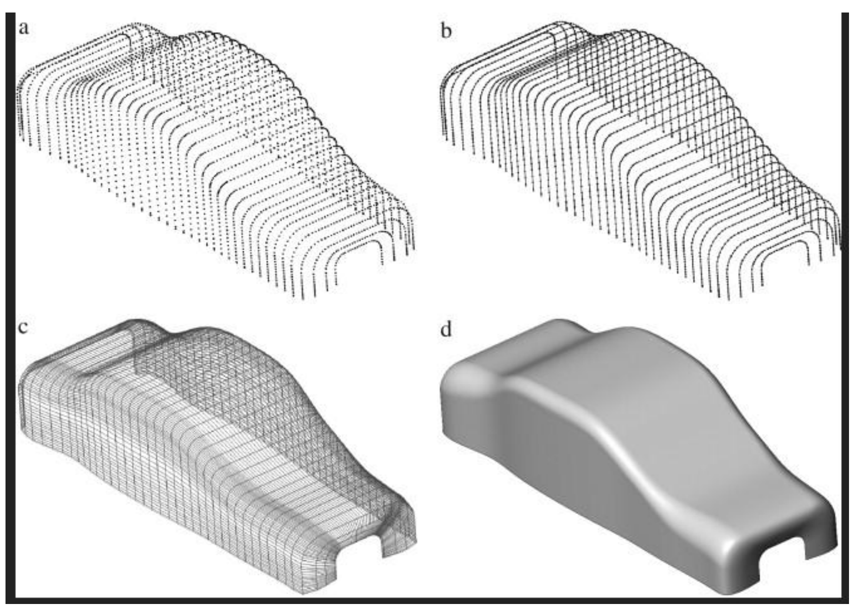
Curves and Surfaces

To do

- Continue to work on ray programming assignment
- Start thinking about final project

Curved Surfaces

- Motivation
  - Exact boundary representation for some objects
  - More concise representation that polygonal mesh
  - Easier to model with and specify for many man-made objects and machine parts (started with car bodies)



Curve and surface Representations

- Curve representation
  - Function: $y = f(x)$
  - Implicit: $f(x, y) = 0$
  - Subdivision: (x, y) as limit of recursive process
  - Parametric: $x = f(t)$, $y = g(t)$
- Curved surface representation
  - Function: $z = f(x, y)$
  - Implicit: $f(x, y, z)=0$
  - Subdivision: (x, y, z) as limit of recursive process
  - Parametric:

    $x = f(s, t)$, $y=g(s, t)$, $z = h(s, t)$

Parametric Surfaces

- Boundary defined by parametic function
  - x = f(u, v)
  - y = f(u, v)
  - Z = f(u, v)
- Example (sphere):
  - $X = \sin(\theta)\cos(\phi)$
  - $Y = \sin(\theta)\sin(\phi)$
  - $Z = \cos(\theta)$

Parametric Representation

- One function vs. many functions (defined piecewise)
- Continuity
- A parametric polynomial curve of order n:

