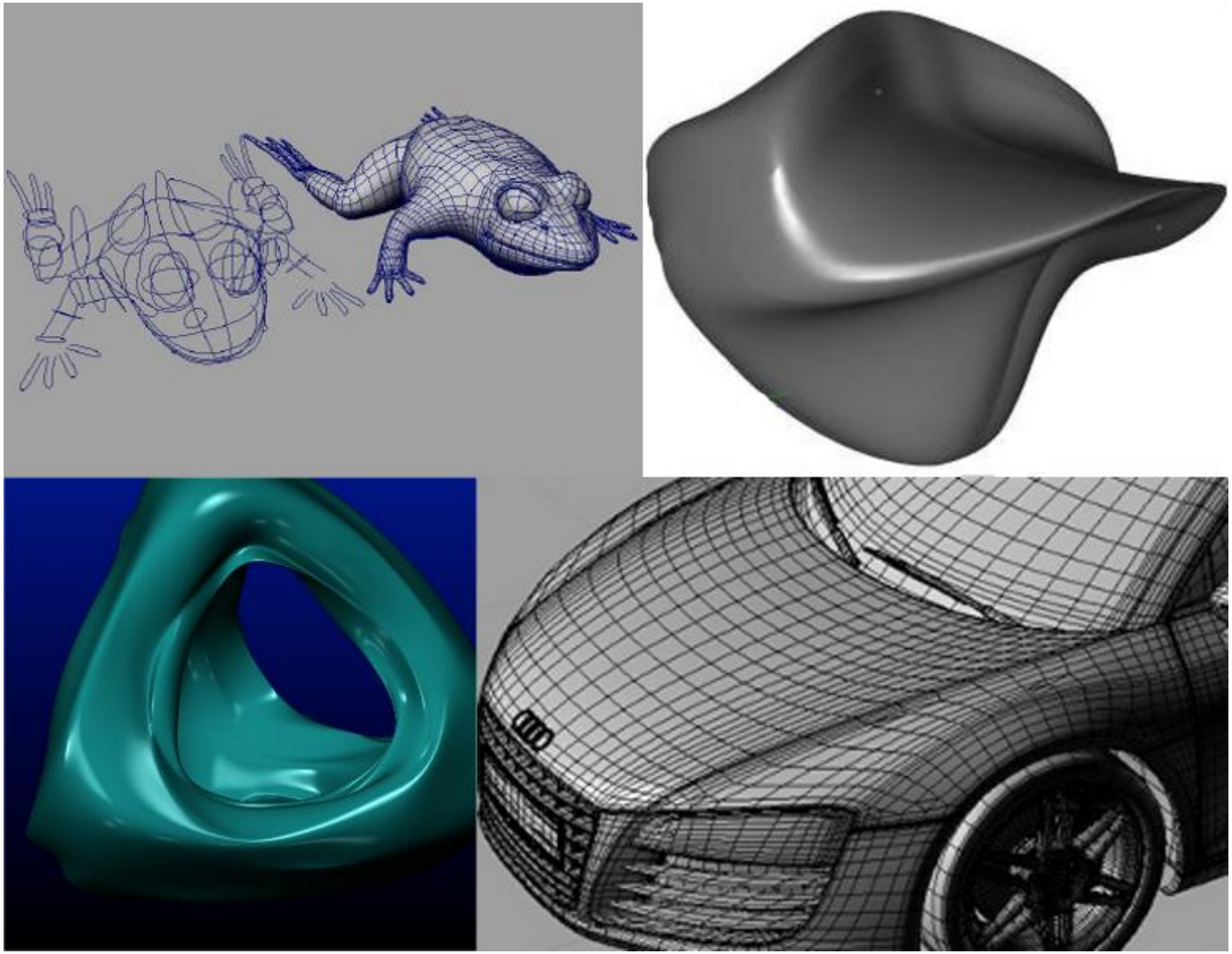
Interactive Computer Graphics: Lecture 12

Introduction to Spline Curves

Splines

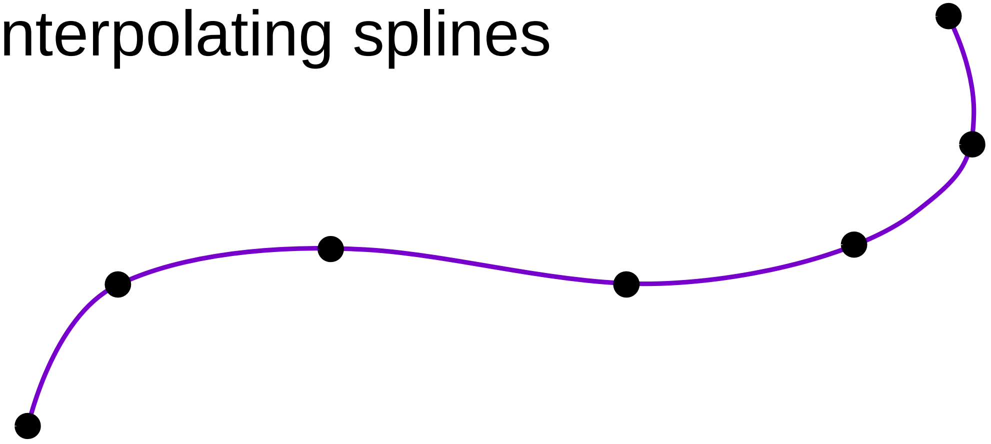


Splines

- The word spline comes from the ship building trade where planks were originally shaped by bending them round pegs fixed in the ground.
- Originally it was the pegs that were referred to as splines.

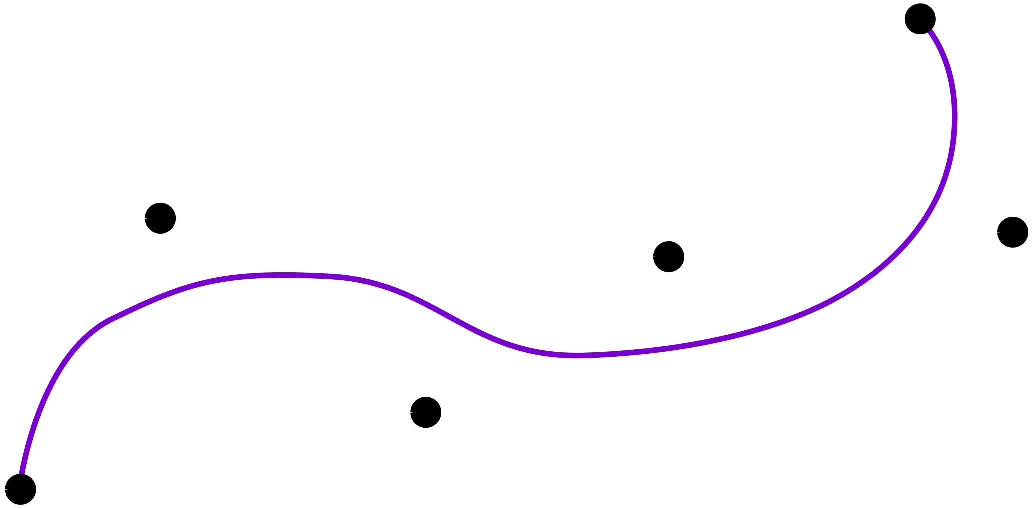
Interpolating Splines

- Modern splines are smooth curves defined from a small set of points often called *knots* or *control points*.
- In one main class of splines, the curve must pass through each point of the set.
- These are called interpolating splines



Approximating Splines

- In other cases the curves do not pass through the points.
- The points act as control points which the user can move to adjust the shape of the curve interactively



Non-Parametric Spline

- The simplest splines are just equations in x and y (for two dimensions)

