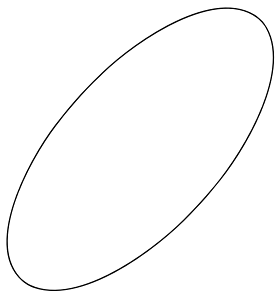
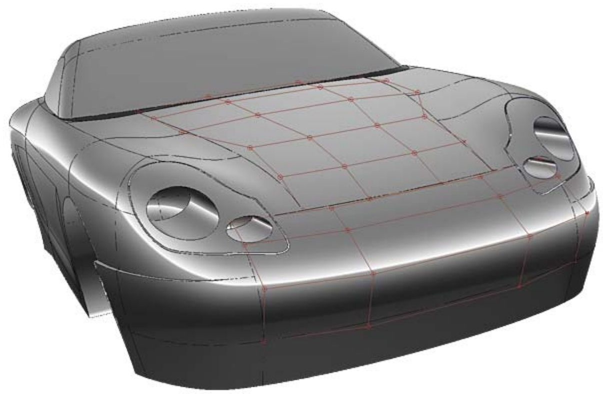
Geometric Modeling

Curves and surfaces





2D Curves: Implicit Form

Point (x,y) lies on the curve iff it satisfies

$$F(x,y)=0$$

- ***Line*** through points **a** = (a_x,a_y) and **b** = (b_x,b_y)

$$F(x,y)=(y-a_y)(b_x-a_x)-(x-a_x)(b_y-a_y)=0$$

- ***Circle*** with radius r centered at **c =** (c_x,c_y)

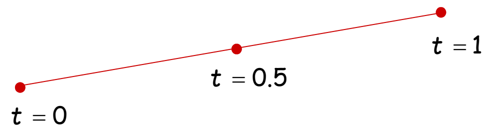
$$F(x,y)=(x-c_x)^2+(y-c_y)^2-r^2=0$$

2D Curves: Parametric Form

A line **through points** $a = (a_x, a_y)$ **and** $b = (b_x, b_y)$

$$x(t) = a_x + (b_x - a_x)t$$
$$y(t) = a_y + (b_y - a_y)t$$

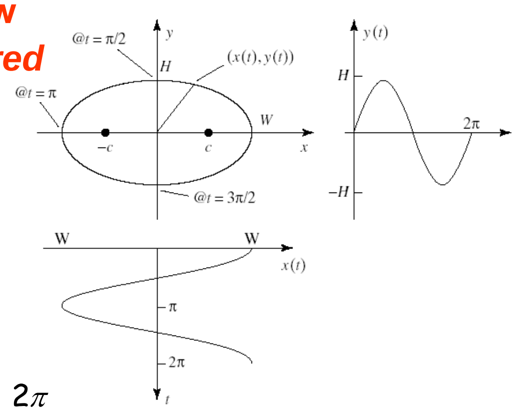
- Sweeps through points on line-segment as t varies from 0 to 1



2D Curves: Parametric Form

An ellipse **of half-width** w **and half-height** h **centered at 0**

$$x(t) = w\cos(t)$$
$$y(t) = h\sin(t)$$



- Sweeps through points on ellipse as t varies from 0 to 2π

Conversion from Parametric to Implicit Form

Eliminate the parameter

- Not always easy to do so

For the ellipse

$$\left(\frac{x}{w}\right)^2 + \left(\frac{y}{h}\right)^2 = 1$$

since

$$\left(\frac{w\cos(t)}{w}\right)^2 + \left(\frac{h\sin(t)}{h}\right)^2 = 1$$

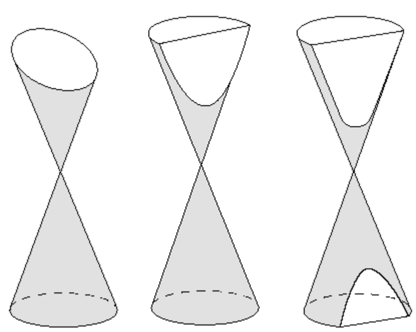
Other Conic Sections

Parabola

- Parametric:

$$x(t) = at^2$$
$$y(t) = 2at$$

- Implicit: $y^2 - 4ax = 0$



Hyperbola

- Parametric: $x(t) = a\sec(t)$

$$y(t) = b\tan(t)$$

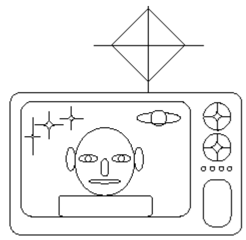
- Implicit: $(x/a)^2 - (y/b)^2 = 1$

Superellipse

Produces nice geometric effects

- Implicit form:

$$\left(\frac{x}{a}\right)^n + \left(\frac{y}{b}\right)^n = 1$$



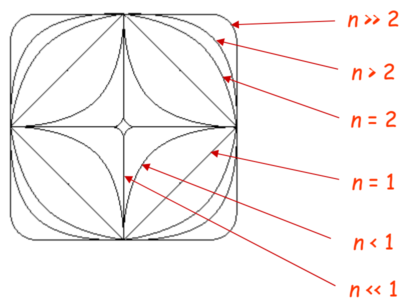
- Parametric form:

$$x(t) = a\cos(t)\left|\cos(t)^{2/n-1}\right|$$

$$y(t) = b\sin(t)\left|\sin(t)^{2/n-1}\right|$$

Supercircle Family

When a = b



Bulge outward for **n > 1**

Bulge inward for **n < 1**

Different Forms of Curve Functions in 3D

Explicit: *$y = f(x)$, $z = g(x)$*

- Cannot get multiple values for single *x*, infinite slopes

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

- Cannot easily compare tangent vectors at joints
- Easy in/out test, normals from gradient

Parametric: *$x = f_x(t)$, $y = f_y(t)$, $z = f_z(t)$*

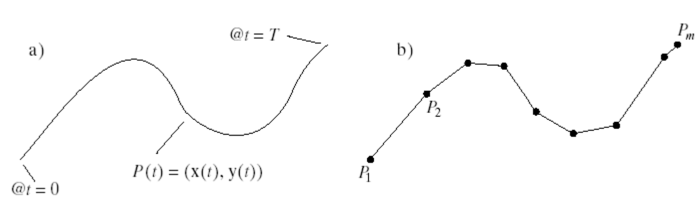
- Overcomes all problems

Drawing Parametric Curves

Compute samples ***of*** $\mathbf{p}(t) = (x(t), y(t))$

$\mathbf{p}_i = \mathbf{p}(t_i) = (x(t_i), y(t_i))$

Approximate the curve by a polyline ***defined through the samples***



Describing Curves by Means of Polynomials

Reminder:

L^{th} degree polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_L t^L$$

$a_0, \ldots, a_L$ are the coefficients

L: is the degree

$L+1$ is the "order" of the polynomial

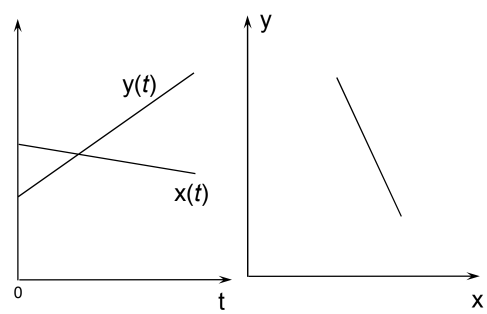
Polynomial Curves of Degree 1

Parametric and implicit forms are linear

$x(t) = at + b$

$y(t) = ct + d$

$F(x,y) = kx + ly + m = 0$



Polynomial Curves of Degree 2

Parametric

$x(t) = at^2+2bt+c$

$y(t) = dt^2+2et+f$

For any choice of constants a,d,c,d,e,f $\rightarrow$ parabola

Implicit

$F(x,y) = Ax^2+2Byx+Cy^2+Dx+Ey+F$

Let $d = AC-B^2$

$d > 0 \rightarrow F(x,y) = 0$ is an ellipse

$d = 0 \rightarrow F(x,y) = 0$ is a parabola

$d < 0 \rightarrow F(x,y) = 0$ is a hyperbola

So

We will use parametric polynomials and constrain them to create desired types of curves

How?

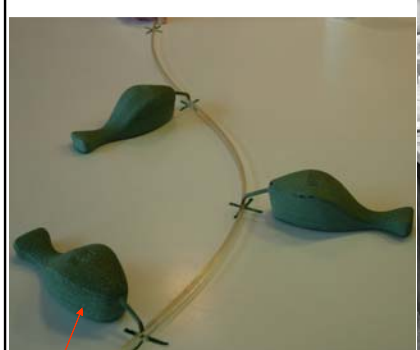
Splines

Piecewise polynomial curves

- Bezier curves
- Hermite curves
- Bernstein polynomials
- Matrix form for splines

Draftsman’s Spline

Boeing Corp.



