COMP3421

Curves, modelling

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Curves

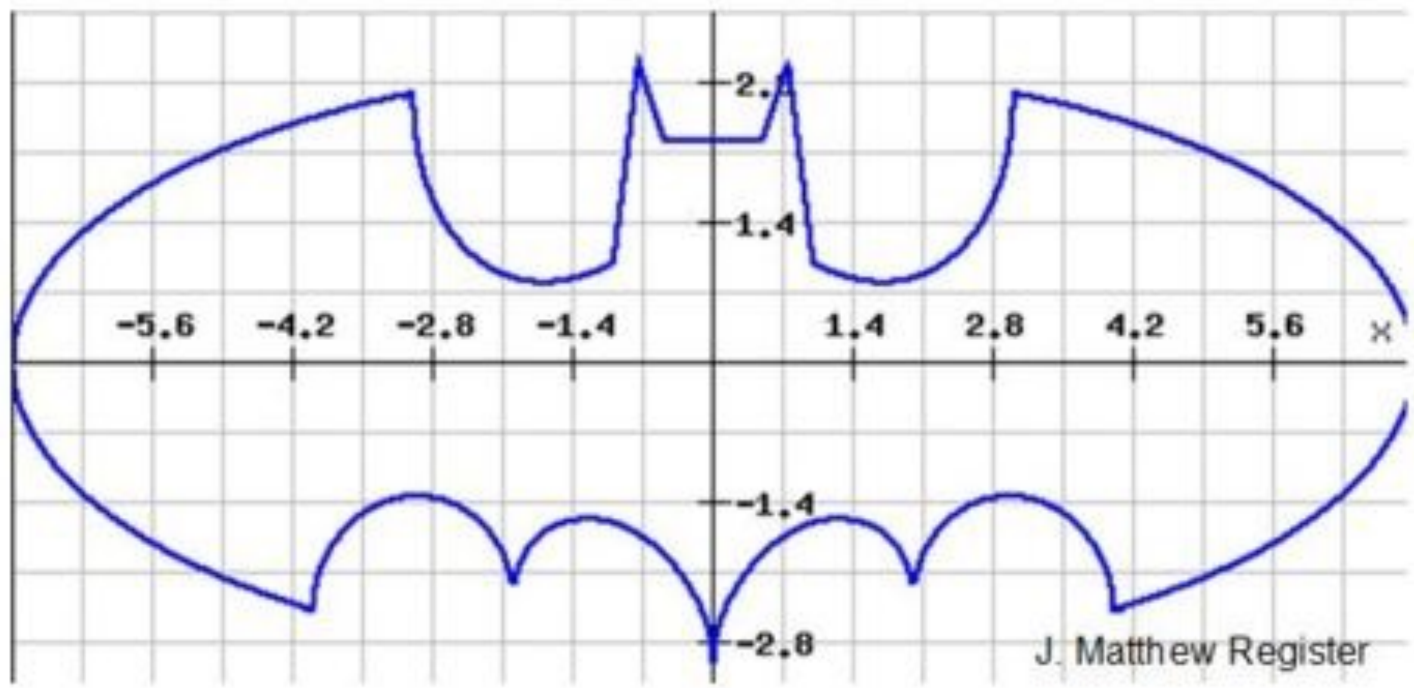
- We want a general purpose solution for drawing curved lines and surfaces. It should:
    - Be easy and intuitive to draw curves
    - General, supporting a wide variety of shapes.
    - Be computationally cheap.

Curves

- Easy

Batman Equation

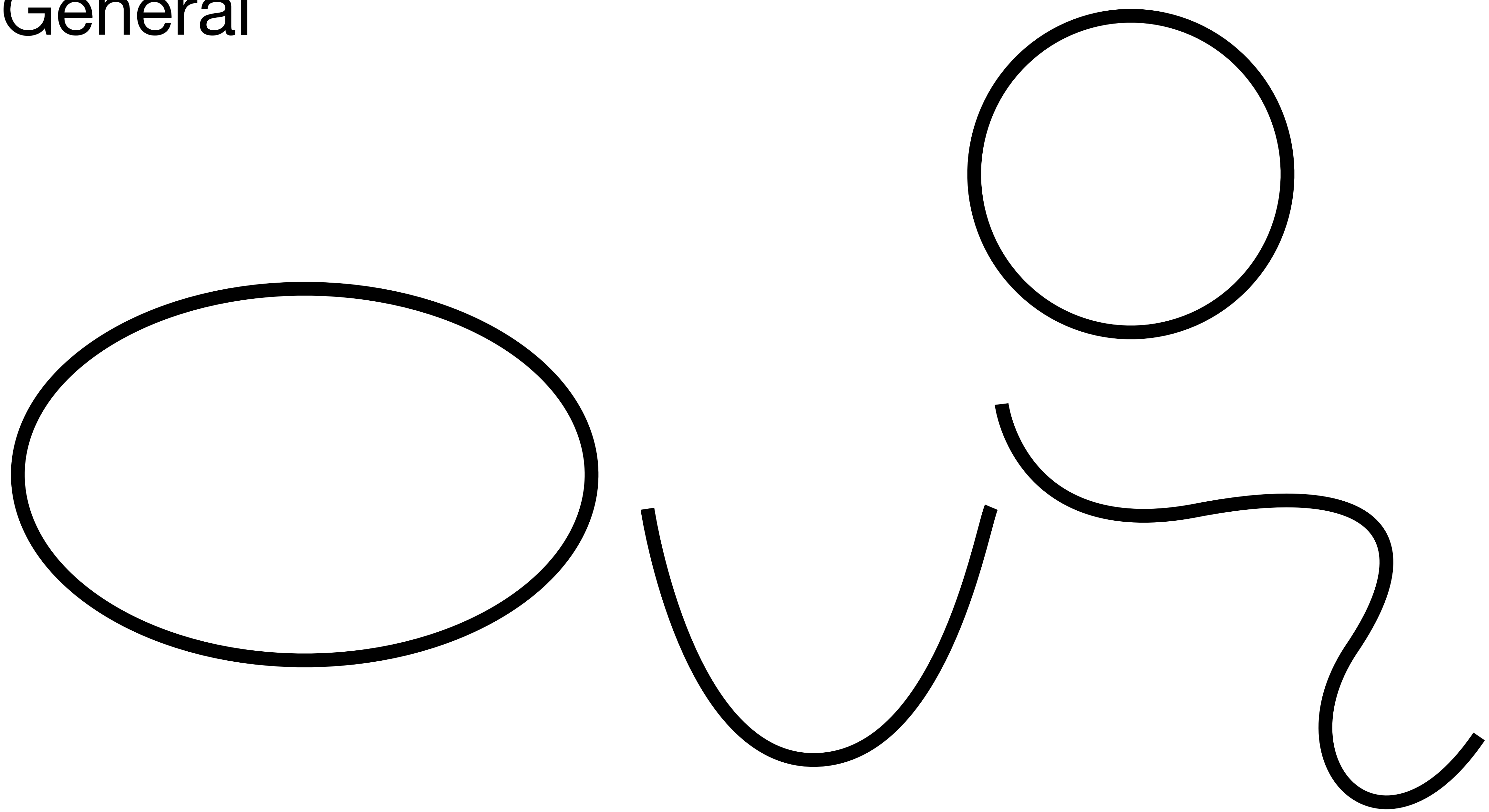
$$\left(\left(\frac{x}{7}\right)^2\sqrt{\frac{||x|-3|}{|x|-3}}+\left(\frac{y}{3}\right)^2\sqrt{\frac{\left|y+\frac{3\sqrt{33}}{7}\right|}{y+\frac{3\sqrt{33}}{7}}}-1\right)\cdot\left(\left|\frac{x}{2}\right|-\left(\frac{3\sqrt{33}-7}{112}\right)x^2-3+\sqrt{1-(||x|-2|-1)^2}-y\right)$$
$$\cdot\left(9\sqrt{\frac{|(|x|-1)(|x|-.75)|}{(1-|x|)(|x|-.75)}}-8|x|-y\right)\cdot\left(3|x|+.75\sqrt{\frac{|(|x|-.75)(|x|-.5)|}{(.75-|x|)(|x|-.5)}}-y\right)$$
$$\cdot\left(2.25\sqrt{\frac{|(x-.5)(x+.5)|}{(.5-x)(.5+x)}}-y\right)\cdot\left(\frac{6\sqrt{10}}{7}+(1.5-.5|x|)\sqrt{\frac{||x|-1|}{|x|-1}}-\frac{6\sqrt{10}}{14}\sqrt{4-(|x|-1)^2}-y\right)=0$$



(this is not easy)

Curves

- General



Curves

- Cheap
  - Drawn every frame (up to 60 times a second)
  - How many curves on a car?

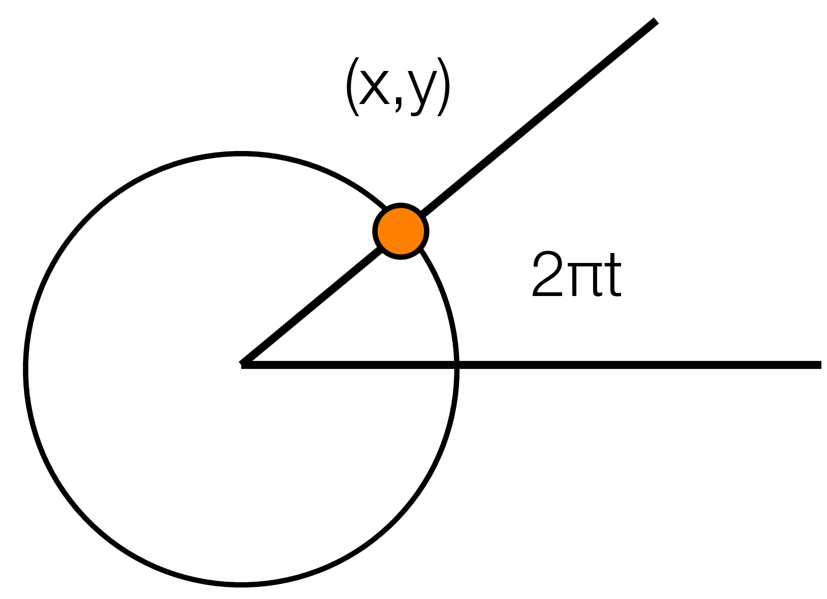
Parametric curves

- It is generally useful to express curves in parametric form:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = P(t), \text{ for } t \in [0,1]$$

- Eg:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos 2\pi t \\ \sin 2\pi t \end{pmatrix}$$

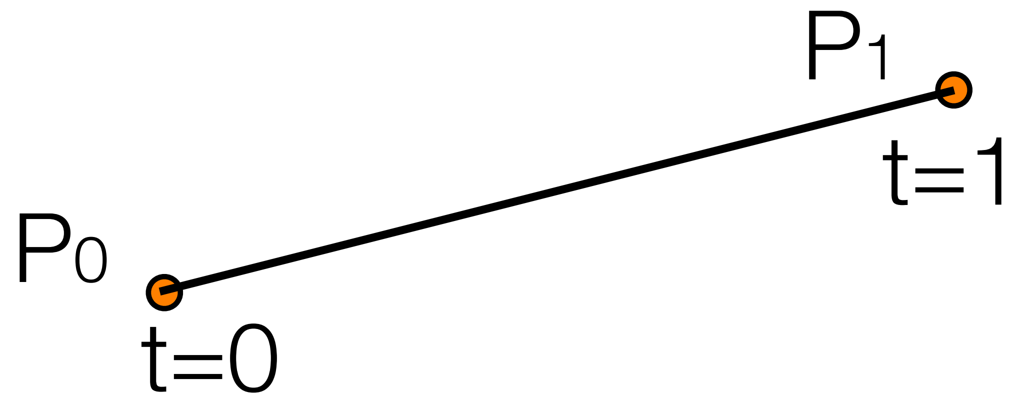


Interpolation

- Trigonometric operations like sin() and cos() are expensive to calculate.
- We would like a solution that involves fewer floating point operations.
- We also want a solution which allows for intuitive curve design.
- Interpolating control points is a good solution to both these problems.