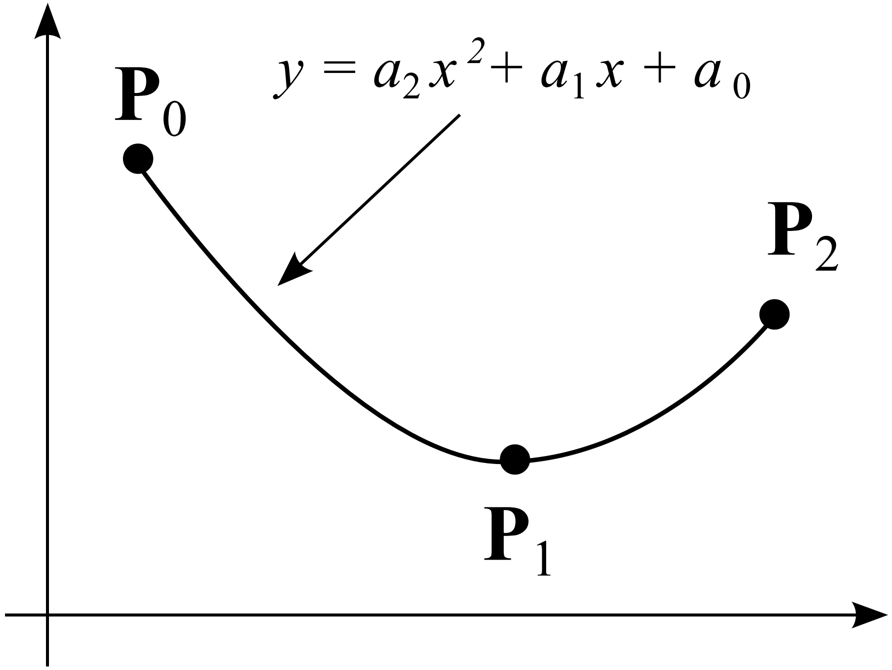
- The most common is the polynomial spline:

$$y = a_2 x^2 + a_1 x + a_0$$

- Given three points we can calculate a_2, a_1 and a_0

A Non-Parametric (Parabolic) Spline

- Example of a degree 2 (parabolic) non-parametric spline:



- There is no control using non parametric splines. Only one curve (a parabola) fits the data.

Parametric Splines

- If we write our spline in a vector form we get:

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

which has a parameter μ

- By convention, as μ ranges from 0 to 1 the point $\mathbf{P}$ traces out a curve.

Calculating simple parametric splines

We can now solve for the vector constants $\mathbf{a}_0$, $\mathbf{a}_1$ and $\mathbf{a}_2$ as follows:

•Suppose we want the curve to start at point $\mathbf{P}_0$

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

•We have μ $= 0$ at the start so

$$\mathbf{P}_0 = \mathbf{a}_0$$

Calculating simple parametric splines

- Suppose we want the spline to end at $\mathbf{P}_2$
- We have that at the end $\mu = 1$
- Thus

$$\begin{aligned} \mathbf{P}_2 &= \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0 \\ &= \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\ \Rightarrow \mathbf{P}_2 &= \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{P}_0 \end{aligned}$$

Calculating simple parametric splines

- And in the middle (say $\mu = 1/2$) we want it to pass through $\mathbf{P}_1$

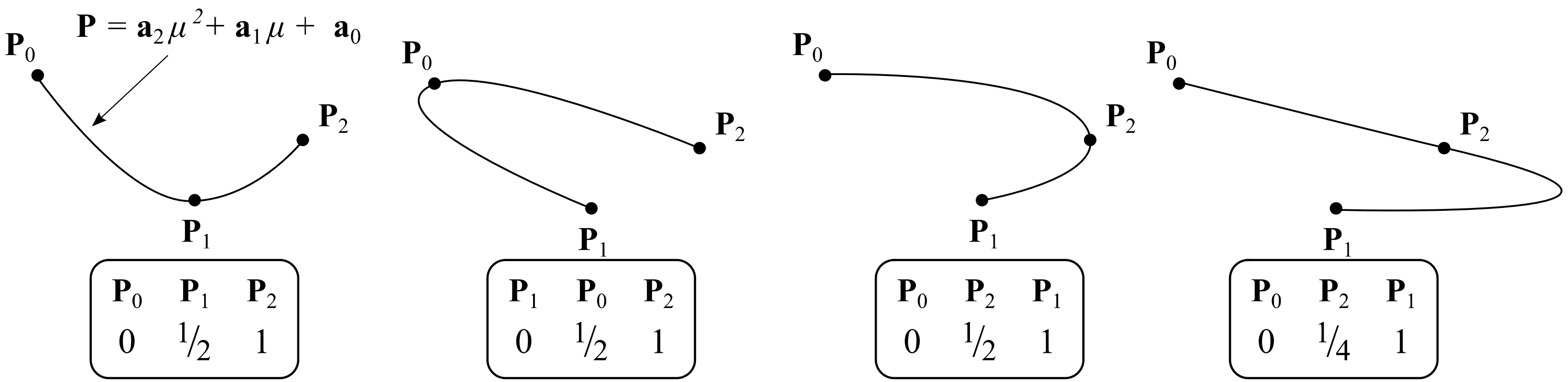
$$\mathbf{P}_1 = \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0$$

$$\Rightarrow \mathbf{P}_1 = \frac{1}{4}\mathbf{a}_2 + \frac{1}{2}\mathbf{a}_1 + \mathbf{P}_0$$

- We have enough equations to solve for $\mathbf{a}_1$ and $\mathbf{a}_2$.

- Notice that the method is the same whether we are working in 2 or 3 dimensions, we just have to solve separately for each of the ordinates in the vectors $\mathbf{a}_1$ and $\mathbf{a}_2$.

Possibilities using parametric splines



Higher order parametric splines

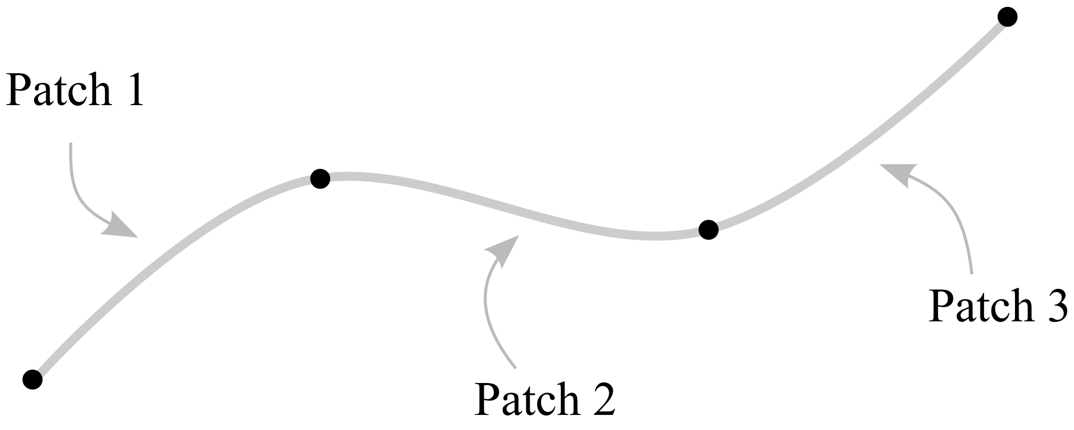
- Parametric polynomial splines must have an order to match the number of knots.

  - 3 knots - quadratic polynomial
  - 4 knots - cubic polynomial
  - etc.

- Higher order polynomials are undesirable since they tend to oscillate

Spline Patches

- To get round the problem, we can piece together a number of patches, each patch being a parametric spline.



