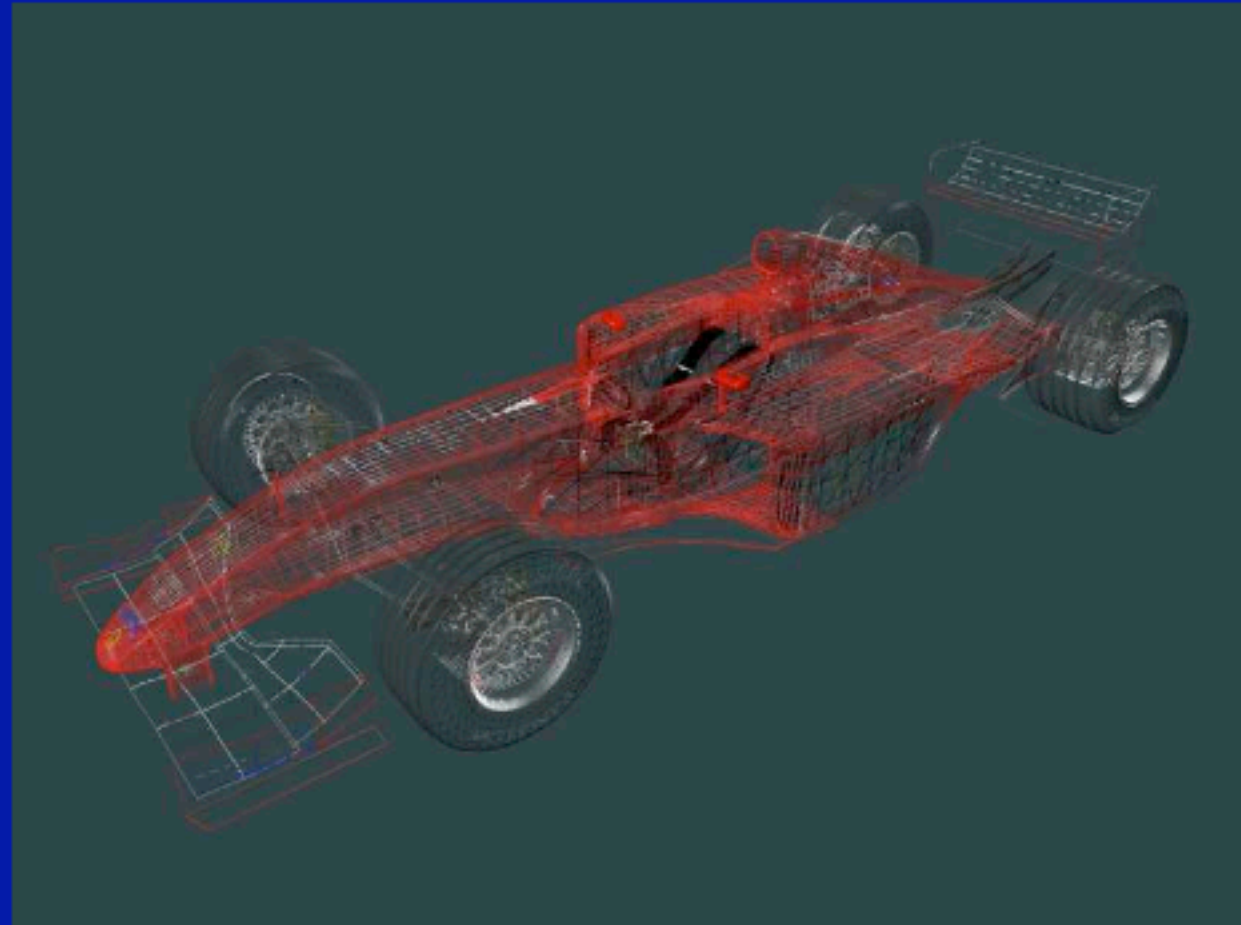


# Modeling Complex Shapes

- We want to build models of very complicated objects
- An equation for a sphere is possible, but how about an equation for a telephone, or a face?
- Complexity is achieved using simple pieces
  - polygons, parametric curves and surfaces, or implicit curves and surfaces
  - This lecture: parametric curves



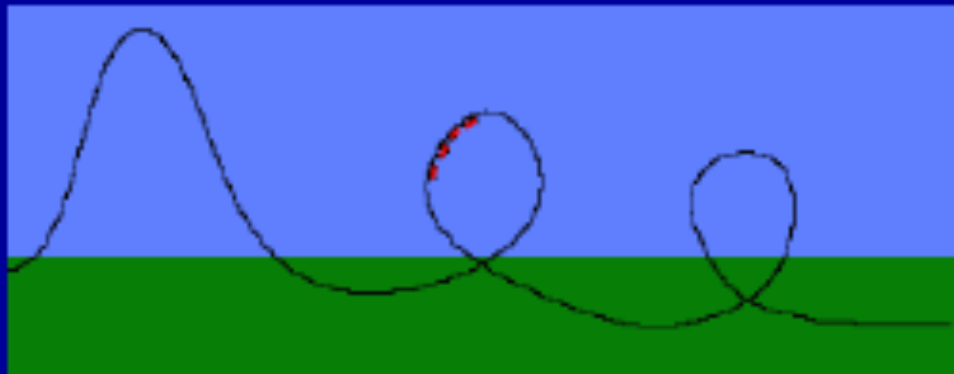
# Parametric Curves

## **Modeling:**

- **parametric curves (Splines)**
- **polygonal meshes**

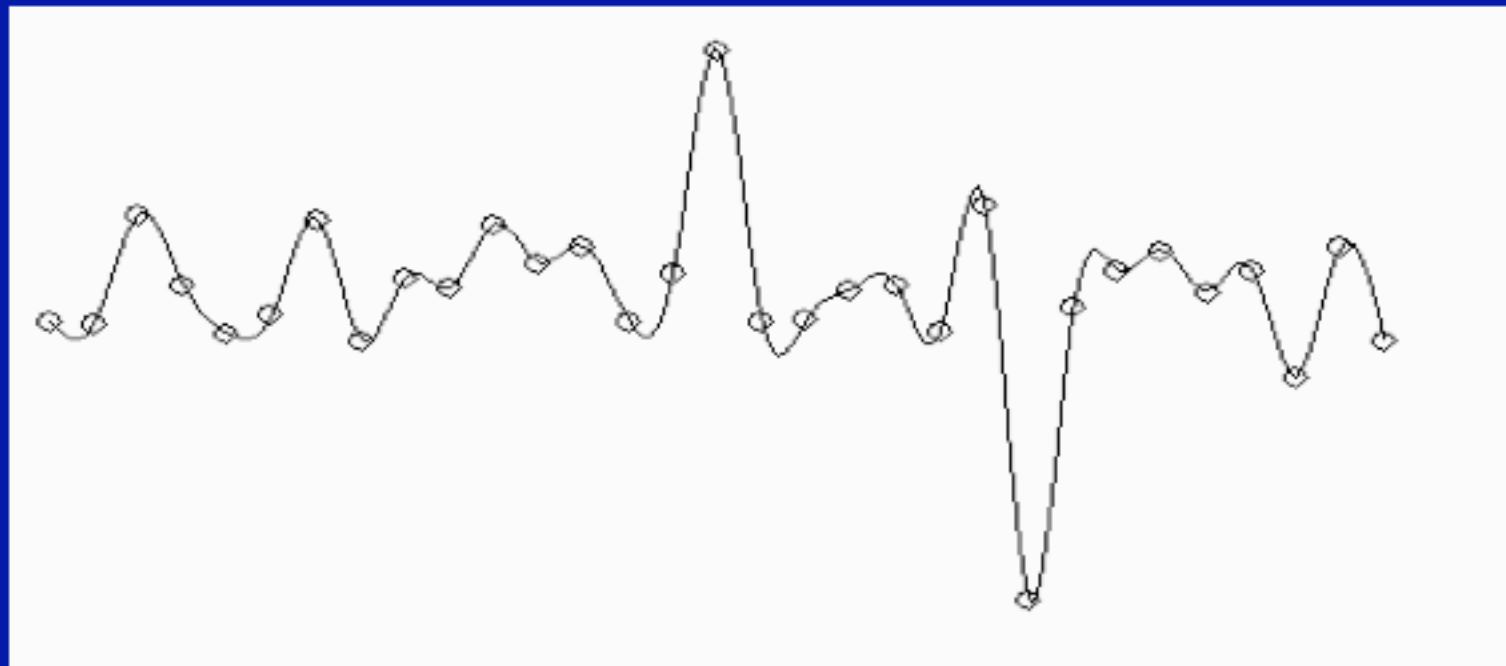
# Roller coaster

- We must model the 3D curve describing the roller coaster, but how?
- How to make the simulation obey the laws of gravity?