$$x(u) = \sum_{i=0}^{n} a_i u^i$$

$$y(u) = \sum_{i=0}^{n} b_i u^i$$

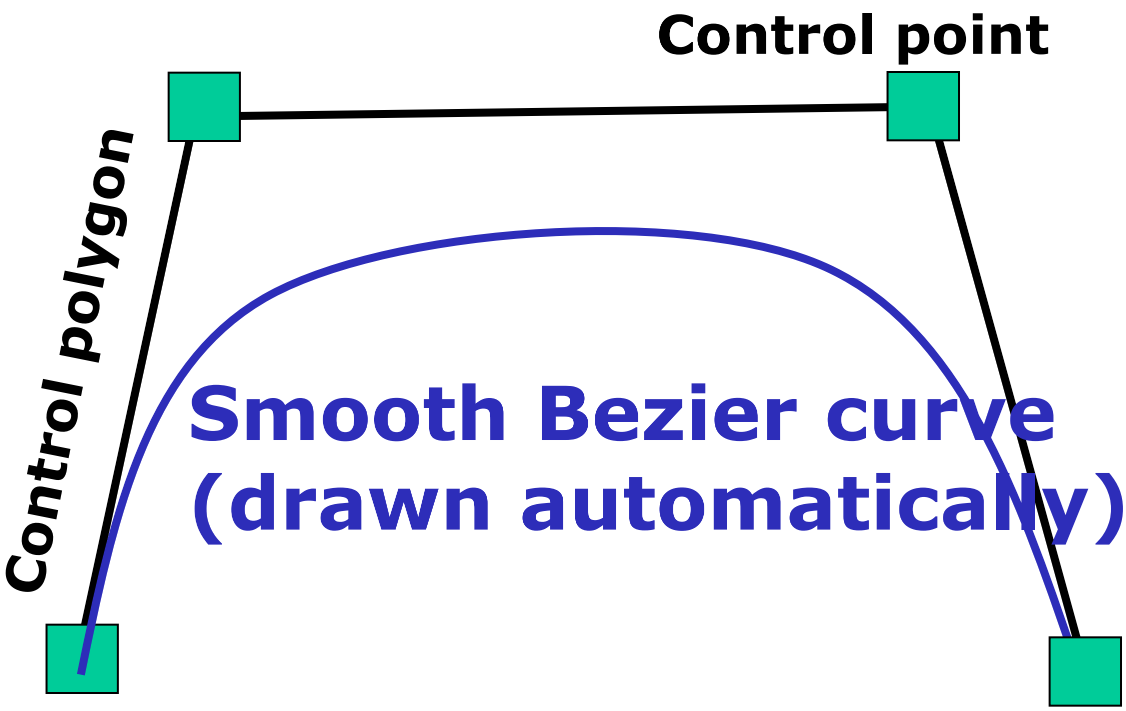
- Advantages of polynomial curves
  - Easy to compute
  - Infinitely differentiable everywhere

Spline Constructions

- Cubic spline is the most common form
- Common constructions
  - **Bezier:** 4 control points
  - **B-splines:** approximating C^2, local control
  - **Hermite:** 2 points, 2 normals
  - **Natural splines:** interpolating, C^2, no local control
  - **Catmull-Rom:** interpolating, C^1, local control

Bezier Curve

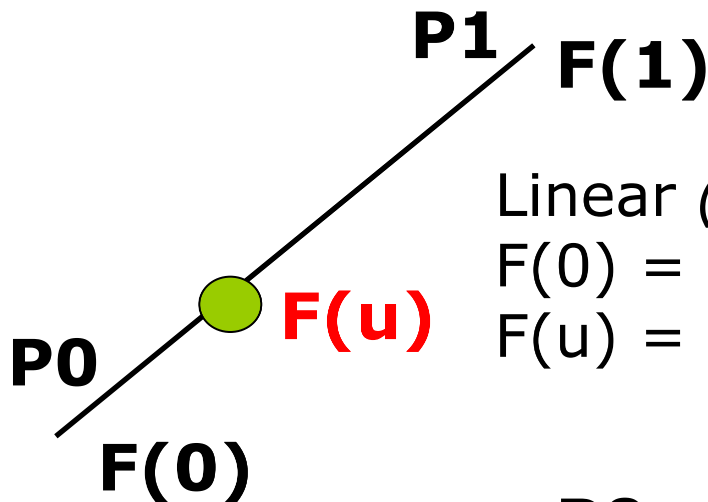
- **Motivation:** Draw a smooth intuitive curve (or surface) given a few key user-specified control points



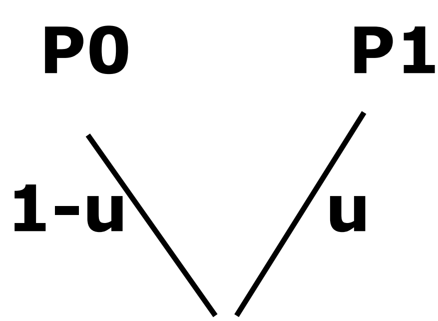
- **Properties:**
  - Interpolates is tangent to end points
  - Curve within convex hull of control polygon

Linear Bezier Curve

- Just a simple linear combination or interpolation (easy to code up, very numerically stable)

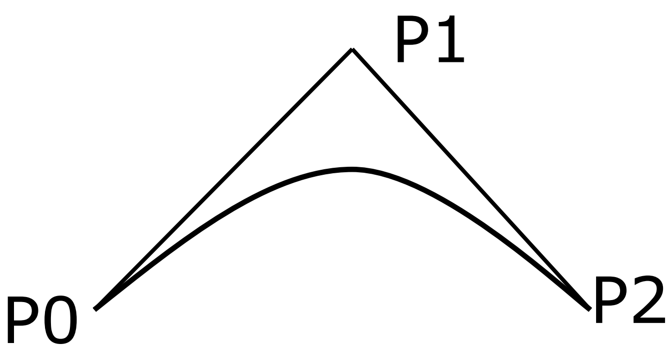


Linear *(Degree 1, Order 2)*
F(0) = P0, F(1) = P1
F(u) = ?