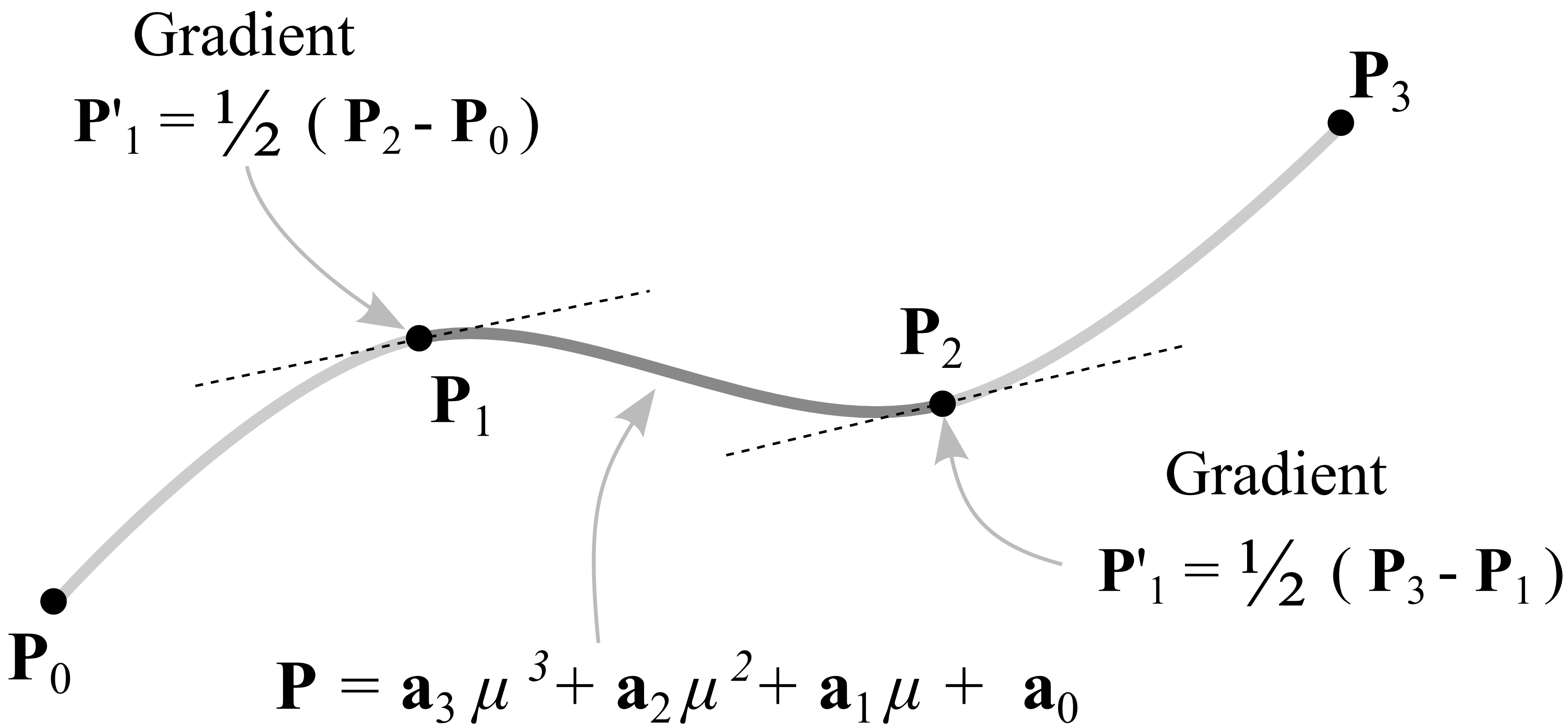
Cubic Spline Patches

- The simplest, and most effective way to calculate parametric spline patches is to use a cubic polynomial.

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

- This allows us to join the patches together smoothly

Choosing the gradients



Indices here are $\{0, 1, 2, 3\}$ but can be any successive set of four numbers taken from the available control points

Calculating a Cubic Spline Patch

- Each patch has the form: $\mathbf{P} = \mathbf{a}_3\mu^3 + \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0$

- For the patch which joins points $\mathbf{P}_{\mathrm{i}}$ and $\mathbf{P}_{\mathrm{i+1}}$, we have

$$\mu = 0 \text{ at } \mathbf{P}_i$$
$$\mu = 1 \text{ at } \mathbf{P}_{i+1}$$

- Substituting these values we get

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

Calculating a Cubic Spline Patch

- Differentiating $\mathbf{P} = \mathbf{a}_3\mu^3 + \mathbf{a}_2\mu^2 + \mathbf{a}_1\mu + \mathbf{a}_0$ we get

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

- Substituting for $\mu = 0$ at $\mathbf{P}_{\mathrm{i}}$ and $\mu = 1$ at $\mathbf{P}_{\mathrm{i+1}}$ we get

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

Calculating a Cubic Spline Patch

- Putting these four equations into matrix form we get:

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

- The initial matrix is always the same whether the points $\mathbf{P}$ are in 2-D or in 3-D

Calculating a Cubic Spline Patch

- Finally, inverting the matrix gives us the values of $\mathbf{a}_0, \ldots,$ $\mathbf{a}_3$ that we want

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

- Notice that the matrix is the same
  - for every patch
  - whether the data are 2-D, 3-D, ...

Parametric and Geometric Continuity

- We want to create smooth and realistic shapes.
  - What exactly do we mean by "smooth"?
  - How precisely do we determine if a given curve or surface is smooth?
- Recall that a parametric curve is defined as:

$$\boldsymbol{P}(\mu) = \begin{pmatrix} x(\mu) \\ y(\mu) \end{pmatrix}$$

- Generally a function is smooth if its derivatives are well-defined up to some order. There are actually two definitions for curves and surfaces, depending on whether the curve or surface is viewed as a function or purely a shape.

Parametric Continuity

- For parametric continuity, we view the curve or surface as a function rather than a shape.
  - A junction between two curves is said to be C^0 continuous if the (x, y) values of the two curves agree.
  - A junction between two curves is said to be C^1 continuous if the (x, y) values of the two curves agree, and all their first derivatives $(dx/ds, dy/ds)$ agree at their junction.
  - A junction between two curves is said to be C^2 continuous if the (x, y) values of the two curves agree, and their first and second parametric derivatives all agree at their junction.

Geometric Continuity

- Geometric continuity can be defined using only the shape of the curve (parametrization does not affect the outcome):
  - A junction between two curves is said to be G^0 continuous if the (x, y) values of the two curves agree. Same as C^0 continuity.
  - A junction between two curves is said to be G^1 continuous if the (x, y) values of the two curves agree, and all their first derivatives $(dx/ds, dy/ds)$ are proportional (the tangent vectors are parallel) at their junction.
  - Higher order geometric continuity is a bit tricky to define.

Bezier Curves

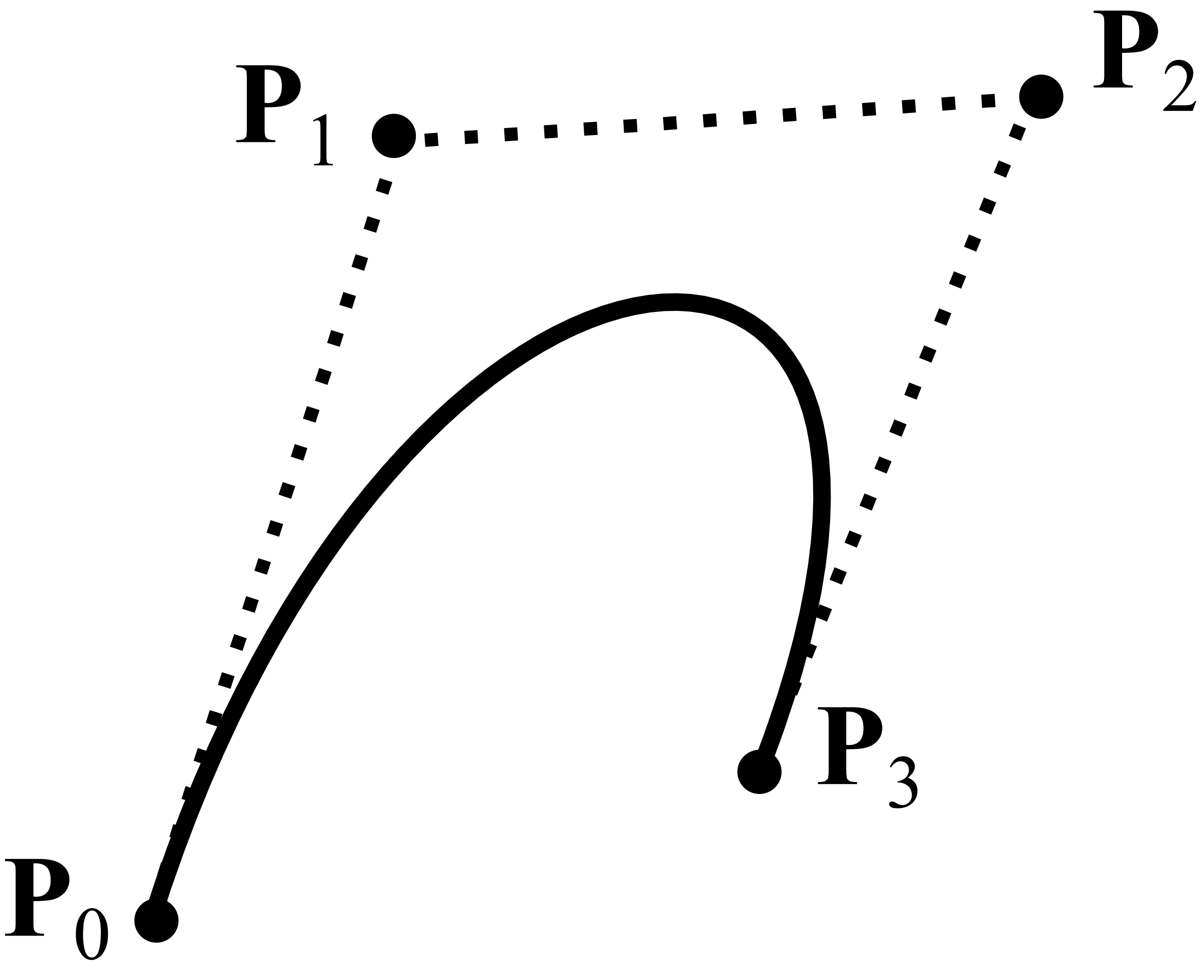
- Bezier curves were developed as a method for CAD design. They give very predictable results for small sets of knots, and so are useful as spline patches.