# What Do We Need From Curves in Computer Graphics?

- Local control of shape (so that easy to build and modify)
- Stability
- Smoothness and continuity
- Ability to evaluate derivatives
- Ease of rendering

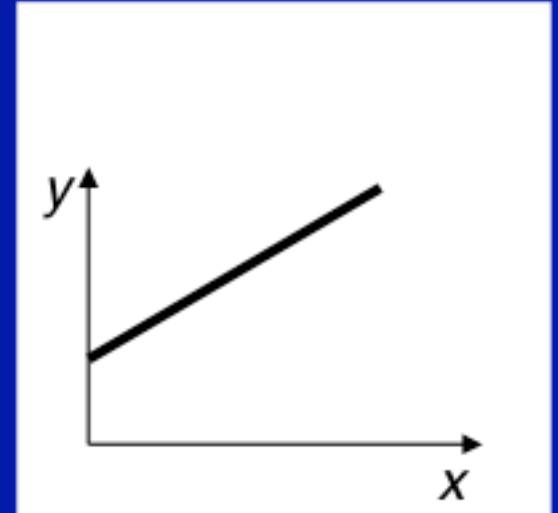


# Curve Representations

- **Explicit:  $y = f(x)$**

$$y = mx + b$$

- Easy to generate points
- Must be a function: big limitation—vertical lines?

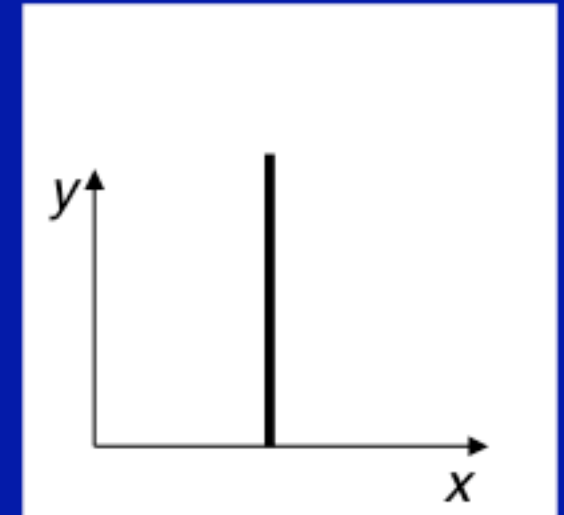


# Curve Representations

- **Explicit:  $y = f(x)$**

$$y = mx + b$$

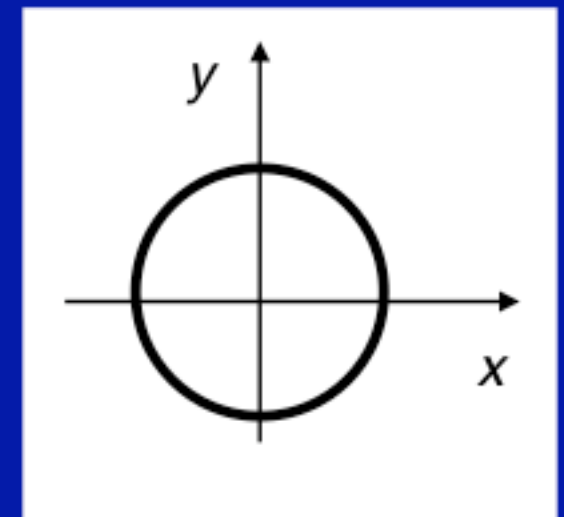
- Easy to generate points
- Must be a function: big limitation—vertical lines?



- **Implicit:  $f(x,y) = 0$**

$$x^2 + y^2 - r^2 = 0$$

- +Easy to test if on the curve
- Hard to generate points



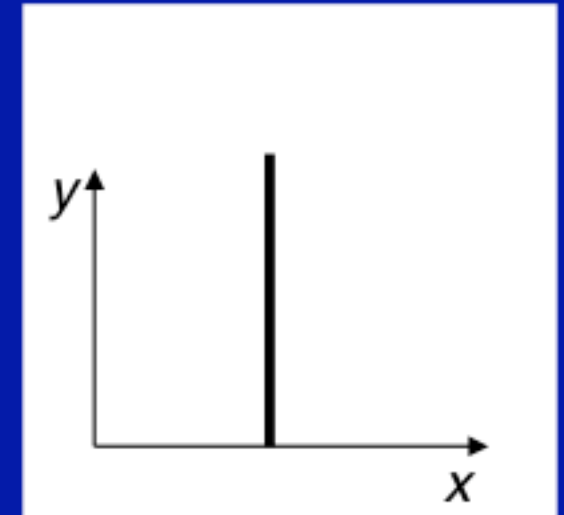
# Curve Representations

- **Explicit:**  $y = f(x)$

$$y = mx + b$$

+ Easy to generate points

– Must be a function: big limitation—vertical lines?