F(u) = (1-u) P0 + u P1

deCastljau: Quadratic Bezier Curve

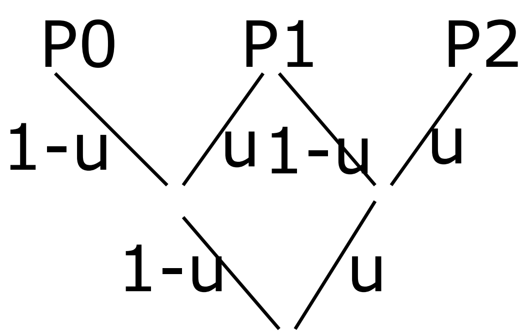


Quadratic

Degree 2, Order 3

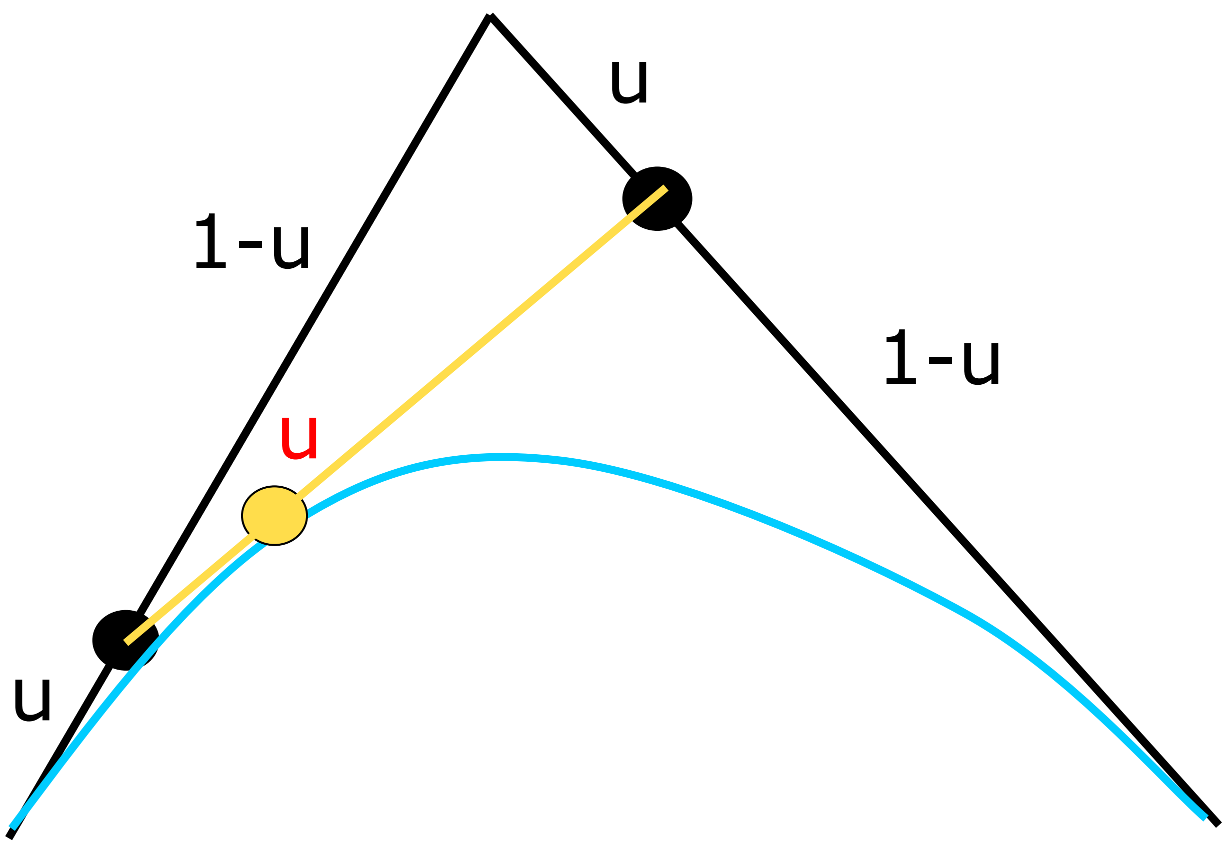
$F(0) = P0, F(1) = P2$

$F(u) = ?$

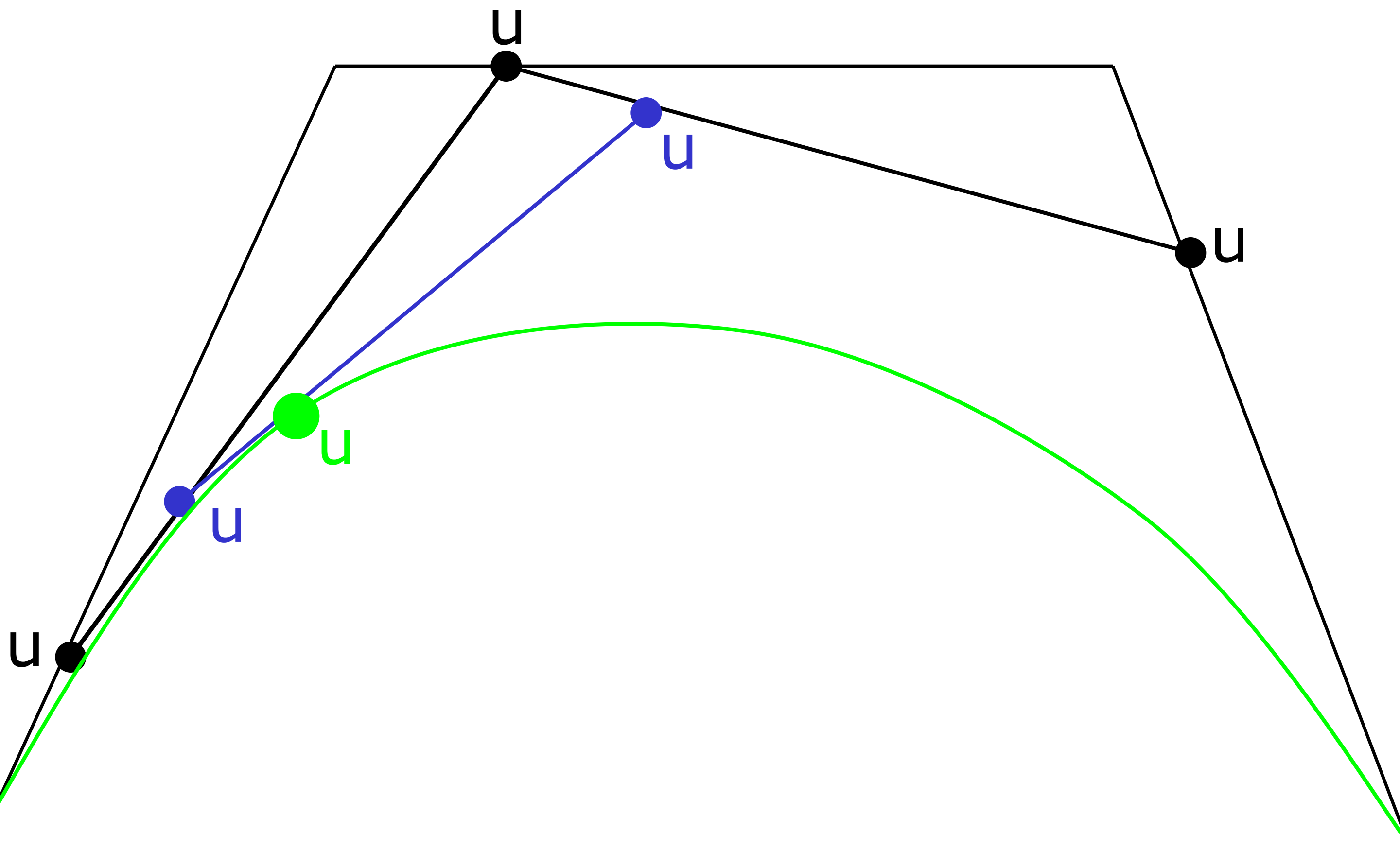


$$F(u) = (1-u)^2\, P0 + 2u(1-u)\, P1 + u^2\, P2$$

Geometric Interpretation: Quadratic

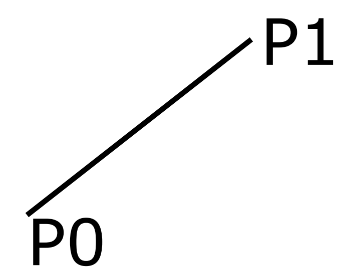


Geometric Interpolation: Cubic

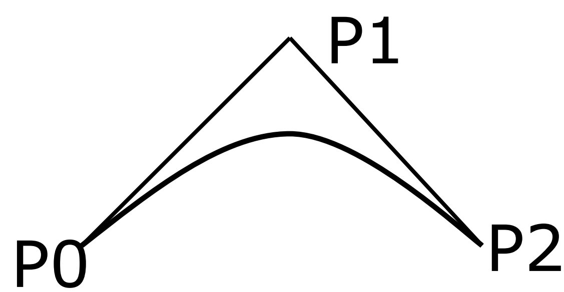


Summary: deCasteljau Algorithm

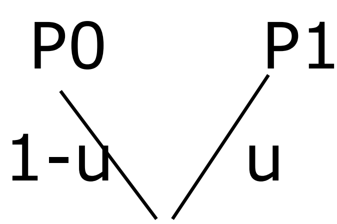