- The main characteristics of Bezier curves are

  – They interpolate the end points

  – The slope at an end is the same as the line joining the end point to its neighbour

A typical Bezier Curve



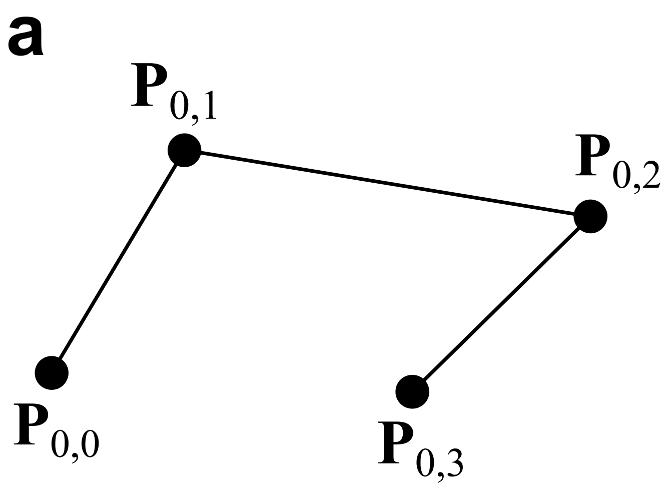
Casteljau's Algorithm

- Bezier curves may be computed and visualised using a geometric construction introduced by Paul de Casteljau.

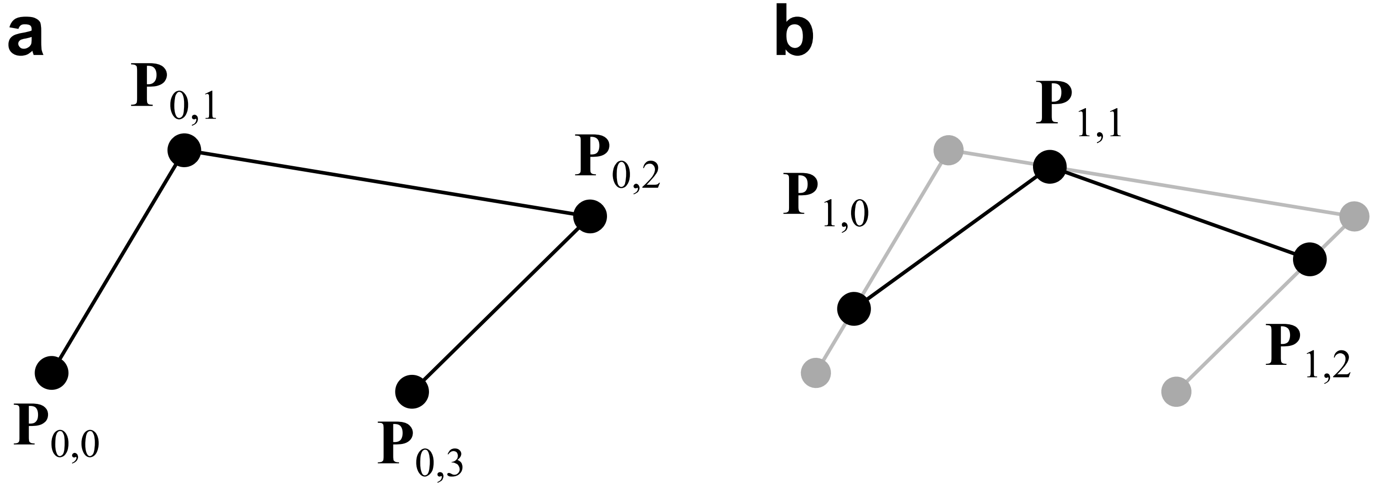
- Like a cubic patch, we need a parameter μ which is
  - 0 at the start of the curve
  - 1 at the end.