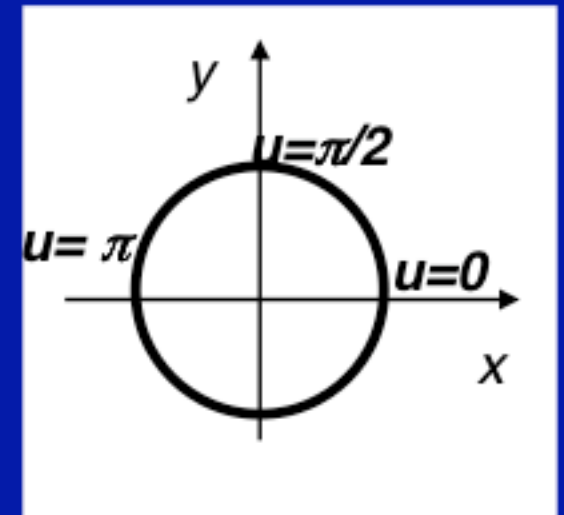


- **Implicit:**  $f(x,y) = 0$

$$x^2 + y^2 - r^2 = 0$$

+ Easy to test if on the curve

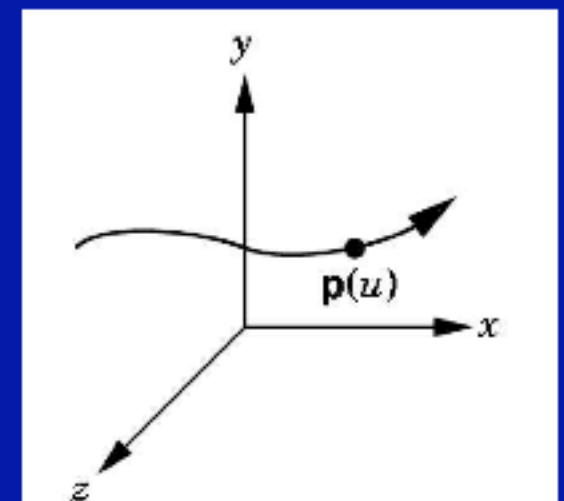
– Hard to generate points



- **Parametric:**  $(x,y) = (f(u), g(u))$

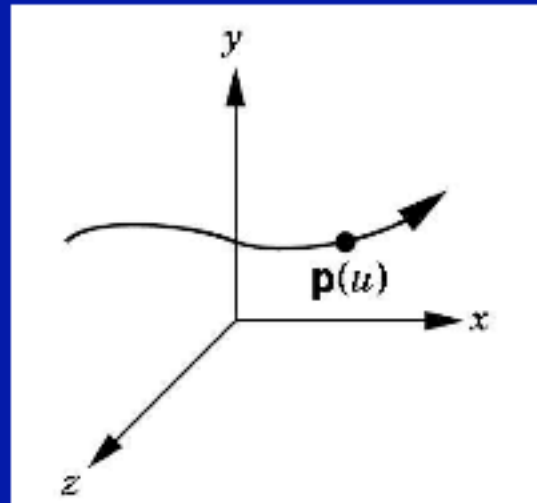
$$(x, y) = (\cos u, \sin u)$$

+ Easy to generate points



# Parameterization of a Curve

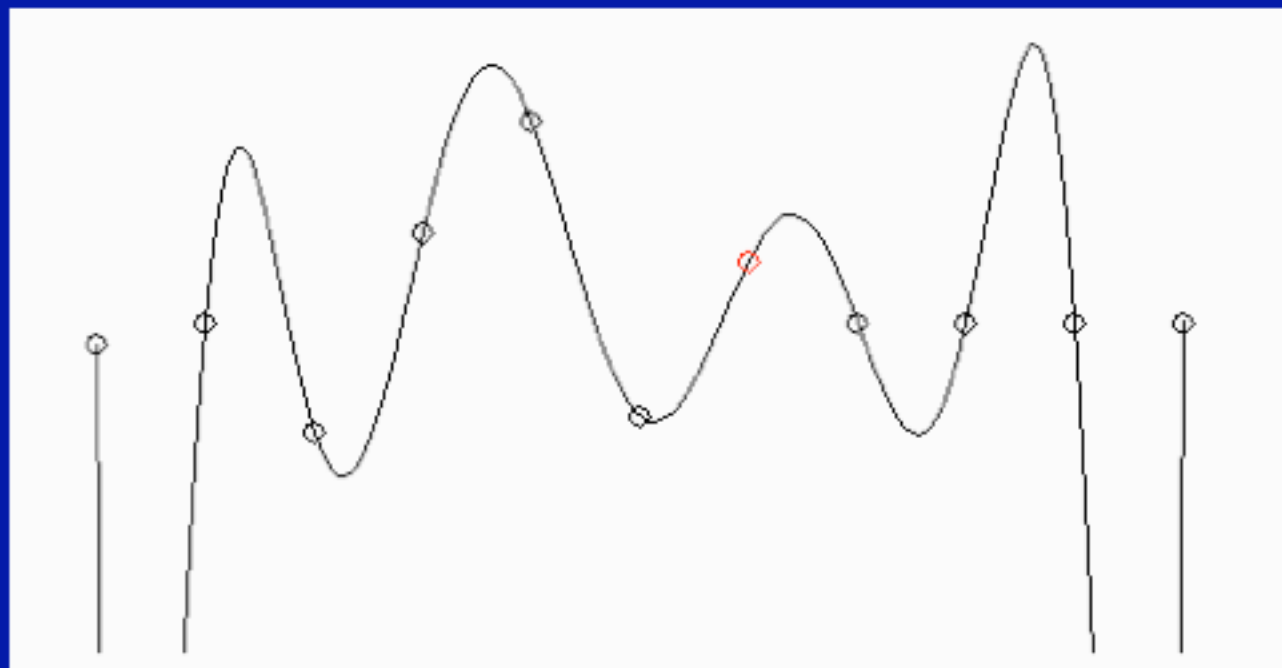
- *Parameterization* of a curve: how a change in  $u$  moves you along a given curve in  $xyz$  space.





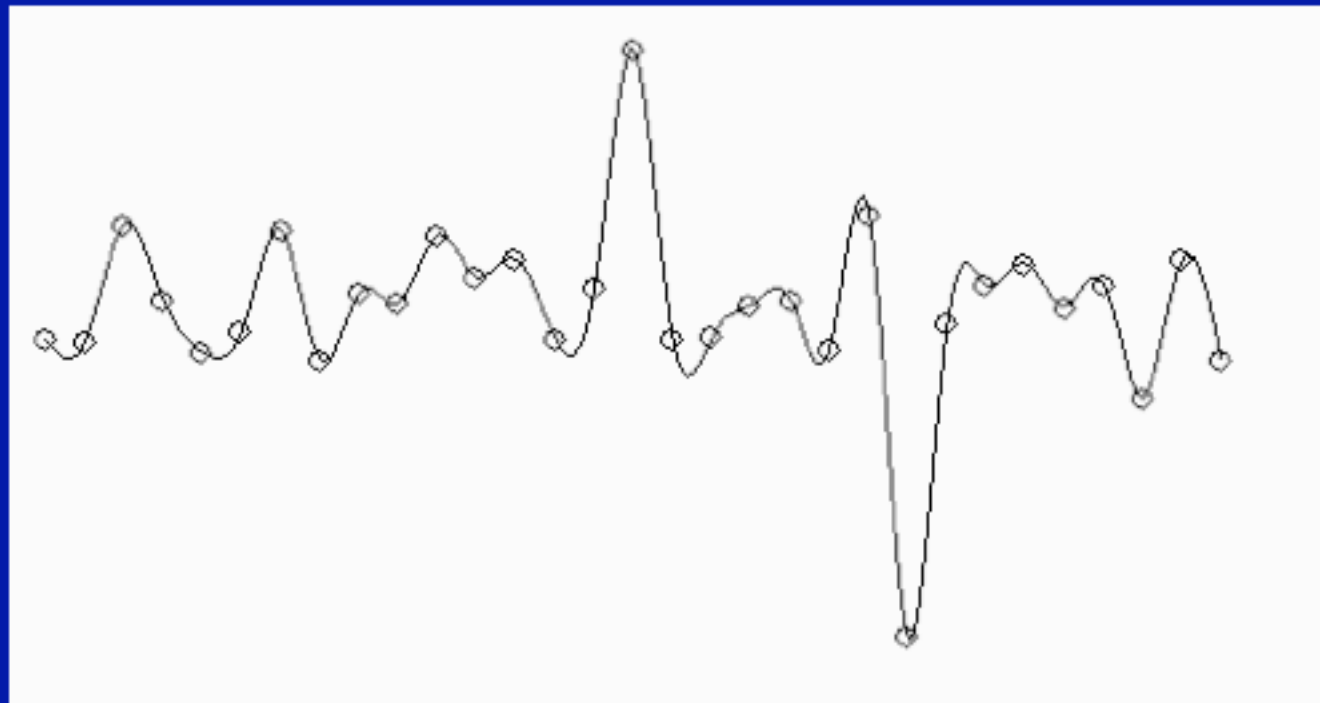
# Polynomial Interpolation

- An  $n$ -th degree polynomial fits a curve to  $n+1$  points
  - called Lagrange Interpolation
  - result is a curve that is too wiggly, change to any control point affects entire curve (nonlocal) – *this method is poor*
- We usually want the curve to be as smooth as possible
  - minimize the wiggles
  - high-degree polynomials are bad