Linear interpolation

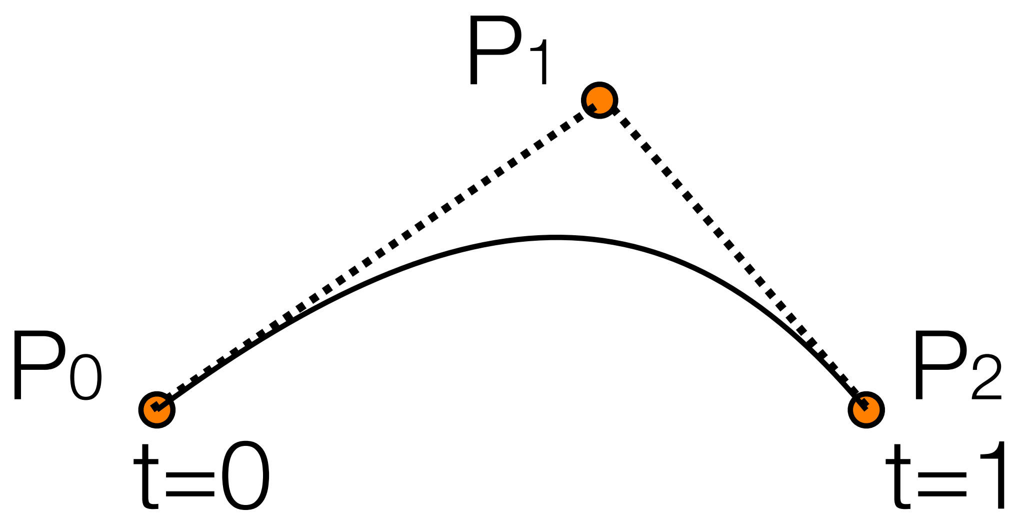
Good for straight lines.
Linear function: Degree 1
2 control points: Order 2



$$P(t) = (1 - t)P_0 + tP_1$$

Quadratic interpolation

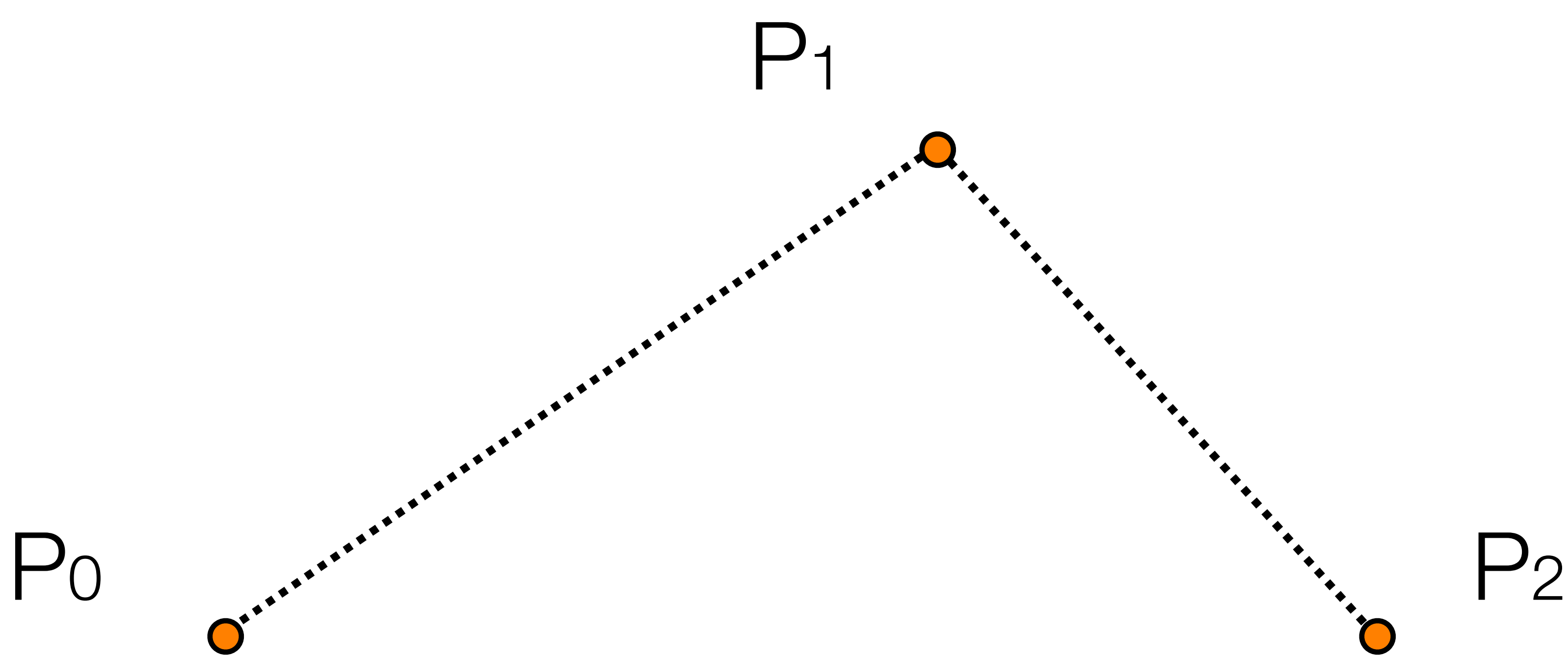
Interpolates (passes through) P0 and P2.
Approximates (passes near) P1.
Tangents at P0 and P2 point to P1.
Curves are all parabolas.



$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

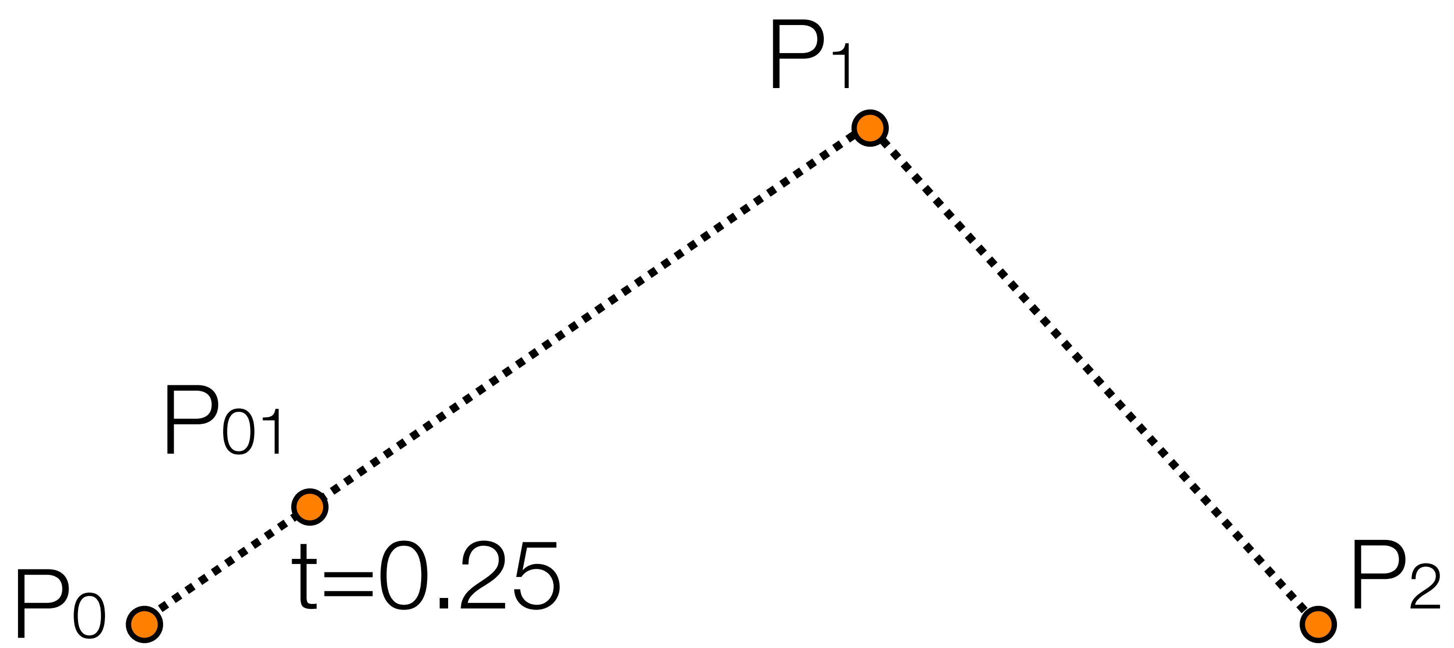
de Casteljau Algorithm

- The quadratic interpolation above can be computed as three linear interpolation steps:



de Casteljau Algorithm

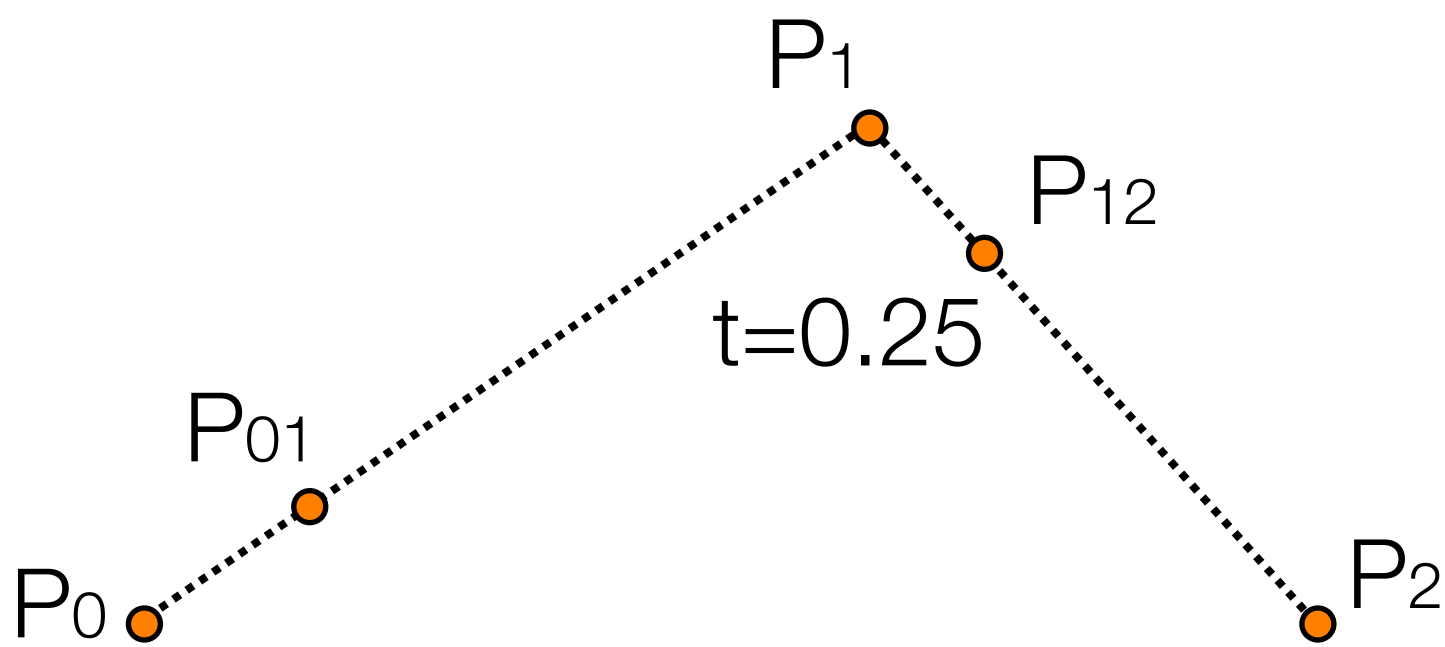
- The quadratic interpolation above can be computed as three linear interpolation steps:



$$P_{01}(t) = (1 - t)P_0 + tP_1$$

de Casteljau Algorithm

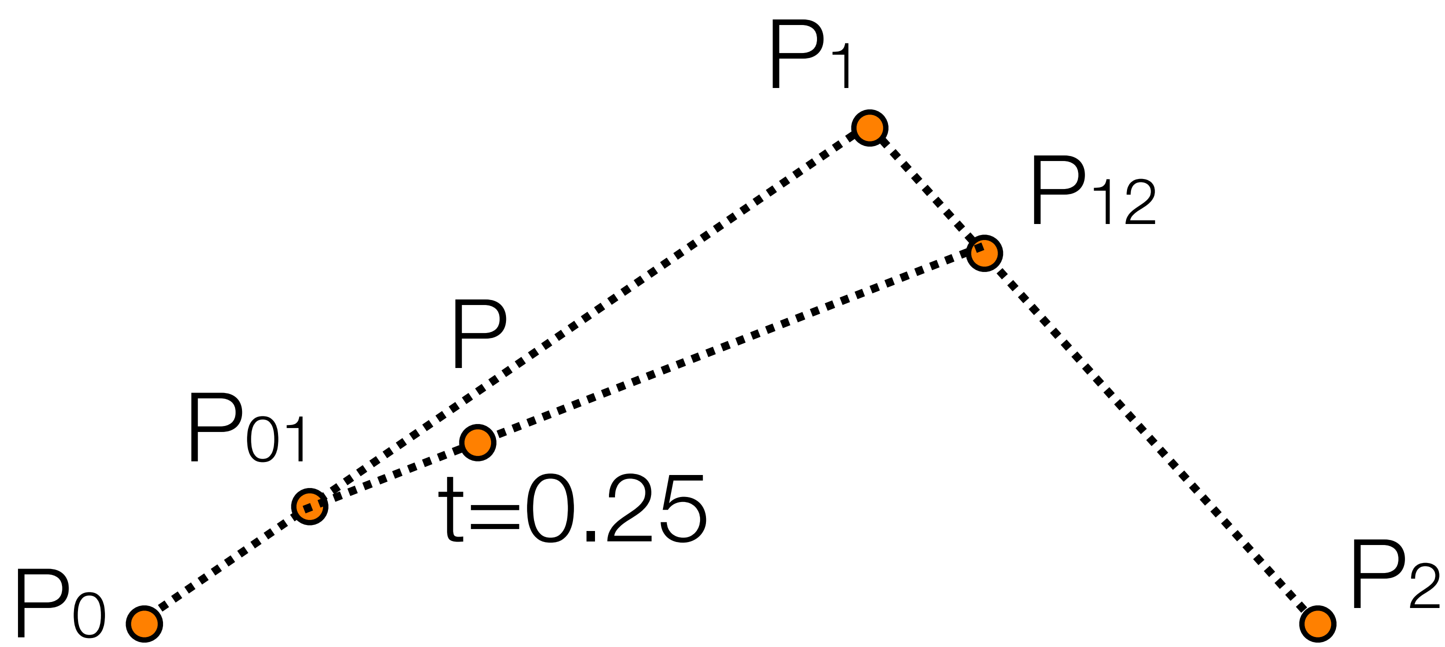
- The quadratic interpolation above can be computed as three linear interpolation steps:



$$P_{12}(t) = (1-t)P_1 + tP_2$$

de Casteljau Algorithm

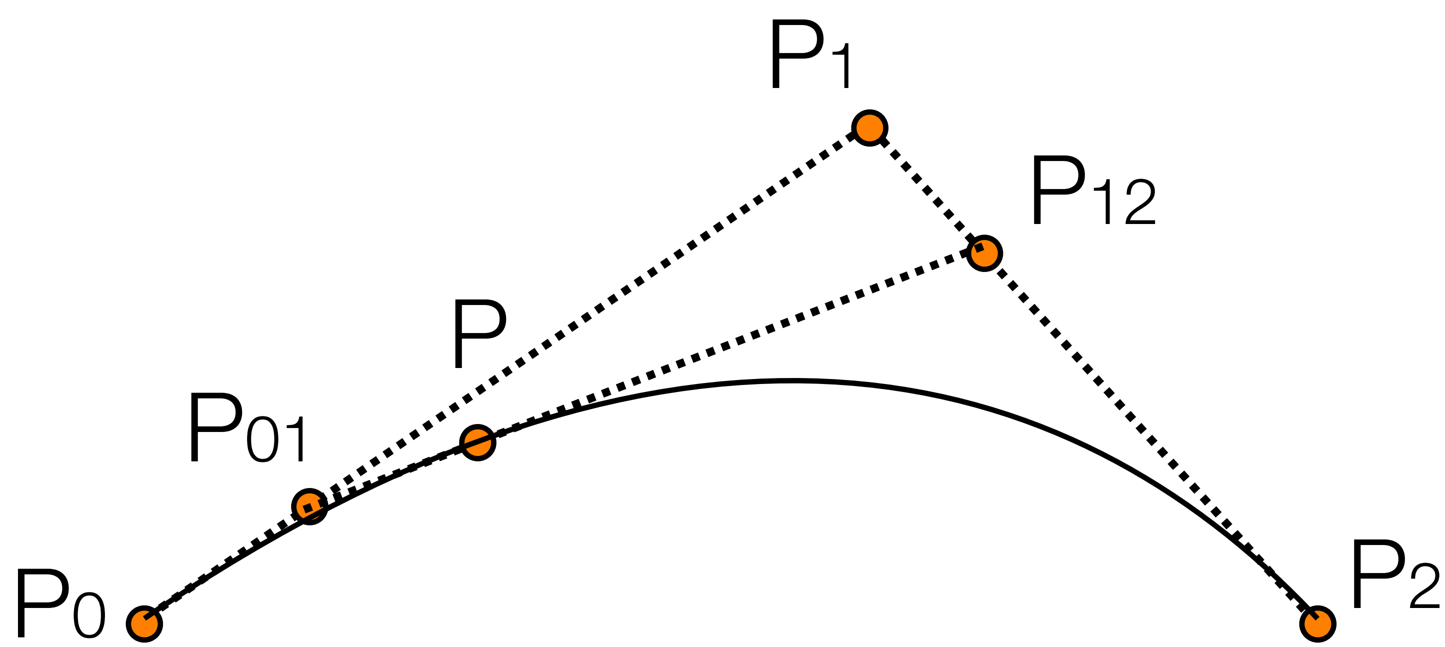
- The quadratic interpolation above can be computed as three linear interpolation steps:



$$P(t) = (1-t)P_{01} + tP_{12}$$

de Casteljau Algorithm

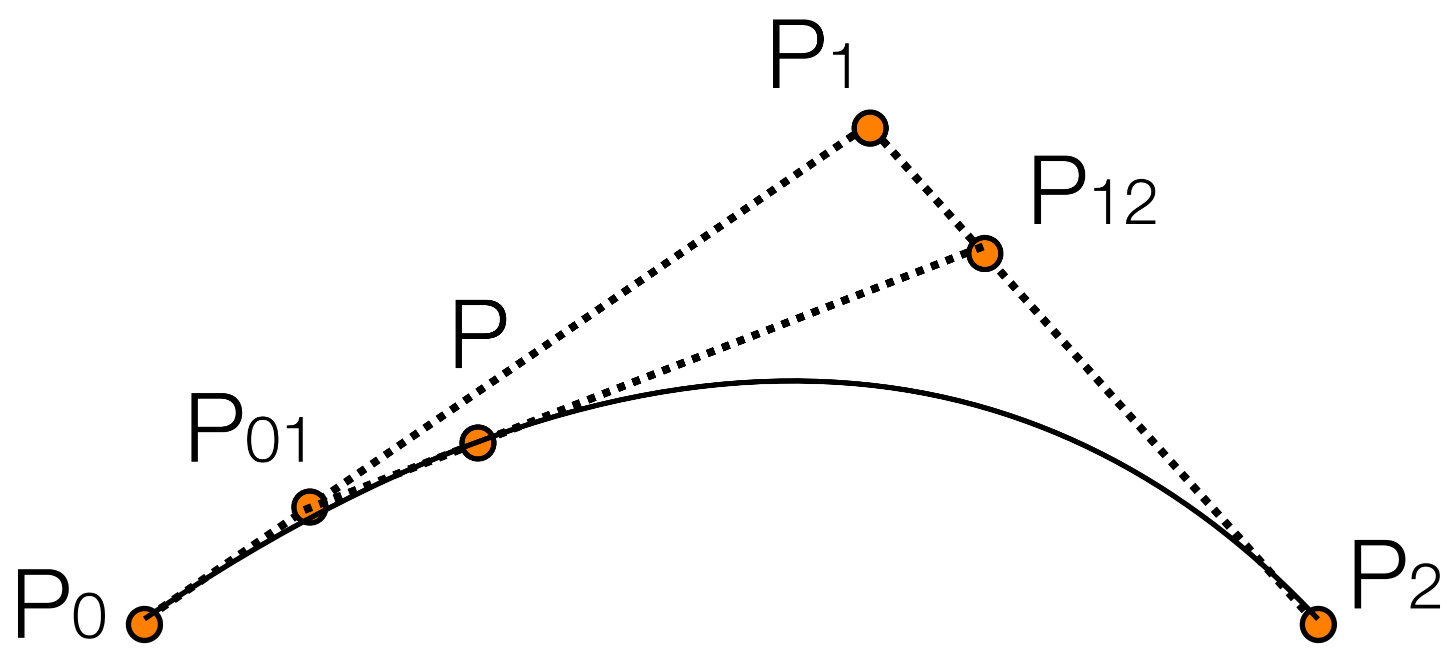
- The quadratic interpolation above can be computed as three linear interpolation steps:



$$P(t) = (1-t)P_{01} + tP_{12}$$

de Casteljau Algorithm

- The quadratic interpolation above can be computed as three linear interpolation steps:



$$P(t) = (1-t)P_{01} + tP_{12}$$

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

Exercise

- Using de Casteljau's algorithm calculate the point at t = 0.75 for the quadratic Bezier with the following control points:

  (0,0), (4,8), and (12,4)

- Confirm your answer using the equation

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

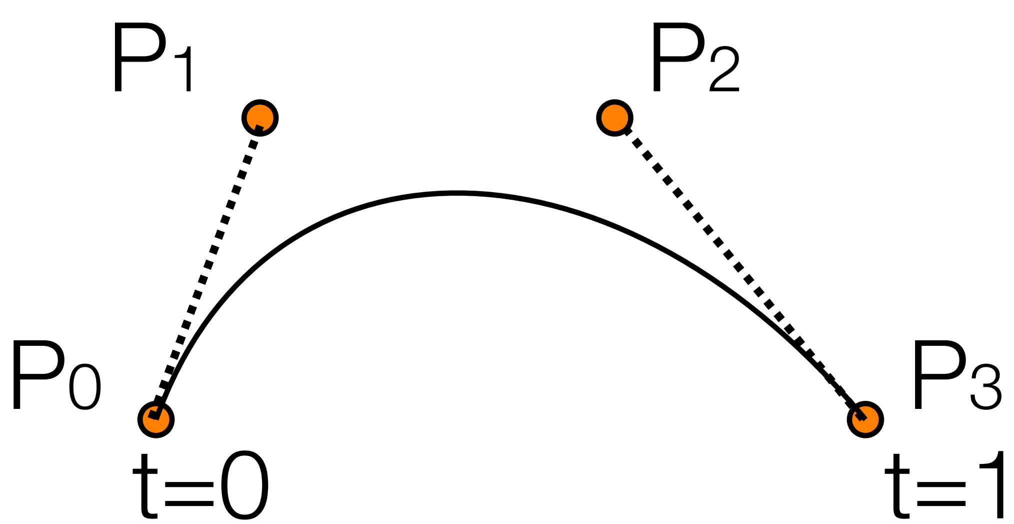
Exercise

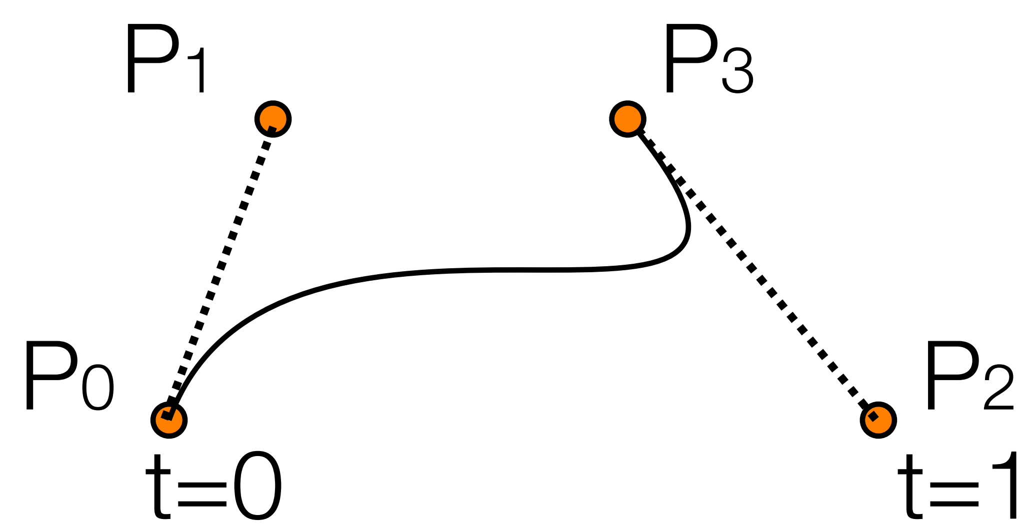
- Prove that de Casteljau's algorithm is equivalent to the quadratic interpolation formula