- The construction
  - is recursive
  - can be made for any value of μ

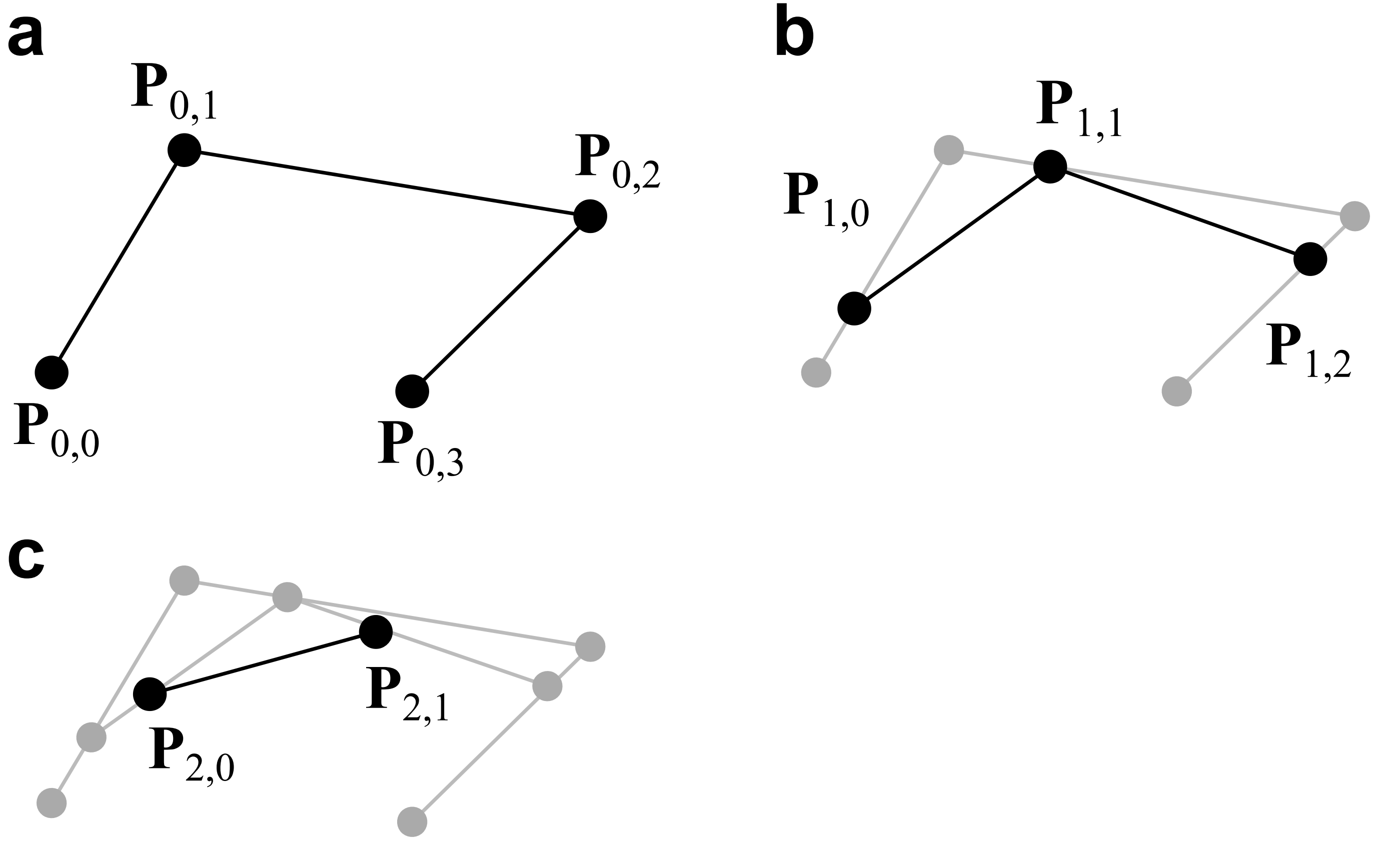
Casteljau's Construction $\mu = 0.25$



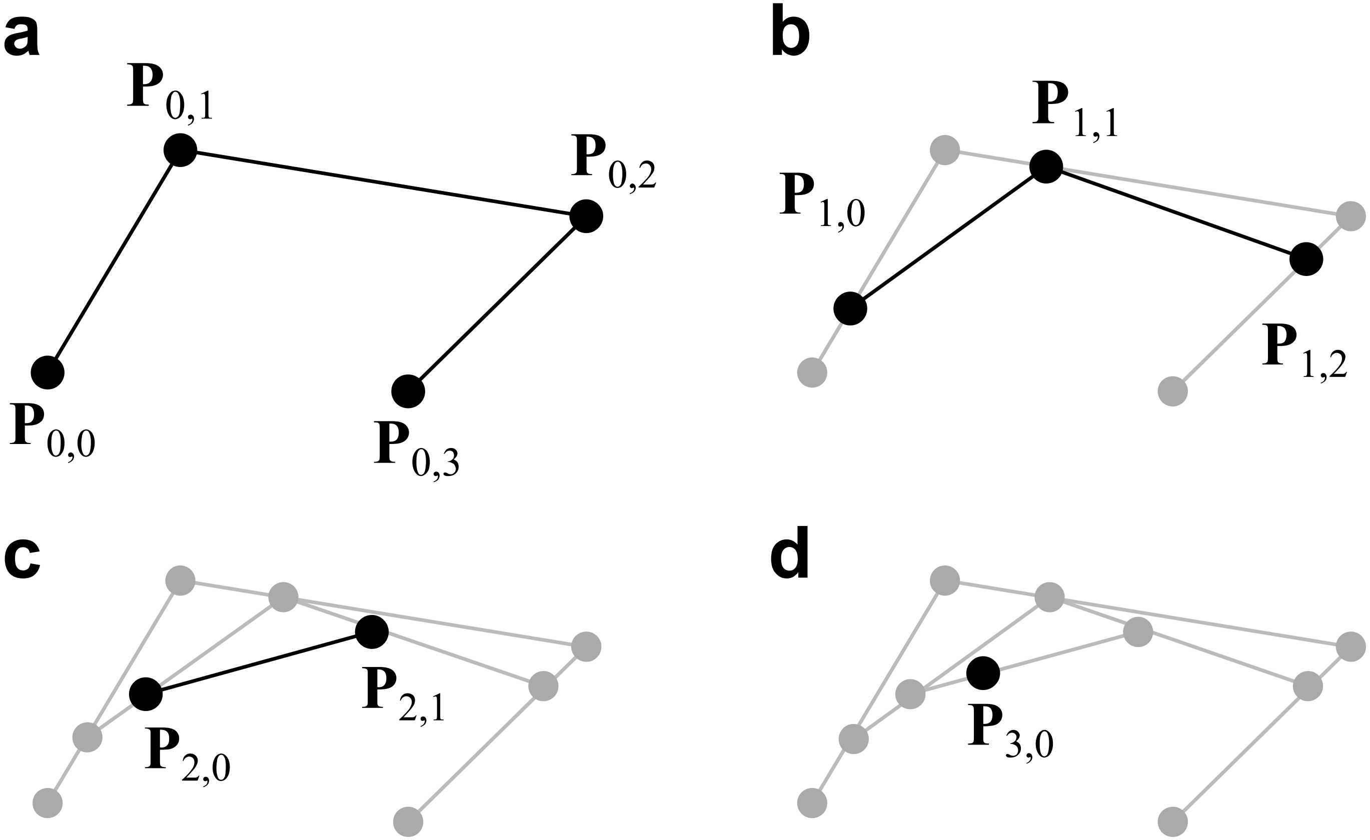
Casteljau's Construction μ = 0.25



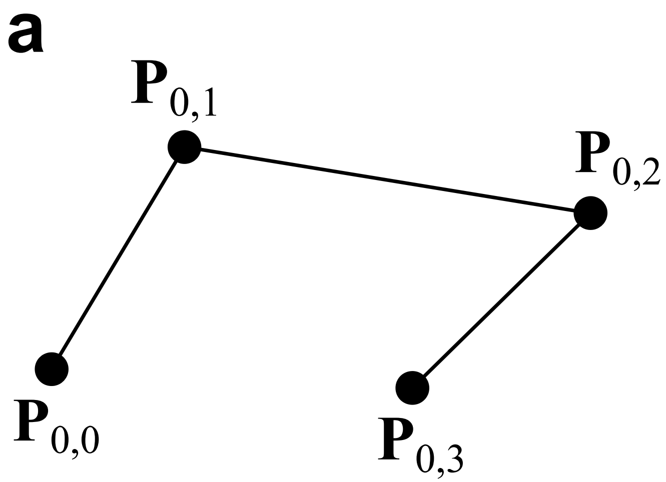
Casteljau's Construction $\mu = 0.25$



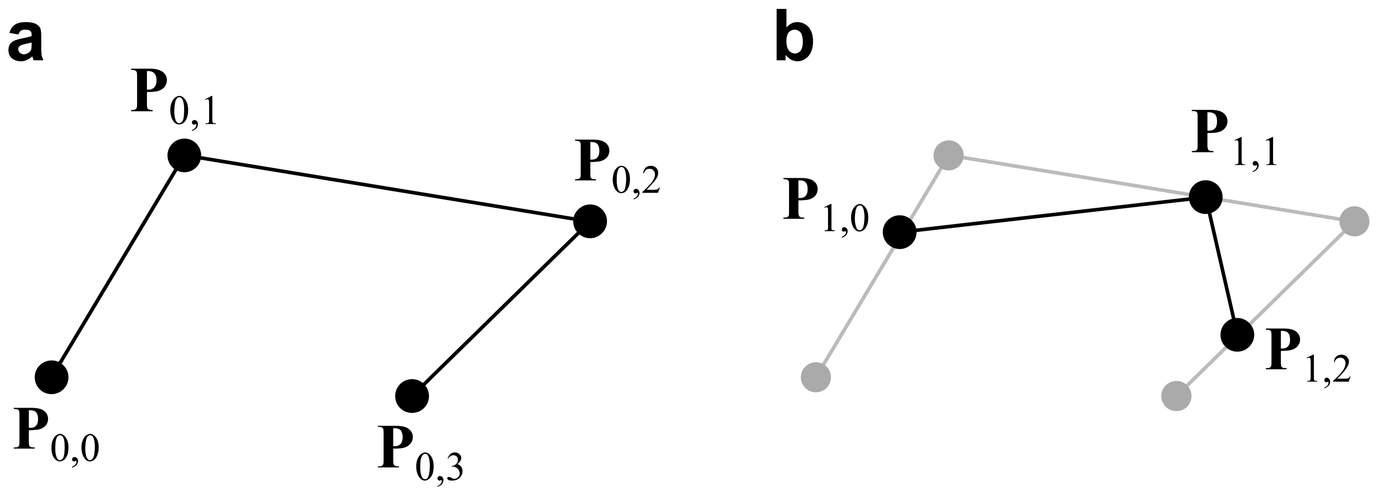