- A recursive implementation of curves at different orders



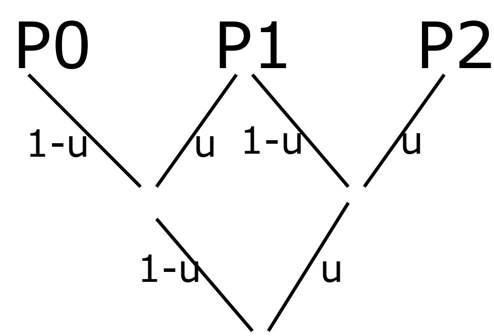
Linear
Degree 1, Order 2
F(0) = P0, F(1) = P1



Quadratic
Degree 2, Order 3
F(0) = P0, F(1) = P2



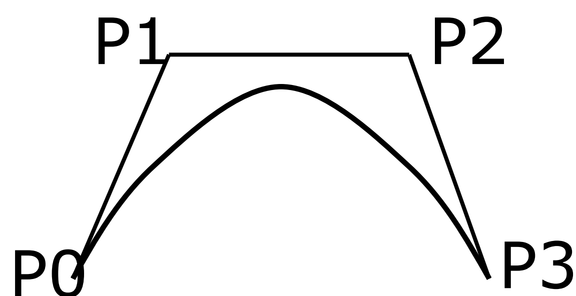
$$F(u) = (1-u)\,P0 + u\,P1$$



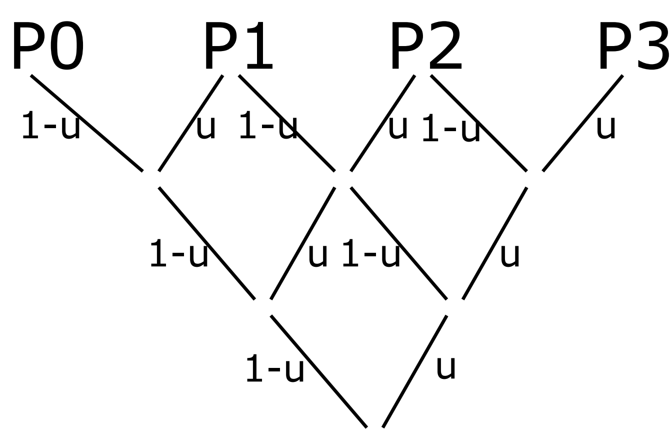
$$F(u) = (1-u)^2\,P0 + 2u(1-u)\,P1 + u^2\,P2$$

Summary: deCasteljau Algorithm

- A recursive implementation of curves at different orders
- Further consideration: polar coordinates



Cubic
Degree 3, Order 4
F(0) = P0, F(1) = P3



$F(u) = (1-u)^3 P0 + 3u(1-u)^2 P1 + 3u^2(1-u) P2 + u^3 P3$

Bezier: Disadvantages

- Single piece, no local control (move a control point, whole curve changes)
- **Complex shapes:** can be very high degree, difficult to deal with
- **In practice:** combine many Bezier curve segments
  - But only position continuous at the joint points since Bezier curves interpolate end-points (which match at segment boundaries)
  - Unpleasant derivative (slope) discontinuities at end-points