Cubic interpolation

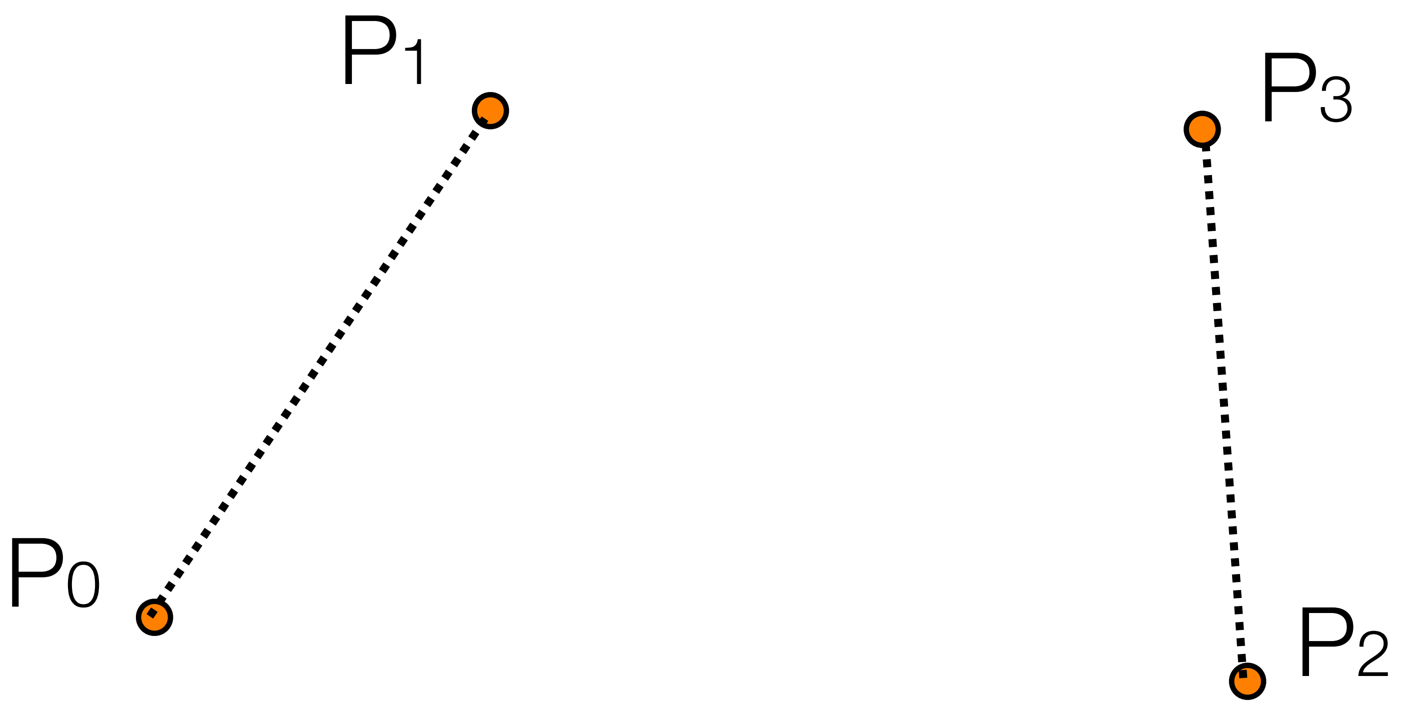
- Interpolates (passes through) P0 and P3.
  Approximates (passes near) P1 and P2.
  Tangents at P0 to P1 and P3 to P2.
  A variety of curves.