Casteljau's Construction $\mu = 0.25$



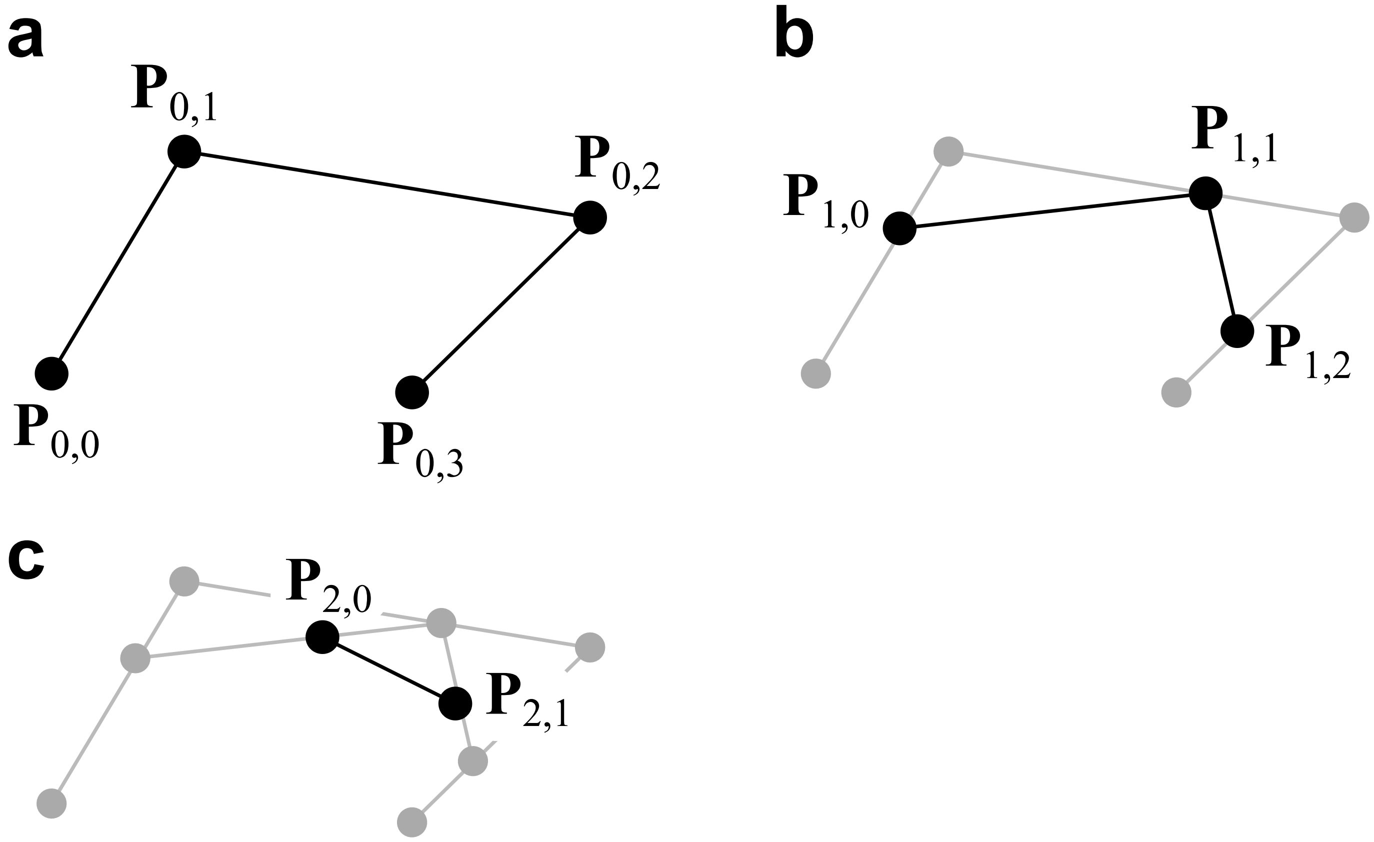
Casteljau's Construction $\mu = 0.6$



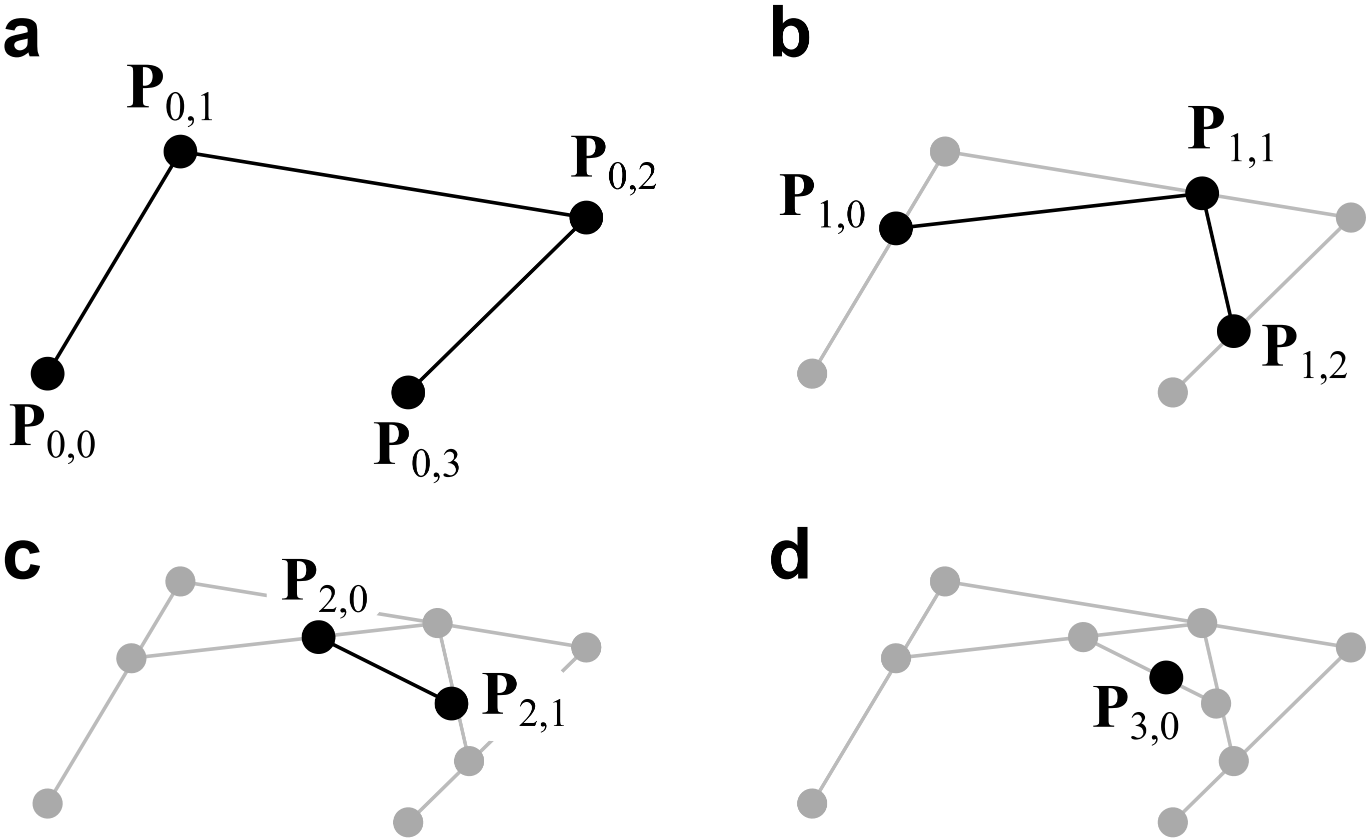
Casteljau's Construction $\mu = 0.6$



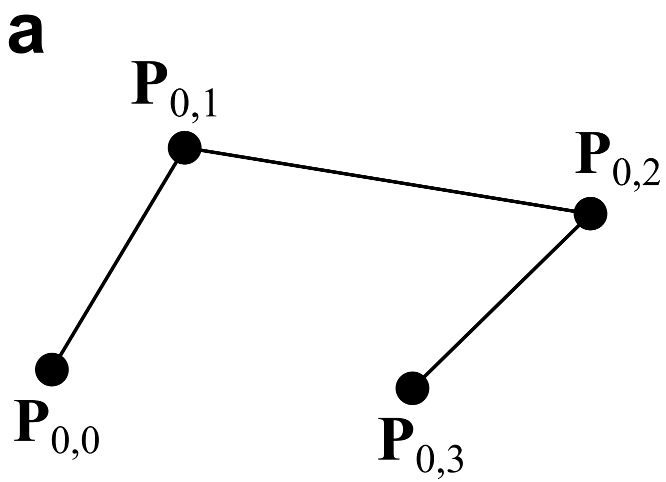
Casteljau's Construction $\mu = 0.6$



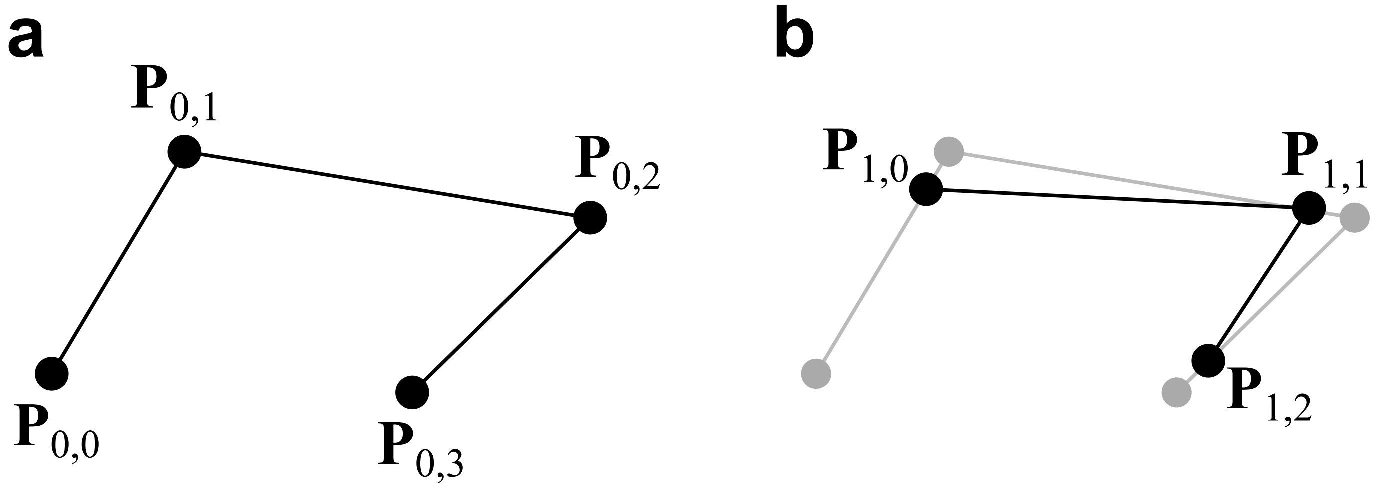
Casteljau's Construction $\mu = 0.6$