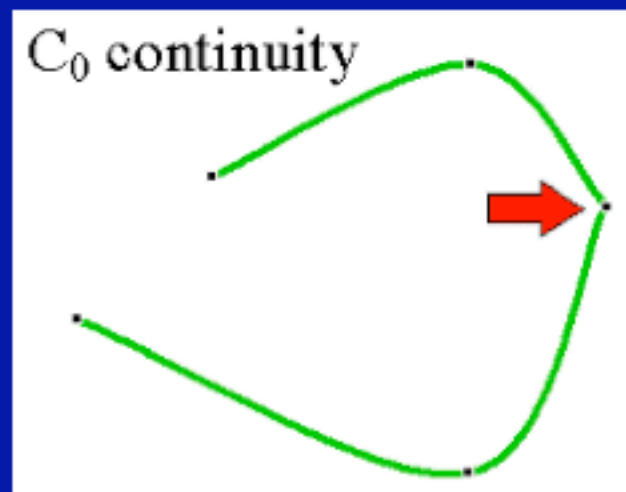
# Splines: Piecewise Polynomials

- A spline is a *piecewise polynomial* - many low degree polynomials are used to interpolate (pass through) the control points
- *Cubic piecewise* polynomials are the most common:
  - piecewise definition gives local control

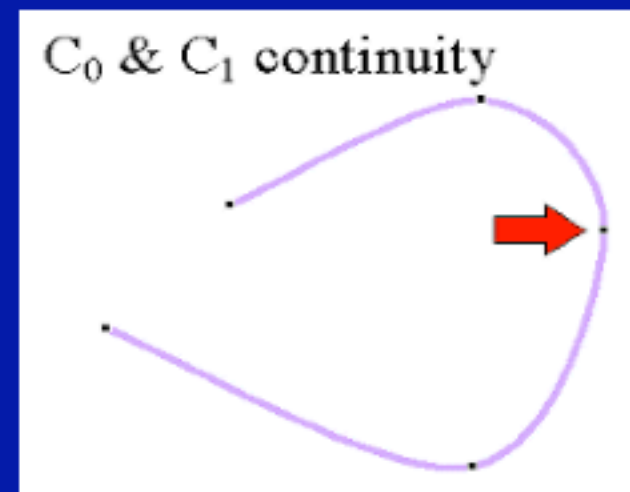


# Piecewise Polynomials

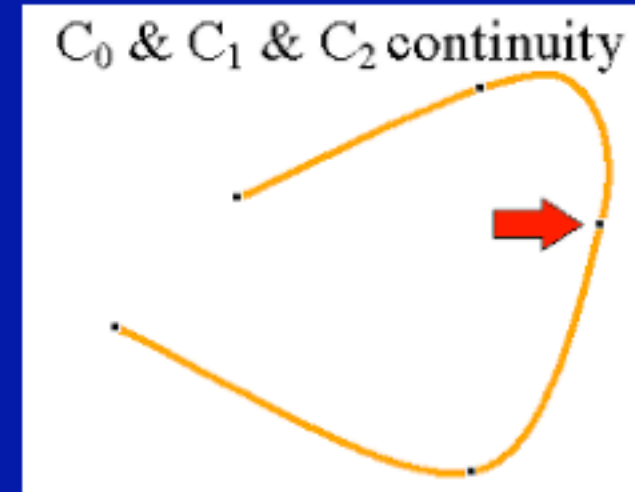
- **Spline:** lots of little polynomials pieced together
- **Want to make sure they fit together nicely**



Continuous in  
position



Continuous in  
position and tangent  
vector



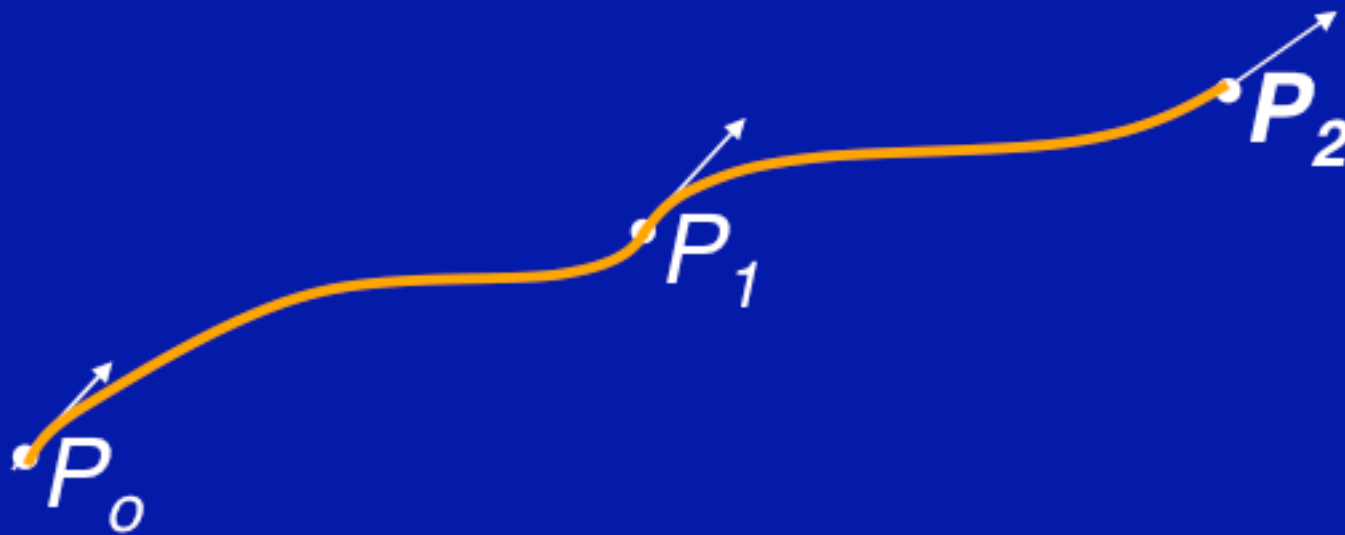
Continuous in  
position, tangent,  
and curvature

# Splines

- **Types of splines:**
  - Hermite Splines
  - Catmull-Rom Splines
  - Bezier Splines
  - Natural Cubic Splines
  - B-Splines
  - NURBS

# Hermite Curves

- Cubic Hermite Splines



**That is, we want a way to specify the end points and the slope at the end points!**

# Splines

chalkboard

# The Cubic Hermite Spline Equation

- Using some algebra, we obtain:

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

**point that  
gets drawn**

**basis**

**control matrix  
(what the user gets to pick)**

- This form typical for splines
  - basis matrix and meaning of control matrix change with the spline type

# The Cubic Hermite Spline Equation

- Using some algebra, we obtain:

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

point that  
gets drawn

basis

control matrix  
(what the user gets to pick)