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

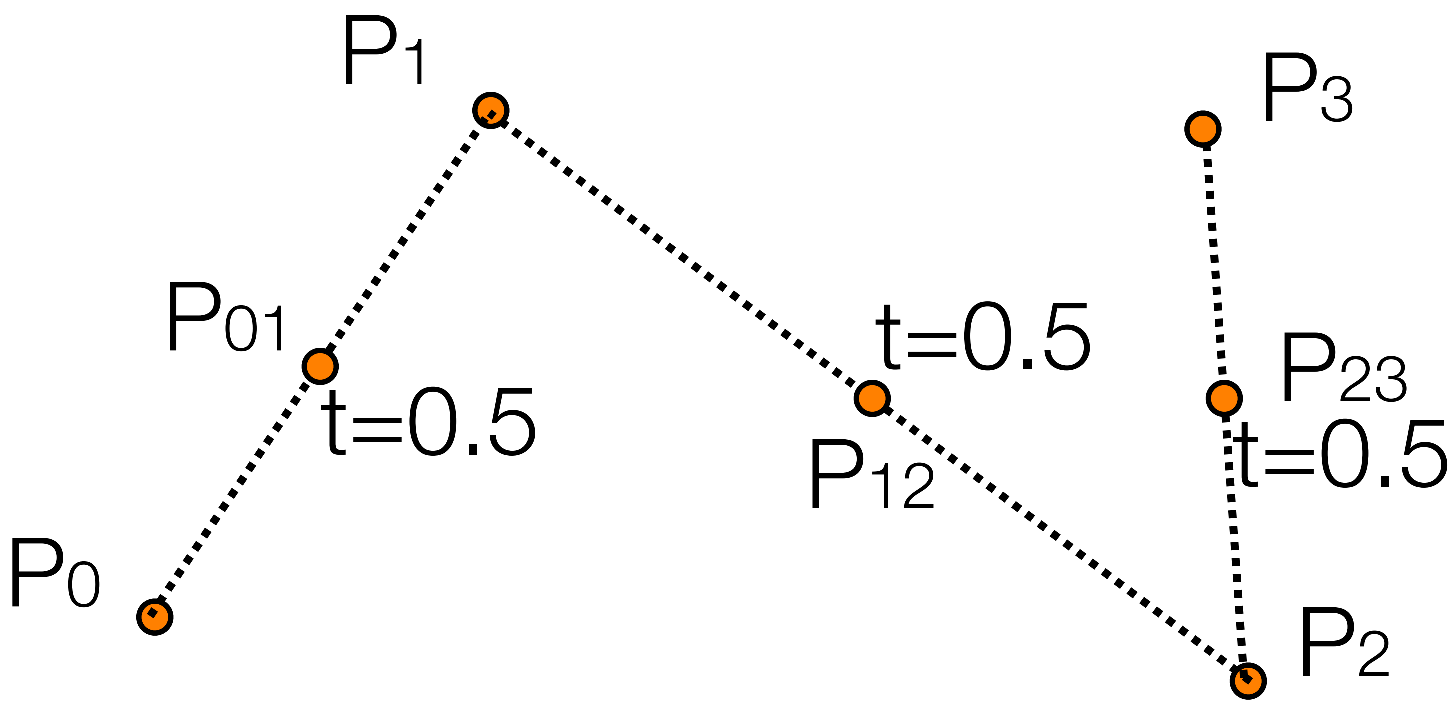




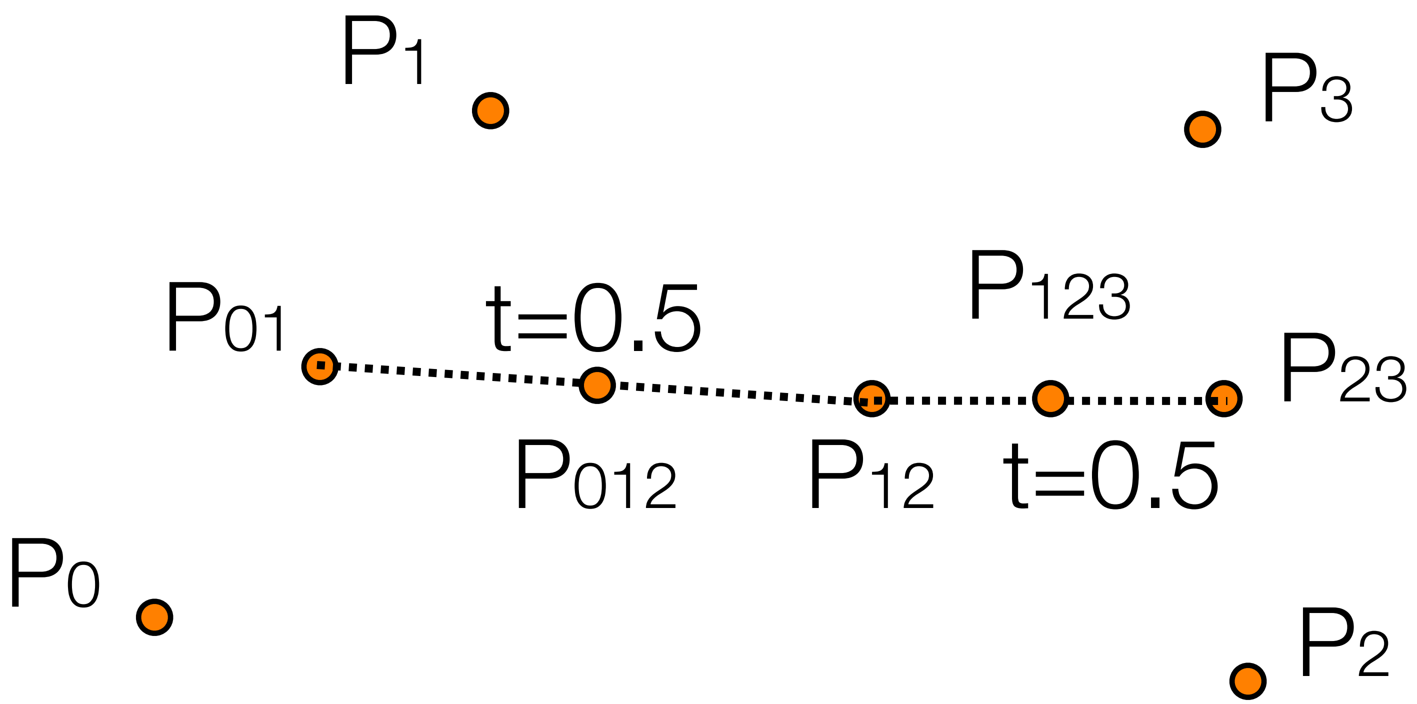
de Casteljau



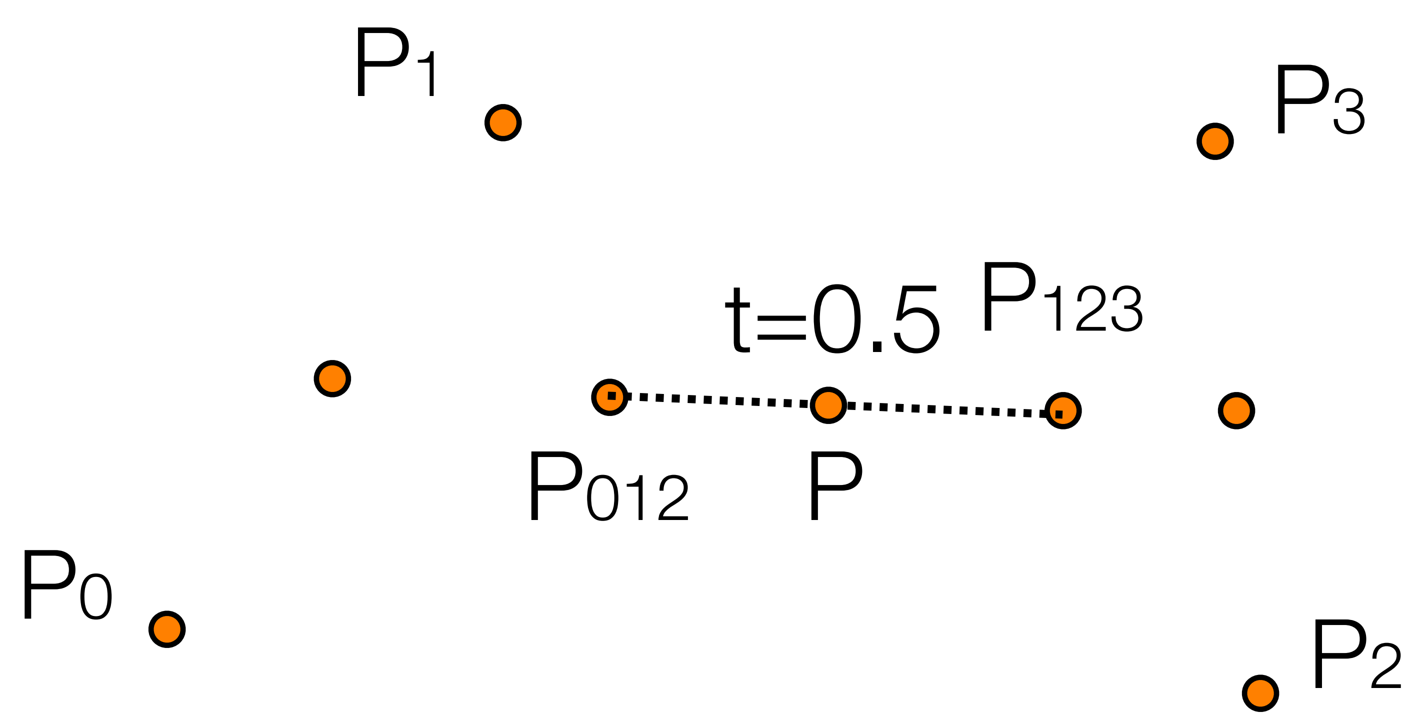
de Casteljau



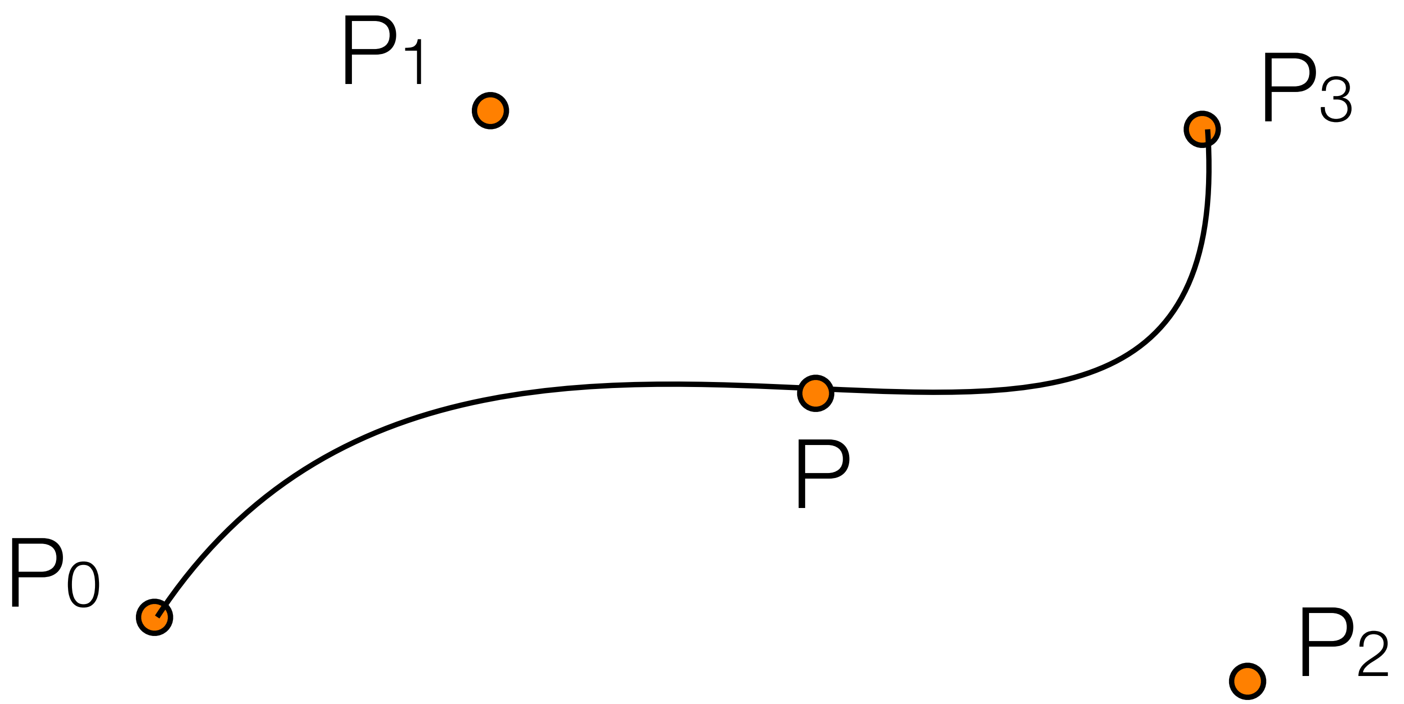
de Casteljau



de Casteljau



de Casteljau



Degree and Order

- **Linear Interpolation:** Degree one curve (m=1), Second Order (2 control points)
- **Quadratic Interpolation:** Degree two curve (m=2), Third Order (3 control points)
- **Cubic Interpolation:** Degree three curve (m=3), Fourth Order (4 control points)
- **Quartic Interpolation:** Degree four curve (m=4), Fifth Order (5 control points)
- Etc…

Bézier curves

This family of curves are known as Bézier curves.

They have the general form:

$$P(t) = \sum_{k=0}^{m} B_k^m(t) P_k$$

where m is the degree of the curve
and $P_0 \dots P_m$ are the control points.

Bernstein polynomials

- The coefficient functions $B_k^m(t)$ are called Bernstein polynomials. They have the general form:

$$B_k^m(t) = \binom{m}{k} t^k (1-t)^{m-k}$$

- where:

$$\binom{m}{k} = \frac{m!}{k!(m-k)!}$$

- is the binomial function.

Binomial Function

- Remember Pascal's triangle

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

Bernstein polynomials

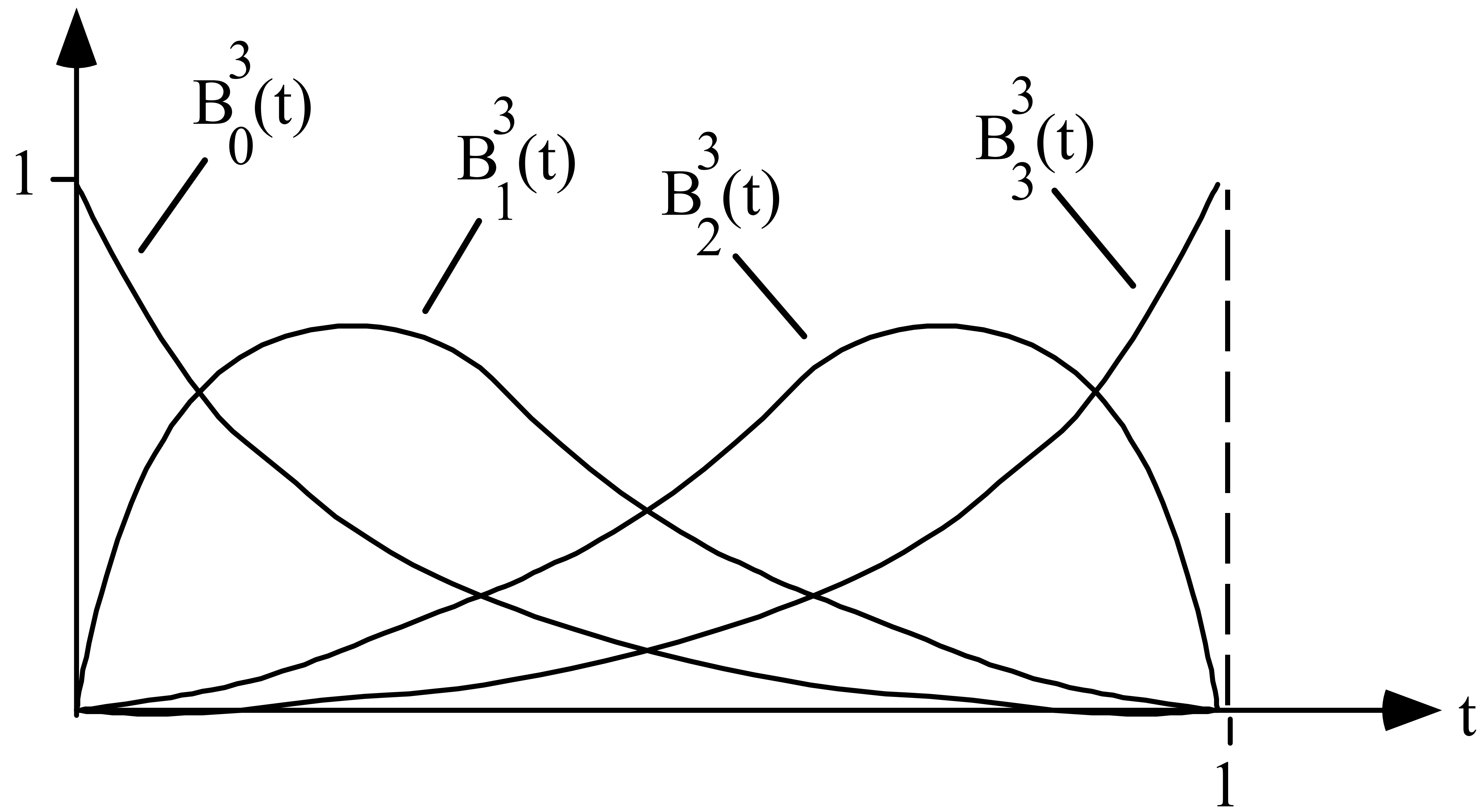
$$B_k^m(t) = \binom{m}{k} t^k (1-t)^{m-k}$$

- For the most common case, m = 3:

$$\begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3t(1-t)^2 \\ B_2^3(t) &= 3t^2(1-t) \\ B_3^3(t) &= t^3 \end{aligned}$$

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

Bernstein Polynomials for m = 3



Properties

- Bézier curves interpolate their endpoints and approximate all intermediate points.

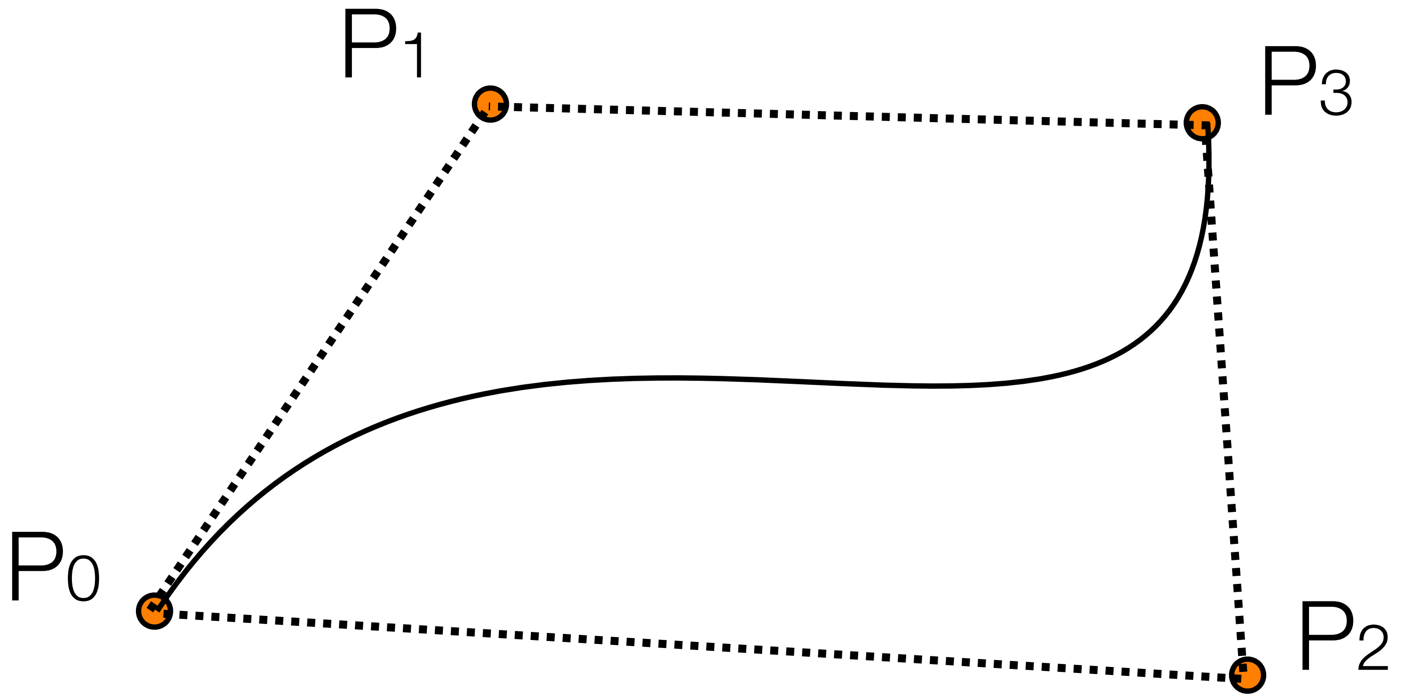
- Bézier curves are convex combinations of points:

$$\sum_{k=0}^{m} B_k^m(t) = 1$$

- Therefore they are invariant under affine transformation. The transformation of a Bézier curve is the curve based on the transformed control points.

Properties

- A Bézier curve lies within the convex hull of its control points:



Tangents

- The tangent vector to the curve at parameter t is given by:

$$\begin{aligned}\frac{dP(t)}{dt} &= \sum_{k=0}^{m} \frac{dB_k^m(t)}{dt} P_k \\ &= m \sum_{k=0}^{m-1} B_k^{m-1}(t)(P_{k+1} - P_k)\end{aligned}$$

- This is a Bézier curve of degree (m-1) on the vectors between control points.