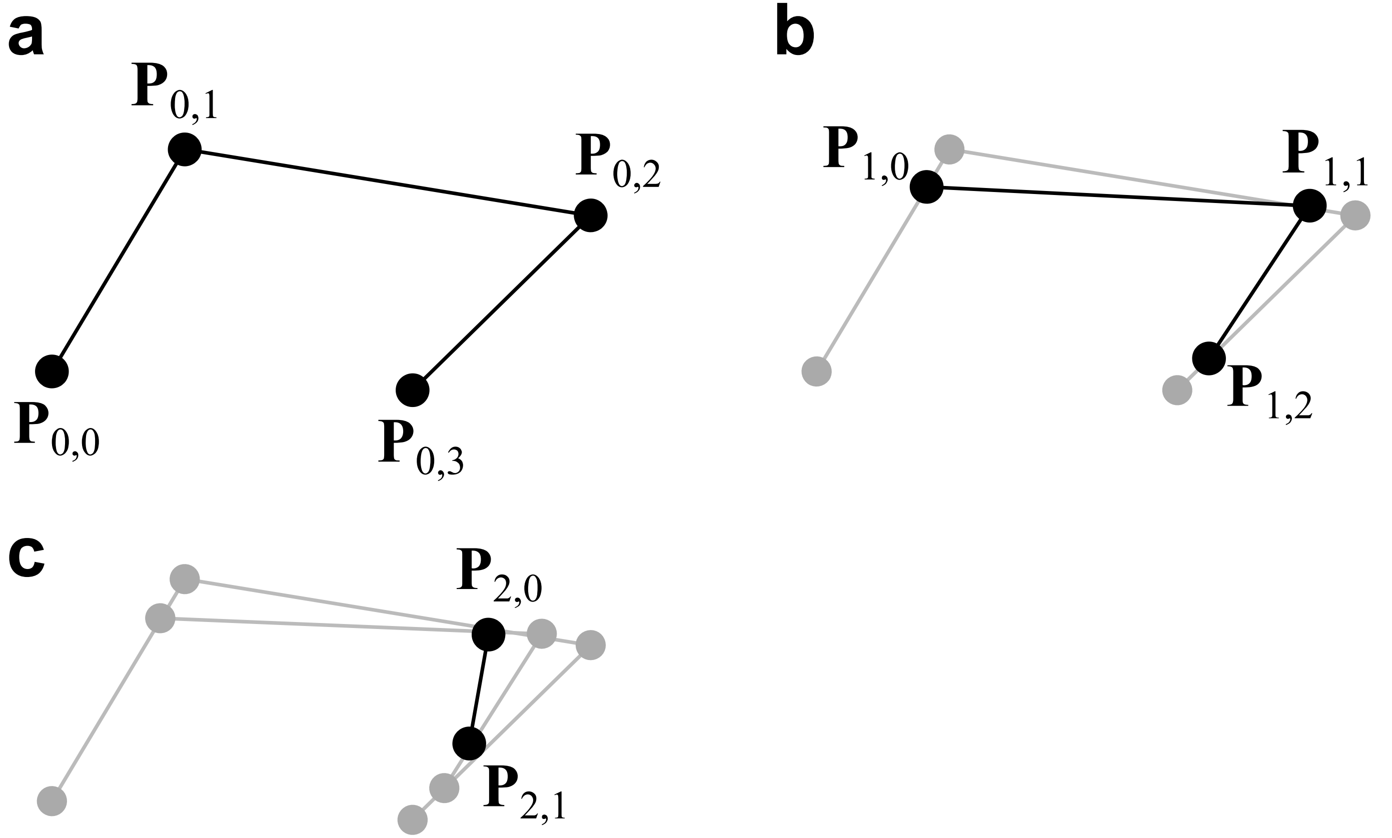
Casteljau's Construction μ = 0.9



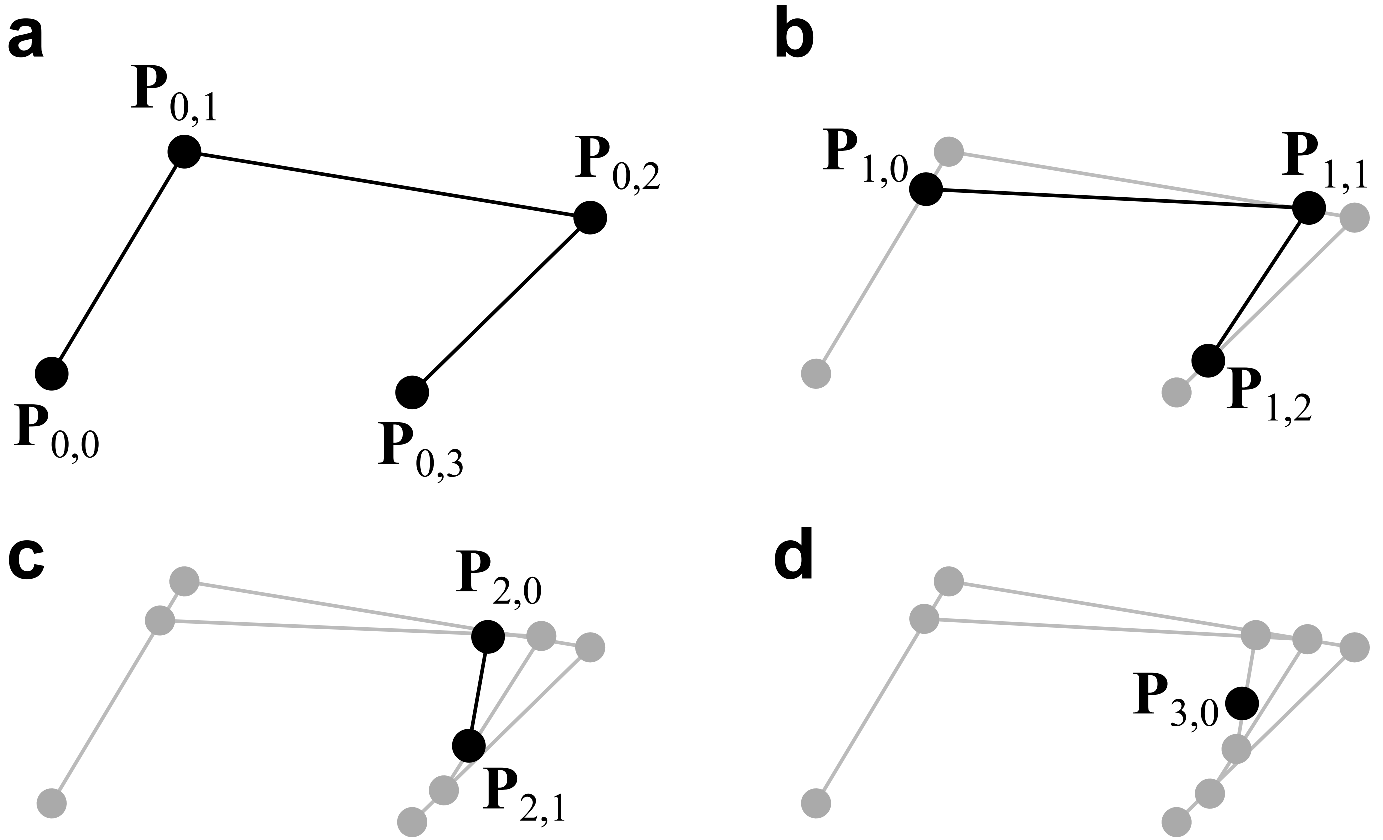
Casteljau's Construction $\mu = 0.9$



Casteljau's Construction $\mu = 0.9$



Casteljau's Construction $\mu = 0.9$



Bernstein Blending Function

- Splines (including Bezier curves) can be formulated as a blend of the knots.