Piecewise polynomial curves

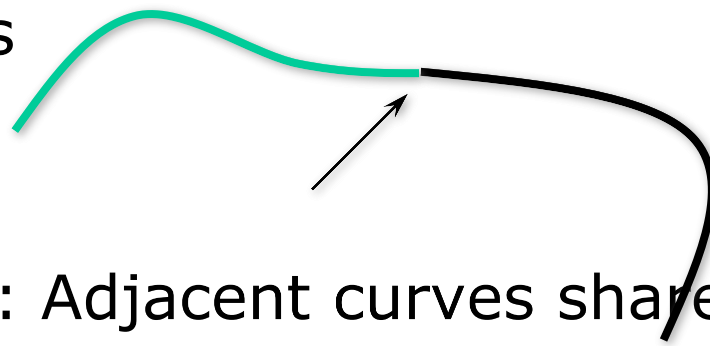
- **Ideas:**
  - Use different polynomial functions for different parts of the curve
- **Advantage:**
  - Flexibility
  - Local control
- **Issue**
  - Smoothness at joints

    (G: geometry continuity:

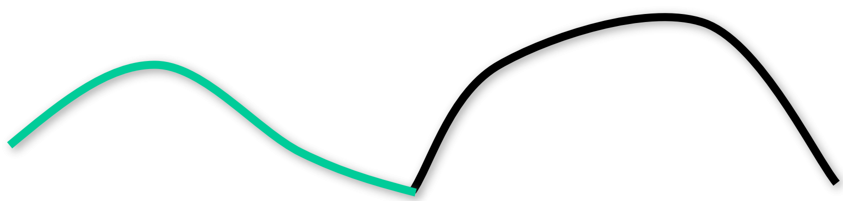
    C: derivative continuity)

Continuity

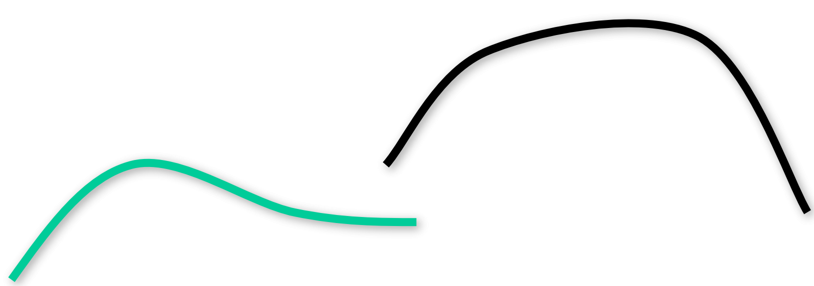
- Continuity C^k indicates adjacent curves have the same kth derivative at their joints



- C^0 continuity: Adjacent curves share
  - Same endpoints: $Q_i(1) = Q_{i+1}(0)$