Exercise

Compute the tangent at $t = 0.25$ for a quadratic Bezier curve with control points (0,0) (4,4) (8,2)

Problem: Polynomial Degree

- The degree of the Bernstein polynomials used is coupled to the number of control points: L+1 control points is a combination of L-degree polynomials.
- High degree polynomials are expensive to compute and are vulnerable to numerical rounding errors

Problem: Local control

- These curves suffer from non-local control.
- Moving one control point affects the entire curve.
- Each Bernstein polynomial is active (non-zero) over the entire interval (0,1). The curve is a blend of these functions so every control point has an effect on the curve for all t from (0,1)

Splines

- A spline is a smooth piecewise-polynomial function (for some measurement of smoothness).

- The places where the polynomials join are called knots.

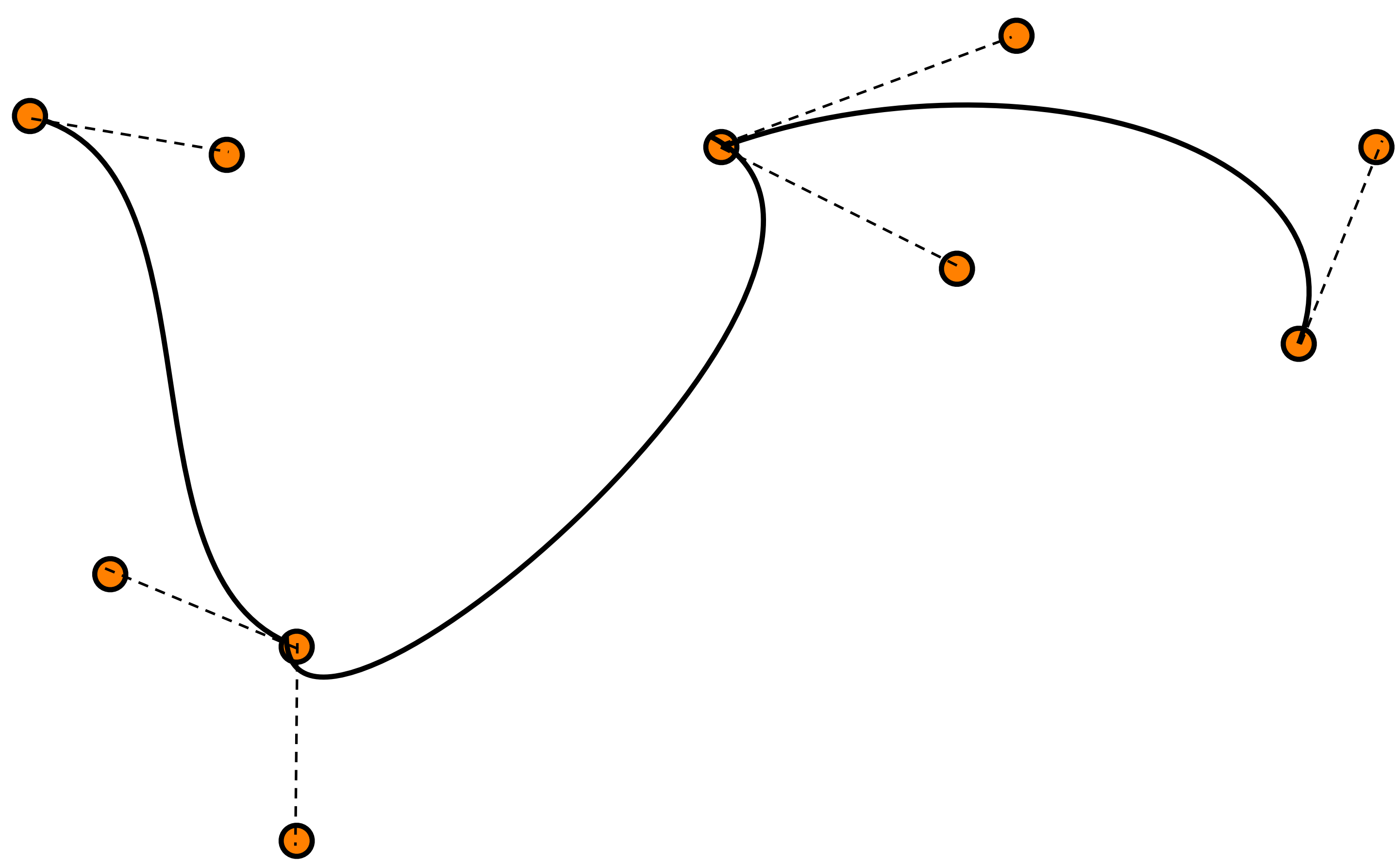
- A joined sequence of Bézier curves is an example of a spline.

Local control

- A spline provides local control.
- A control point only affects the curve within a limited neighbourhood.

Bézier splines

- We can draw longer curves as sequences of Bézier sections with common endpoints:



Generality

- Bezier splines can represent a large variety of different shapes.
- Not all the ones we want, though. We'll come back to this later in the course.

Links

http://www.malinc.se/m/DeCasteljauAndBezier.php
https://www.cse.unsw.edu.au/~cs3421/18s2/demos/nurbs.html

3D Modeling

- What if we want to generate meshes dynamically and not just load them from files?
- How can we make our own 3d meshes that are not just cubes?
- We will look at simple examples along with some clever techniques such as
  - Extrusion
  - Revolution

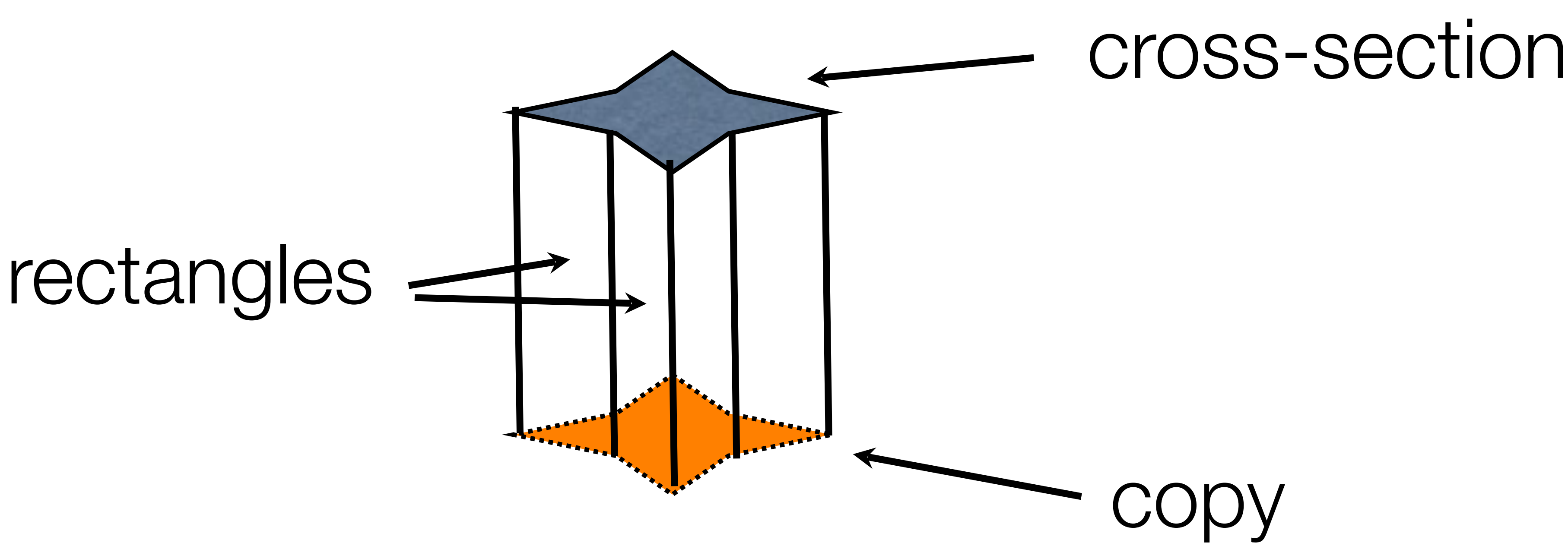
Exercise: Cone

- How can we model a cone?
- There are many ways.
- Simple way: Make a circle using a triangle fan parallel to the x-y plane. For example at z = -3
- Change to middle point to lie at a different z-point for example z = -1.

Extruding shapes

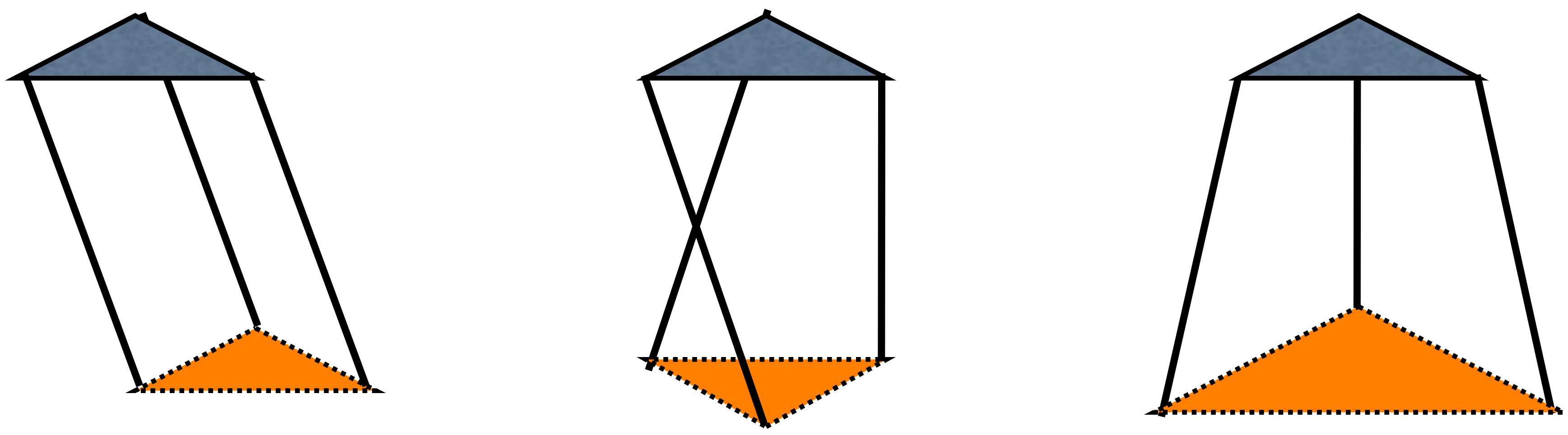
- Extruded shapes are created by sweeping a 2D polygon along a line or curve.

- The simplest example is a prism.



Variations

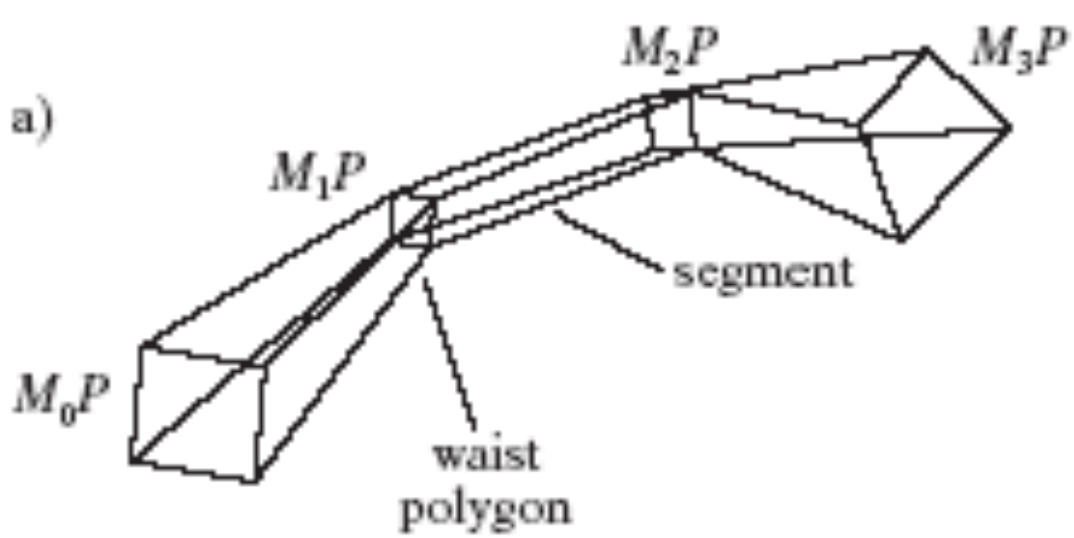
- One end of the prism can be translated, rotated or scaled from the other.