$$p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

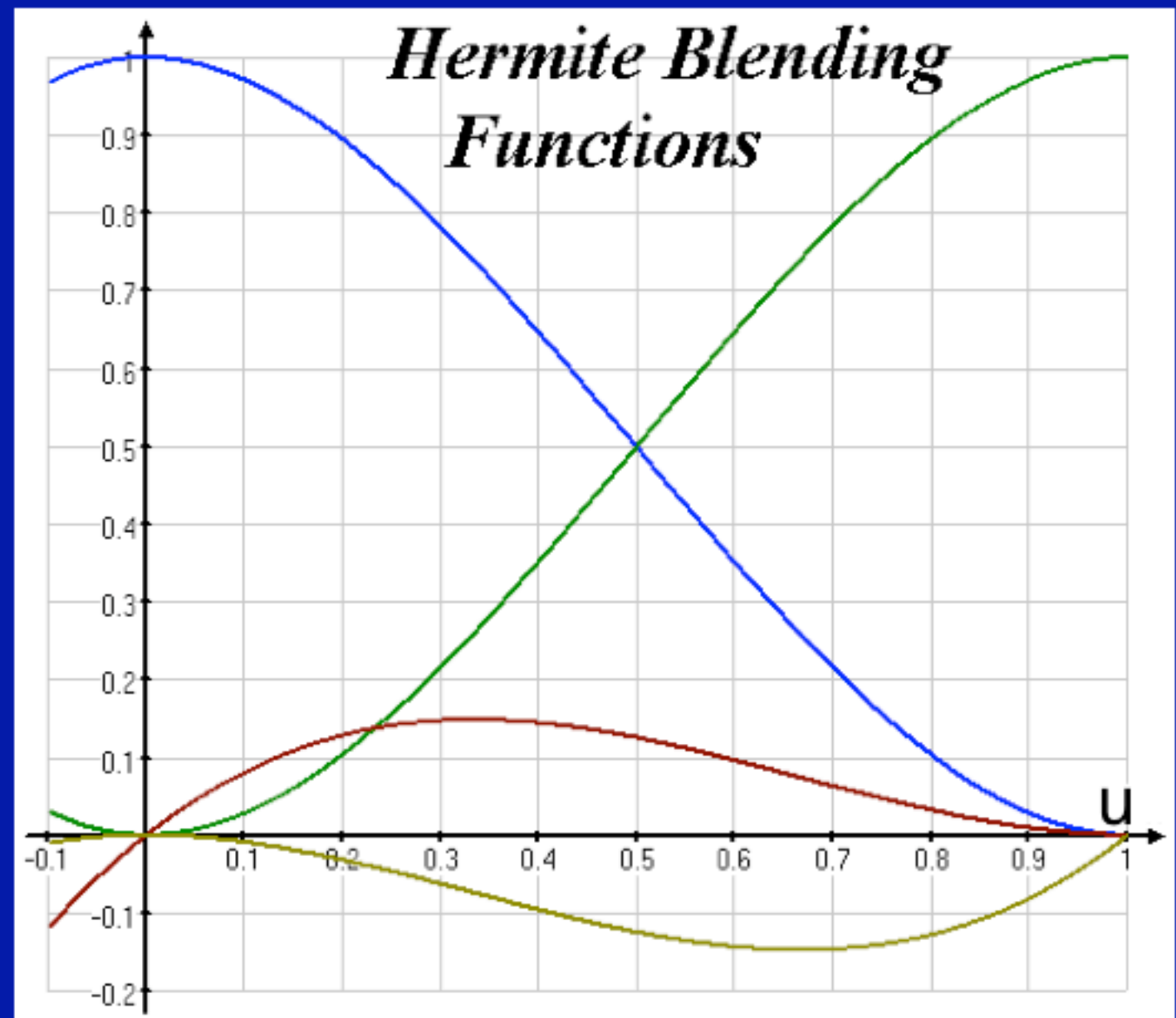
**4 Basis Functions**



# Four Basis Functions for Hermite splines

$$p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

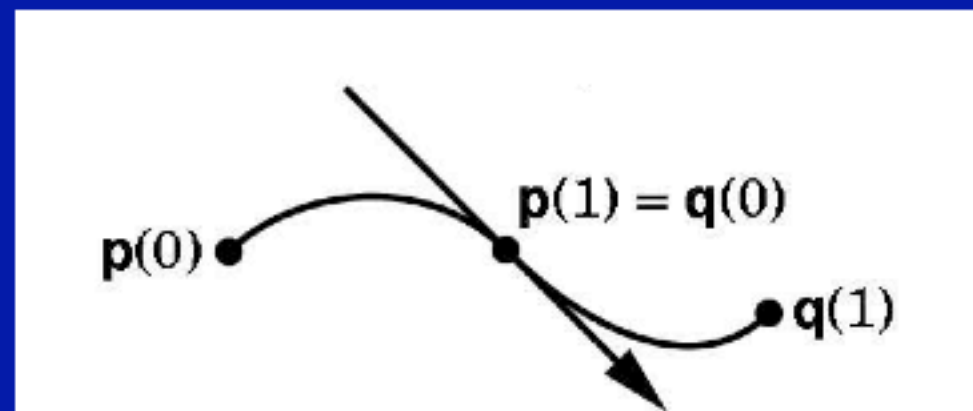
↑  
**4 Basis Functions**



Every cubic Hermite spline is a linear combination (blend) of these 4 functions

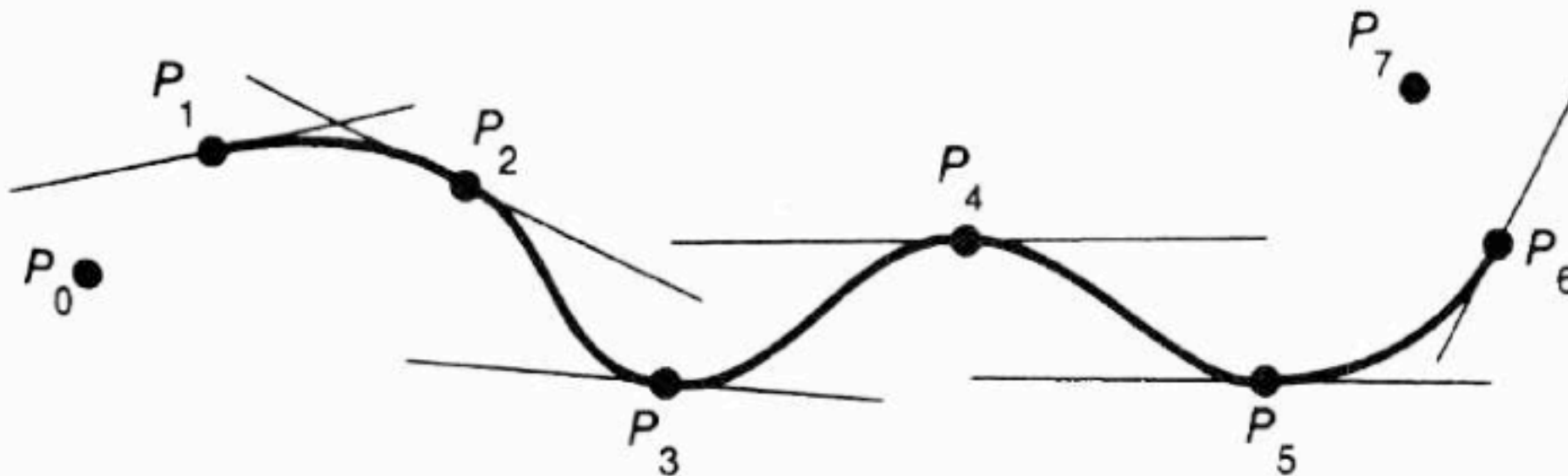
# Piecing together Hermite Curves

- It's easy to make a multi-segment Hermite spline
  - each piece is specified by a cubic Hermite curve
  - just specify the position and tangent at each “joint”
  - the pieces fit together with matched positions and first derivatives
  - gives C1 continuity
- The points that the curve has to pass through are called *knots* or *knot points*