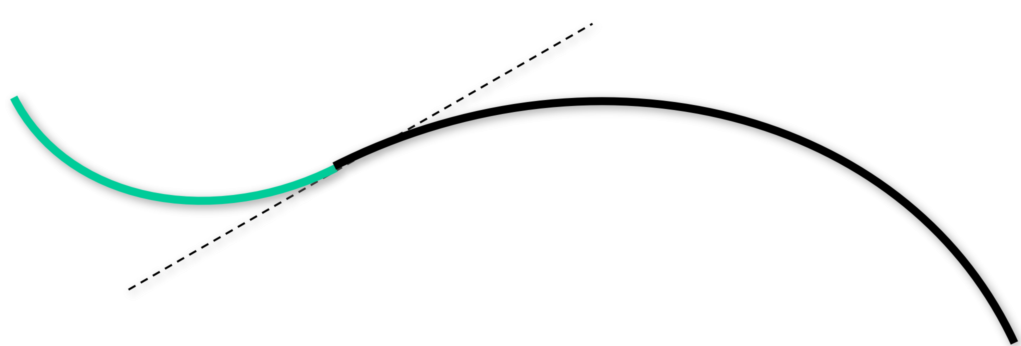


- C^{-1}: discontinuous curves



Continuity

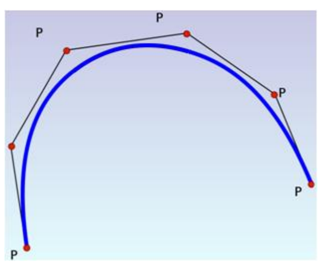
- **C^1 continuity:** Adjacent curves share
  - Same endpoints: $Q_i(1) = Q_{i+1}(0)$ and
  - Same derivative: $Q_i'(1) = Q_{i+1}'(0)$
- **C^2 continuity:**
  - Must have C^1 continuity, and
  - Same second derivatives:
    $Q_i''(1) = Q_{i+1}''(0)$
- Most engineering applications (e.g., those in car and airplane industry) require at least C^1 continuity

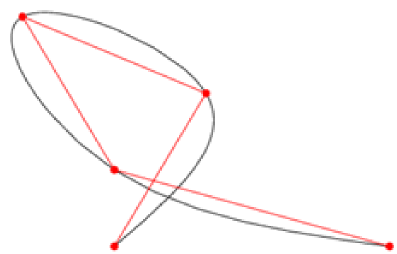


Splines

- More useful form of representation compared to the Bezier curve
- **How they work:** Parametric curves governed by control points
- **Mathematically:** Several representations to choose from. More complicated than vertex lists. See chapter 22 of the book for more information.

  Simple parametric representation:





- **Advantage:** Smooth with just a few control point
- **Disadvantage:** Can be hard to control
- **Uses:**
  - representation of smooth shapes. Either as outlines in 2D or with Patches or Subdivision Surfaces in 3D
  - animation Paths
  - approximation of truncated Gaussian Filters

A Simple Animation Example

- **Problem:** create a car animation that is driving up along the y-axis with velocity [0, 3], and arrive at the point (0, 4) at time t=0. Animate its motion as it turns and slows down so that at time t=1, it is at position (2, 5) with velocity [2, 0].

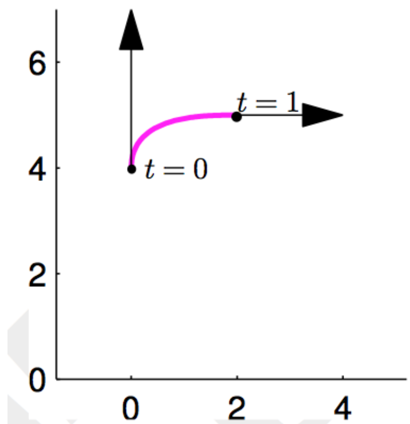


Figure 22.1: Animating a car's motion. Given the initial and final point and velocity, we want to find a path like the magenta curve.

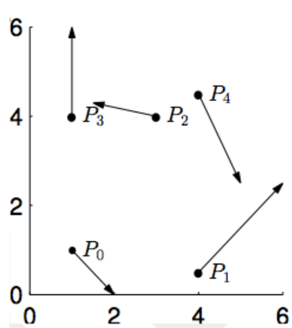
- **Solution**
  - **First step:** generate a mathematical description.
  - **Second step:** choose the curve representation
    - Hermite curve: r(t)=**GMT**(t)

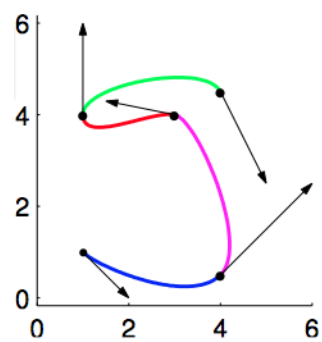
- **Exercise:** Bezier curve representation?

Catmull Rom Spline

- Can be used to solve the following problem.

Figure 22.4: A sequence of points and vectors; we want a curve that passes through the points with the given vectors as velocities.





- Solution:
  - Math representation
  - Curve construction
    - Catmull Rom spline to construct the vectors from the two or three neighbors

take home exercise: read chap 22 in the book and construct the curve and the B-spline using the Chen code.

Subdivision curves

- A simple idea
  - Using the midpoint of the edge from one point to the next, replace that point with a new one to create a new polygon to construct a new curve.
  - problem with this?
- Further readings:
  - Laplacian interpolation and smoothing (Gabriel Taubin @ Brown)
  - Joe Warren@ Rice (on mesh)

Surfaces

- Curves -> Surfaces
- Bezier patch:
  - 16 points
  - Check out the Chen code for surface construction