Segmented Extrusions

- A polygon P extruded multiple times, in different directions with different tapers and twists. The first segment has end polygons M_0P and M_1P, where the initial matrix M_0 positions and orients the starting end of the extrusion. The second segment has end polygons M_1P and M_2P, etc.

a)



b)

Segmented extrusions

- We can extrude a polygon along a path by specifying it as a series of transformations.

$$\begin{aligned} poly &= P_0, P_1, \ldots, P_k \\ path &= \mathbf{M_0}, \mathbf{M_1}, \ldots, \mathbf{M_n} \end{aligned}$$

- At each point in the path we calculate a cross-section:

$$poly_i = \mathbf{M_i} P_0, \mathbf{M_i} P_1, \ldots, \mathbf{M_i} P_k$$

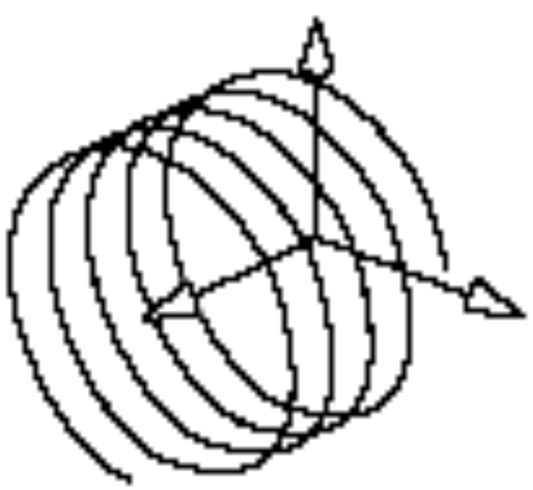
Segmented Extrusion

- Sample points along the spine using different values of t
- For each t:
  - generate the current point on the spine
  - generate a transformation matrix
  - multiply each point on the cross section by the matrix.
  - join these points to the next set of points using quads/triangles.

Segmented Extrusion Example

- For example we may wish to extrude a circle cross-section around a helix spine.

- helix $C(t) = (cos(t), sin(t), bt))$.



Transformation Matrix

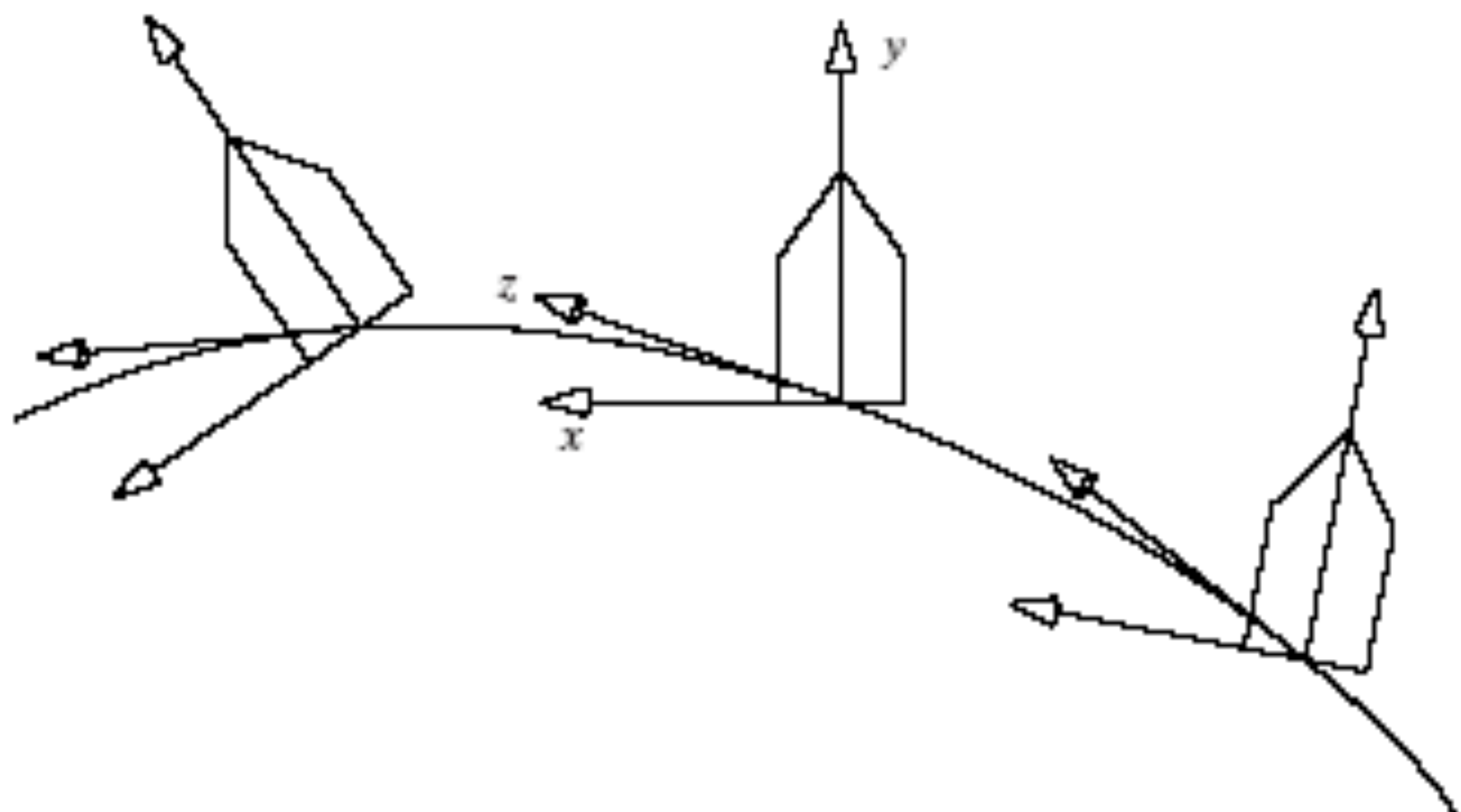
- How can we automatically generate a matrix to transform our cross-section by?

- We need the origin of the matrix to be the new point on the spine. This will translate our cross-section to the correct location.

- Which way will our cross-section be oriented? What should i, j and k of our coordinate frame be?

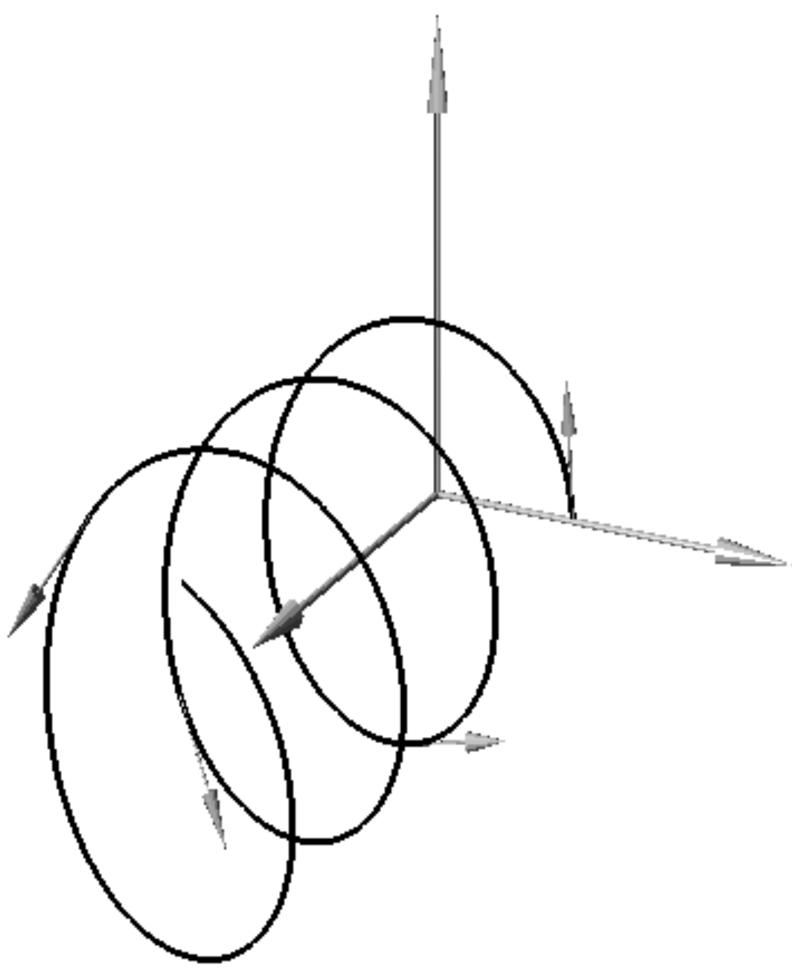
Frenet Frame

- We can get the curve values at various points t_i and then build a polygon perpendicular to the curve at $C(t_i)$ using a Frenet frame.

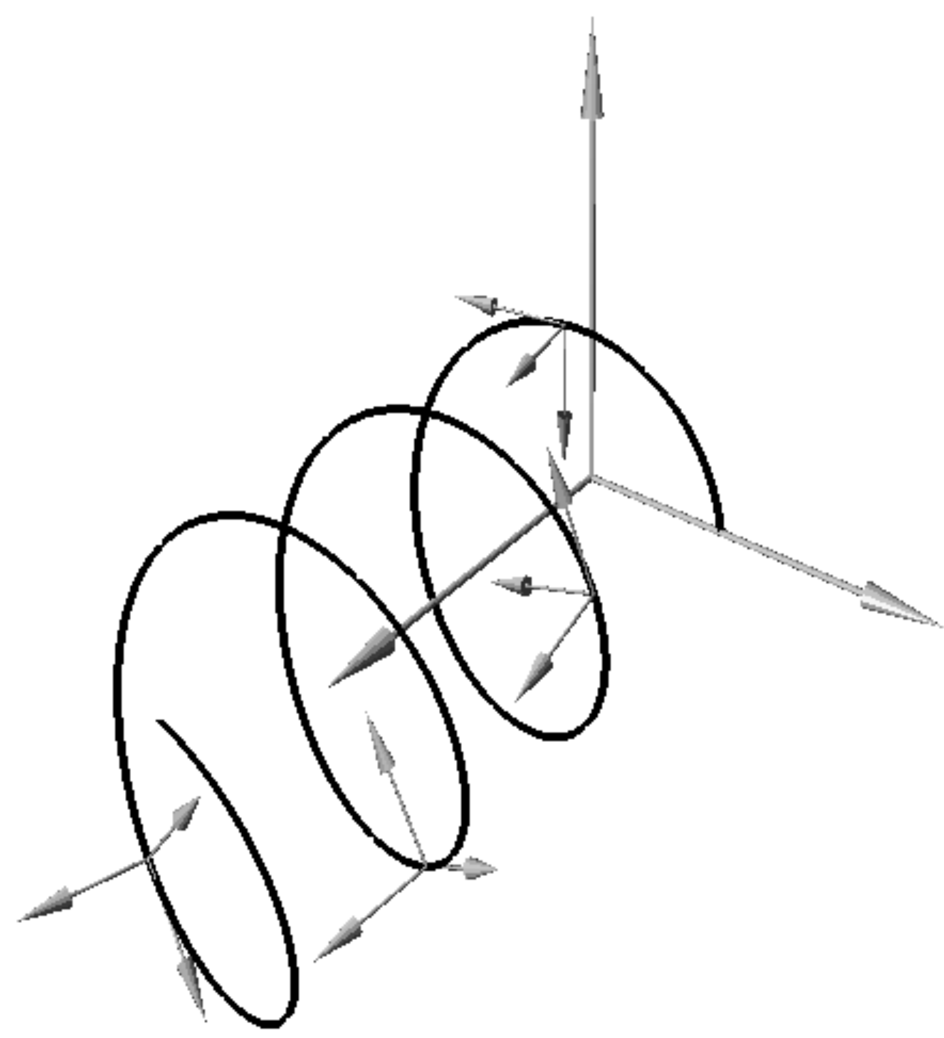


Example

a). Tangents to the helix.



b). Frenet frame at various values of t, for the helix.



Frenet Frame

- Once we calculate the tangent to the spine at the current point, we can use this to calculate normals.

- We then use the tangent and the 2 normals as i, j and k vectors of a co-ordinate frame.

- We can then build a matrix from these vectors, using the current point as the origin of the matrix.