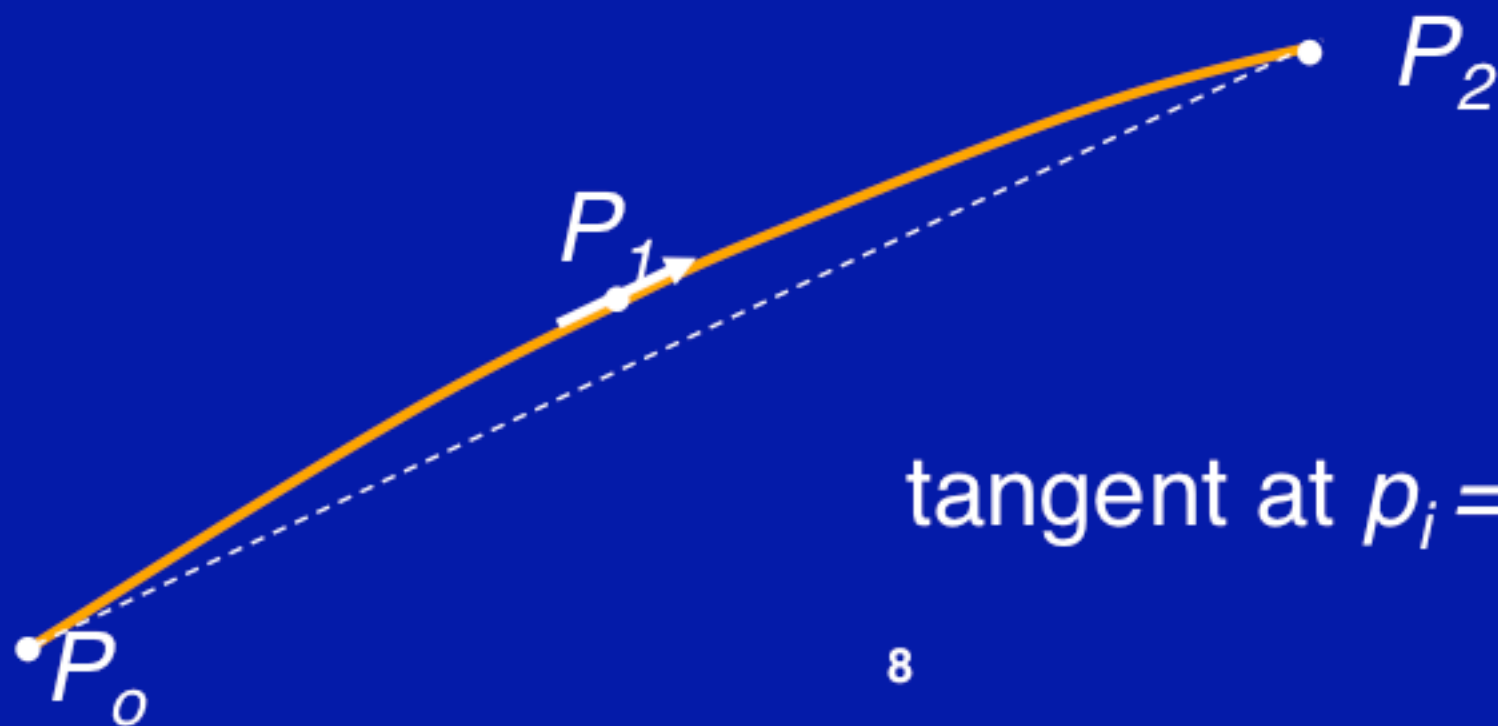
# Catmull-Rom Splines

- With Hermite splines, the designer must specify all the tangent vectors
- Catmull-Rom: an interpolating cubic spline with *built-in  $C^1$  continuity*.



# Catmull-Rom Splines

- With Hermite splines, the designer must specify all the tangent vectors
- Catmull-Rom: an interpolating cubic spline with *built-in  $C^1$  continuity*.



chalkboard

$$\text{tangent at } p_i = s(p_{i+1} - p_{i-1})$$

# Catmull-Rom Spline Matrix

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

**spline coefficients**                      **CR basis**                      **control vector**

- **Derived similarly to Hermite**
- **Parameter s is typically set to s=1/2.**

# Cubic Curves in 3D

- **Three cubic polynomials, one for each coordinate**

$$- x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$- y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

$$- z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

- **In matrix notation**

$$\begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

# Catmull-Rom Spline Matrix in 3D

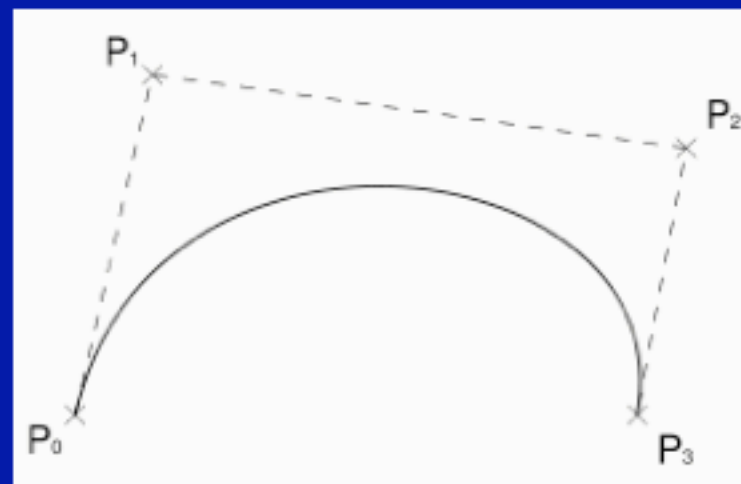
$$\begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}$$

**CR basis**

**control vector**

# Bezier Curves\*

- **Another variant of the same game**
- **Instead of endpoints and tangents, four control points**
  - points  $P_0$  and  $P_3$  are on the curve:  $P(u=0) = P_0$ ,  $P(u=1) = P_3$
  - points  $P_1$  and  $P_2$  are off the curve
  - $P'(u=0) = 3(P_1 - P_0)$ ,  $P'(u=1) = 3(P_3 - P_2)$
- **Convex Hull property**
  - curve contained within convex hull of control points
- **Gives more control knobs (series of points) than Hermite**
- **Scale factor (3) is chosen to make “velocity” approximately constant**