- Consider the vector line equation

$$\mathbf{P} = (1-\mu)\mathbf{P}_0 + \mu\mathbf{P}_1$$

- It is a linear 'blend' of two points, and could also be considered the two-point Bezier curve!

Blending Equation

- Any point on the spline is simply a blend of all the other points. For N+1 knots we have:

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

- where W is the Bernstein blending function

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

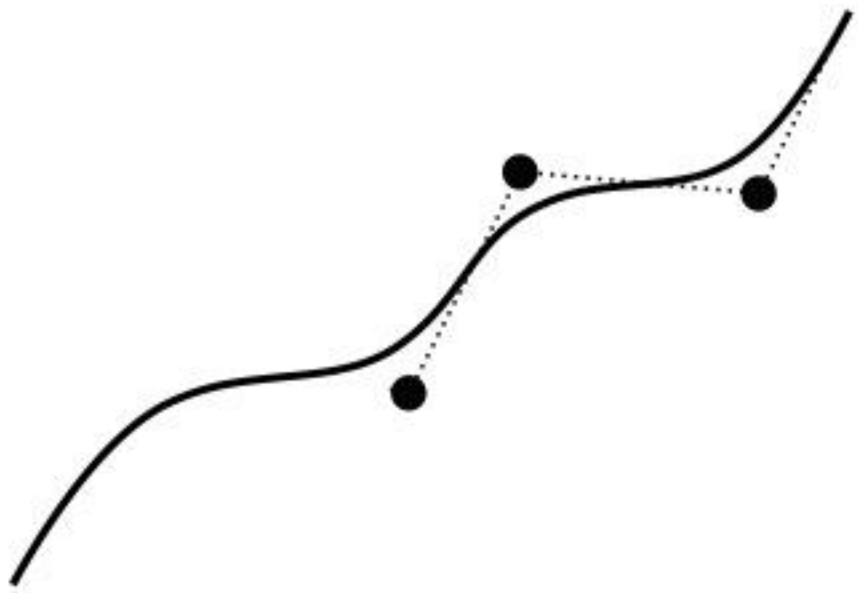
$$\binom{N}{i} = \frac{N!}{(N-i)!i!}$$

Blending Equation: Expansions for different N

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

Bezier Curves lack local control

- Since all the knots of the Bezier curve all appear in the blend they cannot be used for curves with fine detail.
- However they are very effective as spline patches.



Four point Bezier Curves and Cubic Patches

- We can show their equivalence:

Four point Bezier curve	$=$	Cubic patch going through the first and last knots ($\mathbf{P}_0$ and $\mathbf{P}_3$)

- It is possible to show their equivalence by
  - Expanding the iterative blending equation
  - Reversing the de Casteljau algorithm

Expanding the blending equation

- For the case of four knots we can expand the Bernstein blending function to get a polynomial in μ:

$$\begin{aligned}\mathbf{P}(\mu) &= \sum_{i=0}^{3} W(3,i,\mu)\mathbf{P}_i \\ &= (1-\mu)^3\mathbf{P}_0 + 3\mu(1-\mu)^2\mathbf{P}_1 + 3\mu^2(1-\mu)\mathbf{P}_2 + \mu^3\mathbf{P}_3\end{aligned}$$

- This can be multiplied out to give an equation of the form:

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

where

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

Expanding the blending equation

- These equations are linear

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

- Note that $\mathbf{P}_0$ and $\mathbf{P}_3$ are the endpoints
- Recall the matrix form used for a cubic spline patch

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

Expanding the blending equation

- These equations are linear

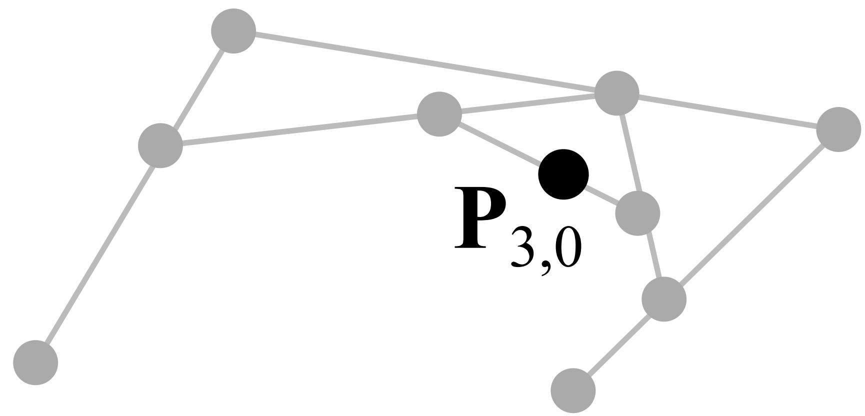
$$\begin{aligned} \mathbf{a}_0 &= \mathbf{P}_0 \\ \mathbf{a}_1 &= 3\mathbf{P}_1 - 3\mathbf{P}_0 \\ \mathbf{a}_2 &= 3\mathbf{P}_2 - 6\mathbf{P}_1 + 3\mathbf{P}_0 \\ \mathbf{a}_3 &= \mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0 \end{aligned}$$

- So we get the directions at the endpoints by using $\mathbf{P}_1$ and $\mathbf{P}_2$.
- We have shown the blending equation is the same as a cubic patch

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

Reversing the de Casteljau algorithm

We start from the point $\mathbf{P}_{3,0}$ and work in reverse to express it in terms of its construction line.



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

Reversing the de Casteljau algorithm

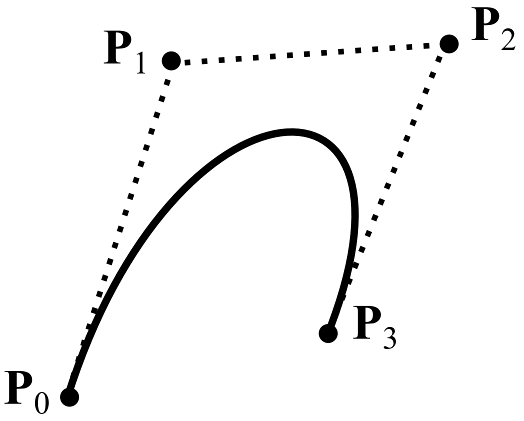
. . . continuing the expansion, we can drop the first subscript (which indicates the recursion level) to get:

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

This is the same as the expanded Bernstein blending polynomial which we have already shown is equivalent to a cubic spline patch

Control Points

- We can summarise the four point Bezier Curve by saying that it has
  - two points that are interpolated ($\mathbf{P}_0$, $\mathbf{P}_3$)
  - two control points ($\mathbf{P}_1$, $\mathbf{P}_2$)
- The curve starts at $\mathbf{P}_0$ and ends at $\mathbf{P}_3$ and its shape can be determined by moving control points $\mathbf{P}_1$, $\mathbf{P}_2$.
- This could be done interactively using a mouse.



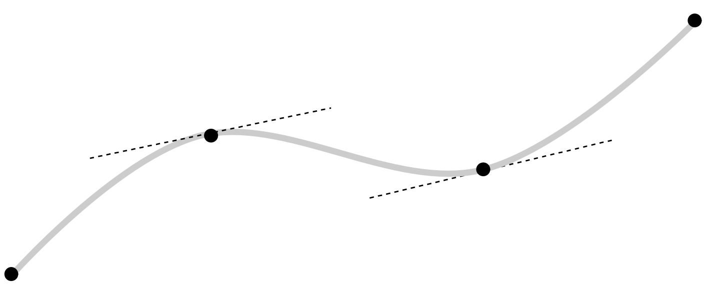
In summary …

- The simplest and most effective way to draw a smooth curve through a set of points is to use a cubic patch.

No interaction needed?

setting the gradients by the central difference $\frac{1}{2}(\mathbf{P}_{i+1} - \mathbf{P}_{i-1})$

is effective.



User wants interactive shape adjustment?

The four point Bezier formulation is ideal