Frenet frame

- We align the **k** axis with the (normalised) tangent, and choose values of **i** and **j** to be perpendicular.

$$\phi = C(t)$$

$$\mathbf{k} = \hat{C}'(t)$$

$$\mathbf{i} = \begin{pmatrix} -k_2 \\ k_1 \\ 0 \end{pmatrix}$$

$$\mathbf{j} = \mathbf{k} \times \mathbf{i}$$

Frenet Frame Calculation

Finding the tangent (our k vector):

1. Using maths. Eg for

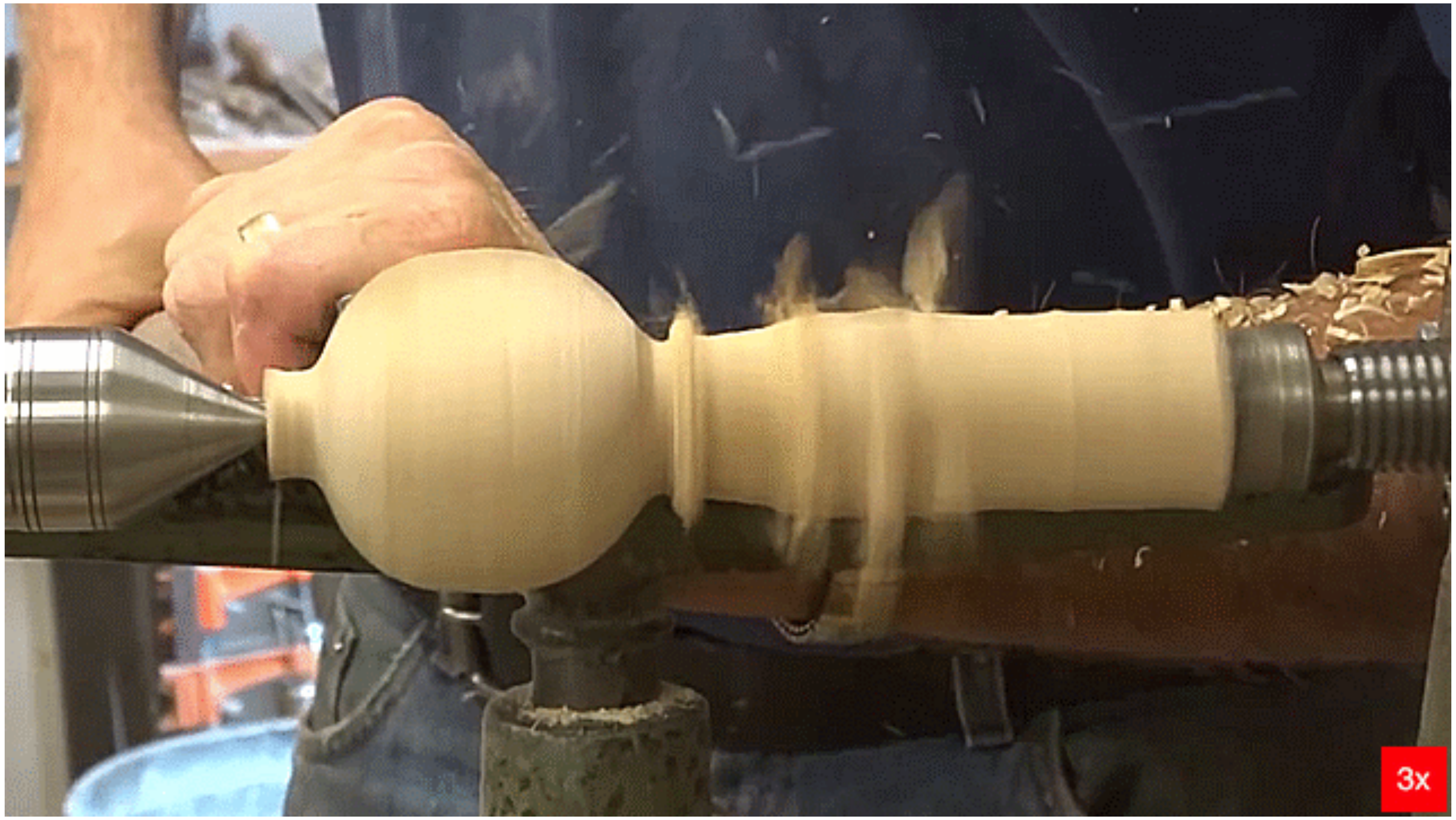
$$C(t) = (\cos(t), \sin(t), bt)$$

$$T(t) = \text{normalise}(-\sin(t), \cos(t), b)$$

2. Or just approximate the tangent

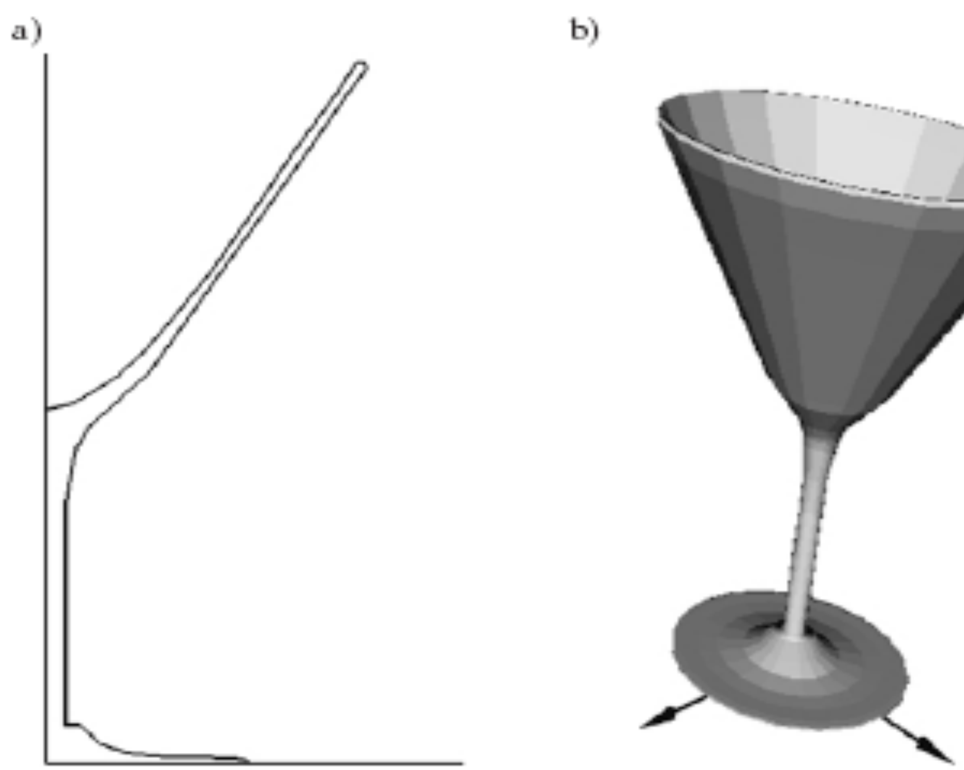
$$\mathbf{T(t) = normalise(C(t+1) - C(t-1))}$$

Revolution



Revolution

- A surface with radial symmetry (i.e. a round object, like a ring, a vase, a glass) can be made by sweeping a half cross-section around an axis.



Revolution

- Given a 2D curve

$$C(t) = (X(t), Y(t))$$

- We can revolve it by adding an extra parameter

$$P(t,\theta) = (X(t)\cos(\theta),\ Y(t),\ X(t)\sin(\theta))$$

L-Systems

- A Lindenmayer System (or L-System) is a method for producing fractal structures.
- They were initially developed as a tool for modelling plant growth.
- http://madflame991.blogspot.com.au/p/lindenmayer-power.html

L-Systems

- Can give us realistic plants and trees

Some determinstic 3D branching plants.

Rewrite rules

An L-system is a formal grammar:
a set of symbols and rewrite rules. Eg:

Symbols:

A, B, +, -

Rules:

A → B - A - B

B → A + B + A

Iteration

We start with a given string of symbols and then iterate, replacing each on the left of a rewrite rule with the string on the right.

A

B - A - B

A + B + A - B - A - B - A + B + A

B - A - B + A + B + A + B - A - B - ...

Drawing

Each string has a graphical interpretation, usually using turtle graphics commands:

A = draw forward 1 step

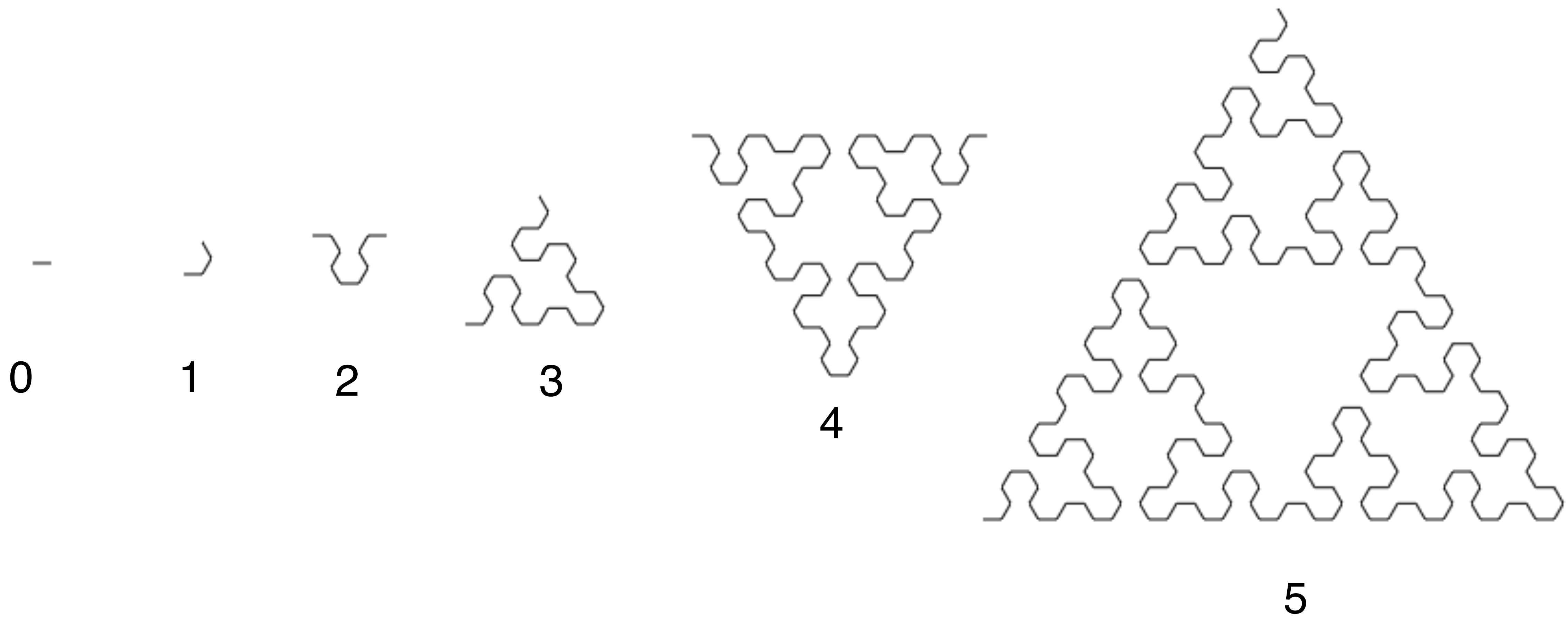
B = draw forward 1 step

+ = turn left 60 degrees

- = turn right 60 degrees

Sierpinski Triangle

- This L-System generates the fractal known as the Sierpinski Triangle:



Parameters

We can add parameters to our rewrite rules handle variables like scaling:

$A(s) \rightarrow B(s/2) - A(s/2) - B(s/2)$

$B(s) \rightarrow A(s/2) + B(s/2) + A(s/2)$

$A(s)$: draw forward s units

$B(s)$: draw forward s units

Push and Pop

We can also use a LIFO stack to save and restore global state like position and heading:

A → B [+ A] - A
B → B B

A : forward 10 B : forward 10

+: rotate 45 left - : rotate 45 right
[: push] : pop ;

Stochastic

We can add multiple productions with weights to allow random selection:

(0.5) A → B [A] A

(0.5) A → A

B → B B

Example

(0.5) X → F - [[X] + X] + F [+ F X] - X

(0.5) X → F - F [+ F X] + [[X] + X] - X

F → F F



3D L-Systems

We can build 3D L-Systems by allowing symbols to translate to models and transformations of the coordinate frame.

C : draw cylinder mesh
F : translate(0,0,10)
X : rotate(10, 1, 0, 0)
Y : rotate(10, 0, 1, 0)
S : scale(0.5, 0.5, 0.5)

Example

S -> A [+ B] + A

A -> A - A + A - A

B -> BA

After 1 iteration?

After 2 iterations?

After 3 iterations?

: A forward 10

: + rotate 45 (CW)

: - rotate -90

: [push

:] pop

Example in Format For Web Demo

```
-> S
1 A [ + B ]  + A
-> A
1 A - A +  A - A
-> B
1 BA
```

```
: A
forward 10
: +
rotate 45
: -
rotate -90
: [
push
: ]
pop
```

Algorithmic Botany

- You can read a LOT more here:
- http://algorithmicbotany.org/papers/