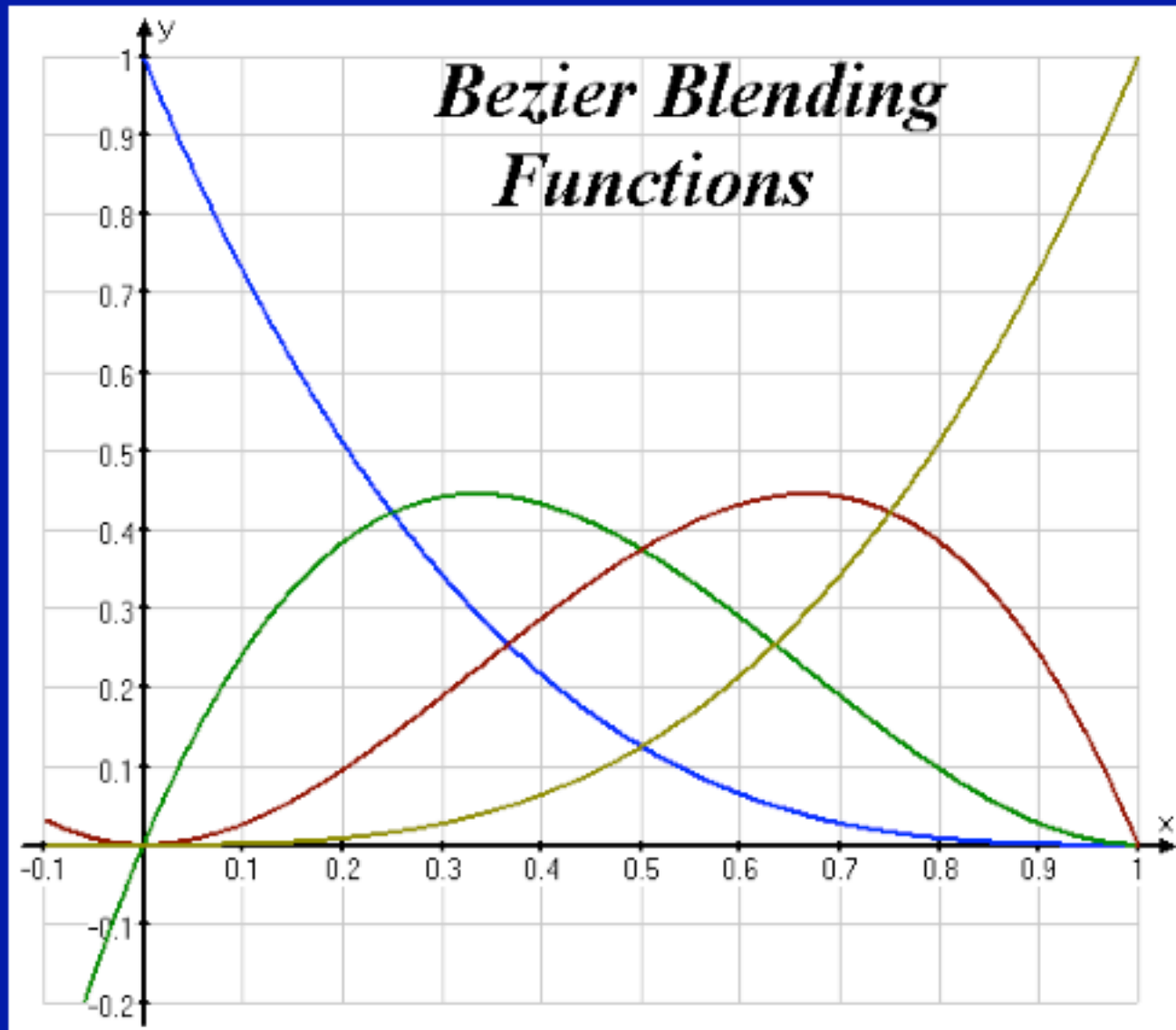


# The Bezier Spline Matrix\*

$$[x \ y \ z] = [u^3 \ u^2 \ u \ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}$$

**Bezier basis**                      **Bezier control vector**

# Bezier Blending Functions\*



$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

Also known as the order 4, degree 3 Bernstein polynomials

Nonnegative, sum to 1

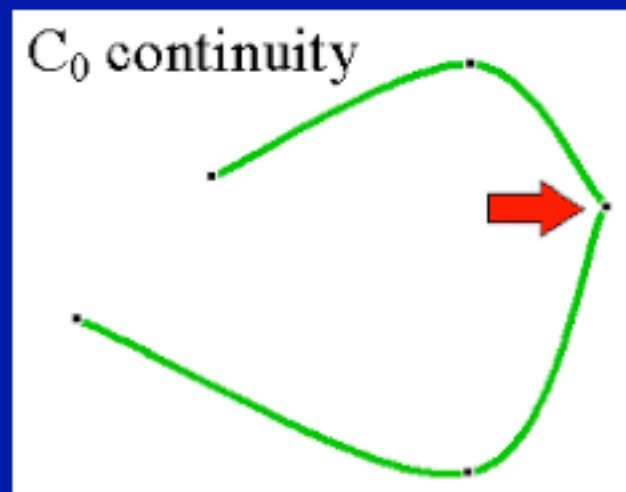
The entire curve lies inside the polyhedron bounded by the control points

# Splines with More Continuity?

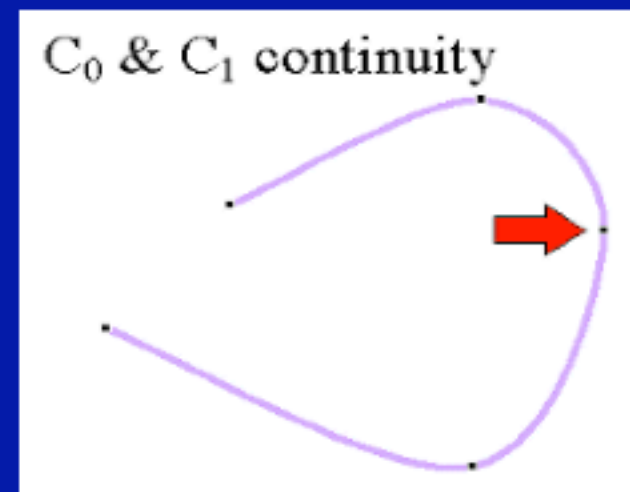
- How could we get  $C^2$  continuity at control points?
- Possible answers:
  - Use higher degree polynomials  
degree 4 = quartic, degree 5 = quintic, ... but these get computationally expensive, and sometimes wiggly
  - Give up local control      natural cubic splines  
A change to any control point affects the entire curve
  - Give up interpolation      cubic B-splines  
Curve goes near, but not through, the control points

# Piecewise Polynomials

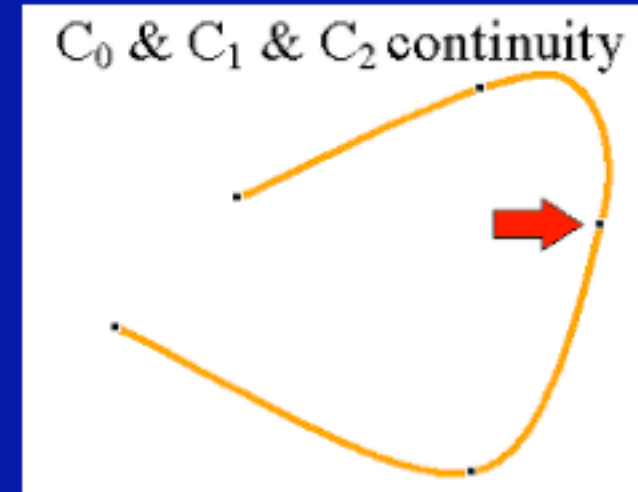
- **Spline:** lots of little polynomials pieced together
- **Want to make sure they fit together nicely**



Continuous in position



Continuous in position and tangent vector



Continuous in position, tangent, and curvature



# Comparison of Basic Cubic Splines

Type	Local Control	Continuity	Interpolation
Hermite	YES	C1	YES
Bezier	YES	C1	YES
Catmull-Rom	YES	C1	YES
Natural	NO	C2	YES
B-Splines	YES	C2	NO

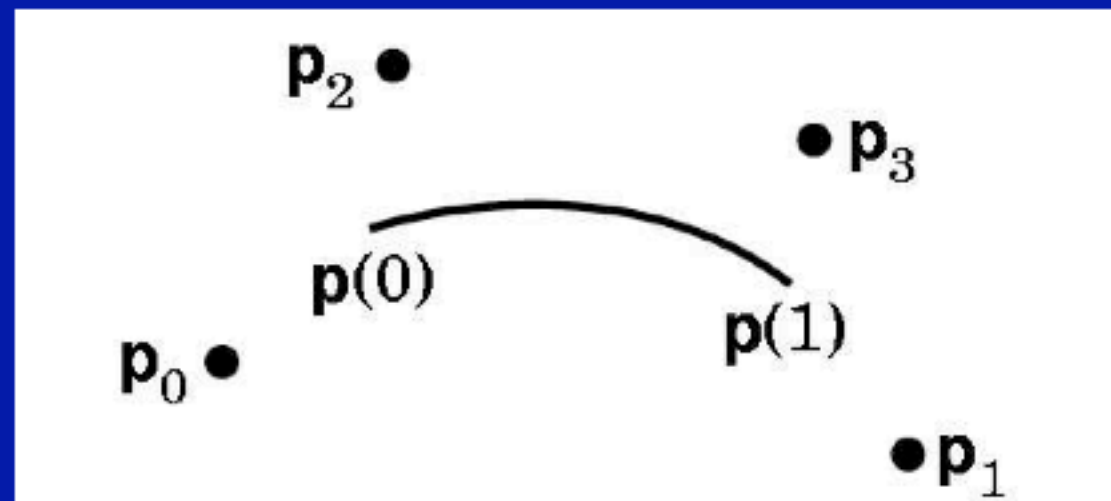
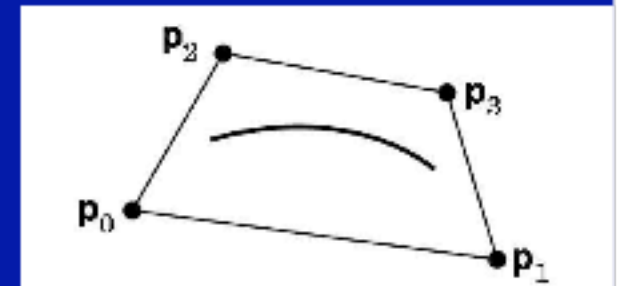
- **Summary**
  - Can't get C2, interpolation and local control with cubics

# Natural Cubic Splines\*

- If you want 2nd derivatives at joints to match up, the resulting curves are called *natural cubic splines*
- It's a simple computation to solve for the cubics' coefficients. (See *Numerical Recipes in C* book for code.)
- Finding all the right weights is a *global* calculation (solve tridiagonal linear system)

# B-Splines\*

- Give up interpolation
  - the curve passes *near* the control points
  - best generated with interactive placement (because it's hard to guess where the curve will go)
- Curve obeys the convex hull property
- C2 continuity and local control are good compensation for loss of interpolation



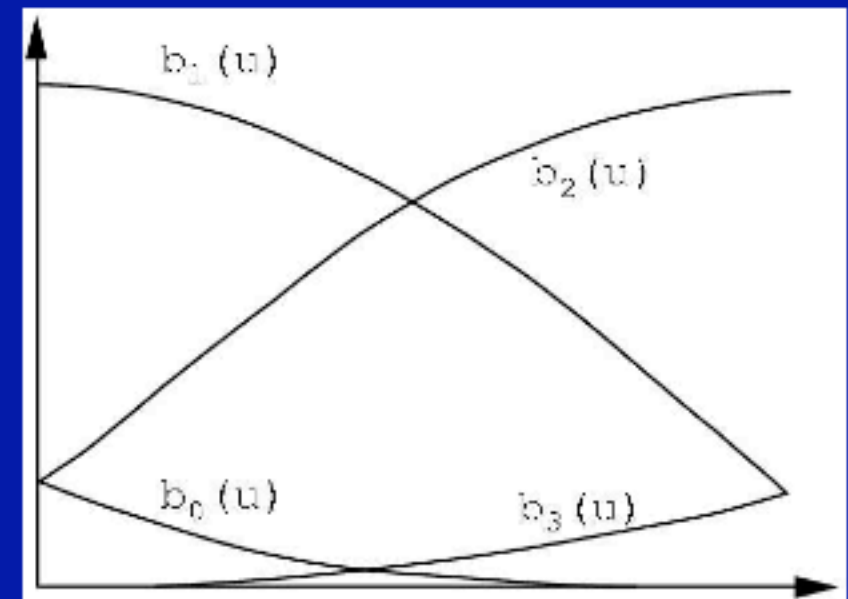


# B-Spline Basis\*

- We always need 3 more control points than spline pieces

$$M_{Bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

$$G_{Bsi} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$





# How to Draw Spline Curves

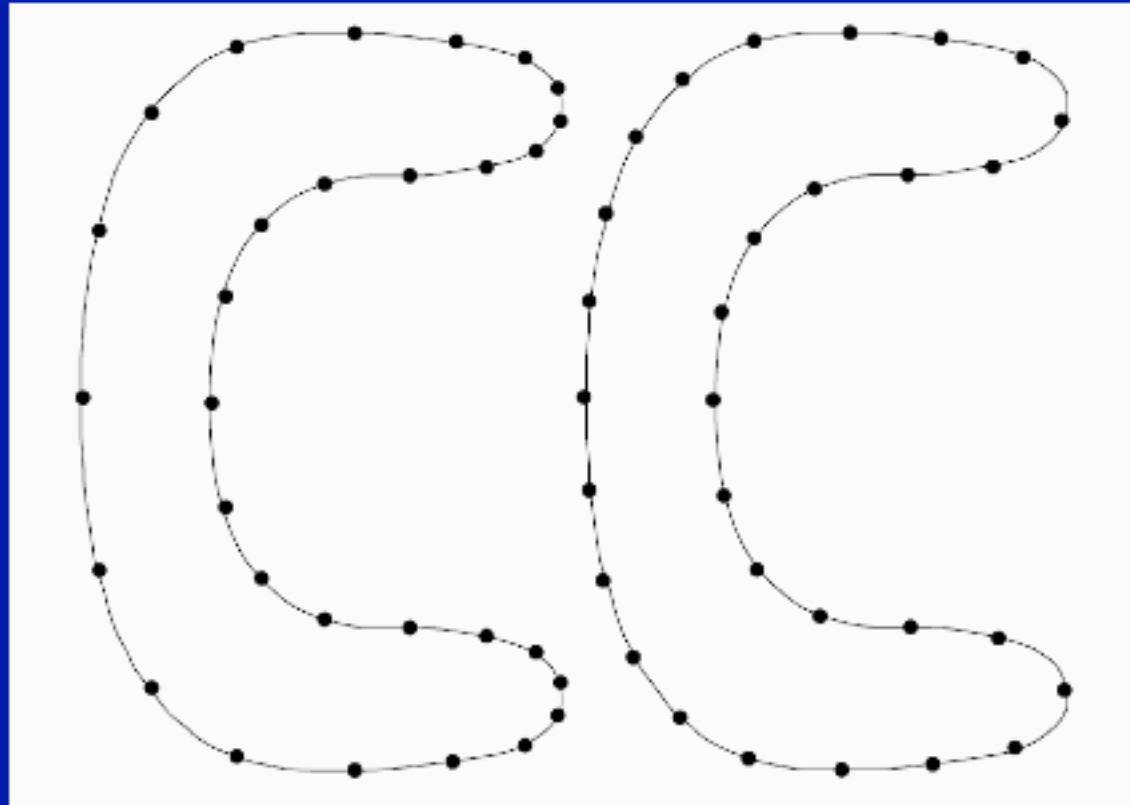
- **Basis matrix eqn. allows same code to draw any spline type**
- **Method 1: brute force**
  - Calculate the coefficients
  - For each cubic segment, vary  $u$  from 0 to 1 (fixed step size)
  - Plug in  $u$  value, matrix multiply to compute position on curve
  - Draw line segment from last position to current position

$$[x \ y \ z] = [u^3 \ u^2 \ u \ 1] \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}$$

**CR basis**                      **control vector**

# How to Draw Spline Curves

- **What's wrong with this approach?**
  - Draws in even steps of  $u$
  - Even steps of  $u \neq$  even steps of  $x$
  - Line length will vary over the curve
  - Want to bound line length
    - »too long: curve looks jagged
    - »too short: curve is slow to draw



# Drawing Splines, 2

- **Method 2: recursive subdivision** - vary step size to draw short lines

```
Subdivide(u0,u1,maxlinelength)
  umid = (u0 + u1)/2
  x0 = P(u0)
  x1 = P(u1)
  if |x1 - x0| > maxlinelength
    Subdivide(u0,umid,maxlinelength)
    Subdivide(umid,u1,maxlinelength)
  else drawline(x0,x1)
```

- **Variant on Method 2** - subdivide based on curvature
  - replace condition in “if” statement with straightness criterion
  - draws fewer lines in flatter regions of the curve



# In Summary...

- **Summary:**
  - piecewise cubic is generally sufficient
  - define conditions on the curves and their continuity
- **Things to know:**
  - basic curve properties (what are the conditions, controls, and properties for each spline type)
  - generic matrix formula for uniform cubic splines  $x(u) = uBG$
  - given definition derive a basis matrix