“Duck”

Interactive Curve Design

Geometric approach

Constraints → Polynomial → Curve

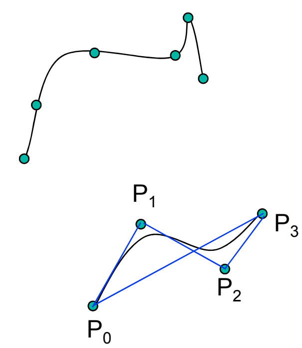
Any t

$P_0,\ldots,P_L \rightarrow$ Curve generation $\rightarrow P(t)$

P_i is a control point

$P_0 \ldots P_L$ is the control polygon

Interpolation vs Approximation

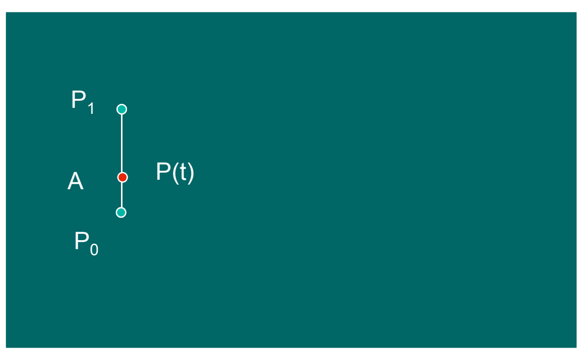


Bezier Curves
The De Casteljau Algorithm

Tweening

Two points

(line)



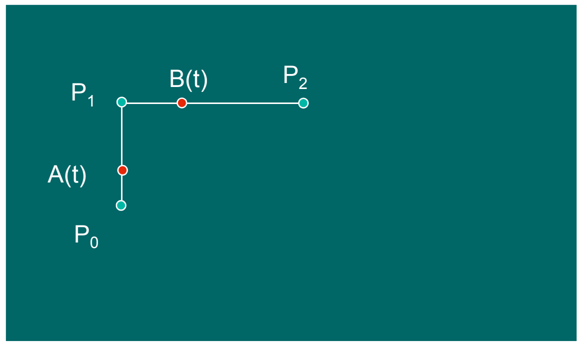
$A(t) = (1\text{-}t)P_0 + tP_1$

$P(t) = A(t)$

Bezier Curves
The De Casteljau Algorithm

Tweening

Three points



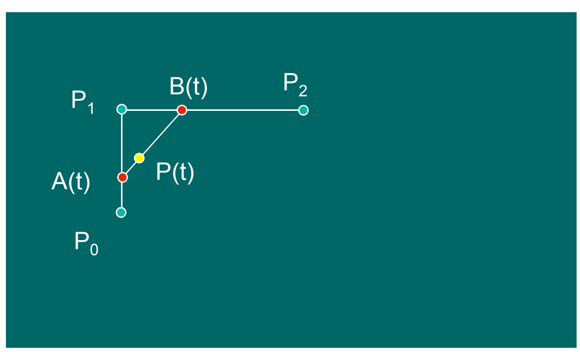
$A(t) = (1\text{-}t)P_0 + tP_1$

$B(t) = (1\text{-}t)P_1 + tP_2$

Bezier Curves
The De Casteljau Algorithm

Tweening

Three points



$A(t) = (1\text{-}t)P_0 + tP_1$

$B(t) = (1\text{-}t)P_1 + tP_2$

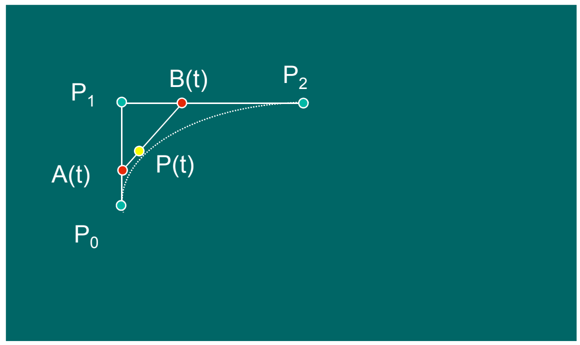
$P(t) = (1\text{-}t)A(t) + tB(t) = (1\text{-}t)^2P_0 + 2t(1\text{-}t)P_1 + t^2P_2$

Bezier Curves
The De Casteljau Algorithm

Tweening

Three points

(parabola)



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

$B(t) = (1-t)P_1 + tP_2$

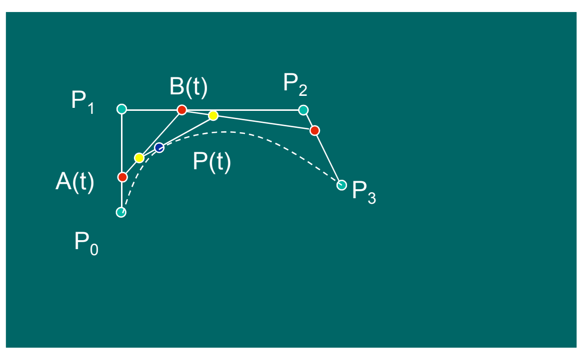
$P(t) = (1-t)A(t) + tB(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2$

Bezier Curves
The De Casteljau Algorithm

Tweening

Four points

(parabola)



$P(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3$

Cubic Bernstein Polynomials

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

$B^3_0 (t) = (1-t)^3$

$B^3_1 (t) = 3(1-t)^2 t$

$B^3_2 (t) = 3(1-t)t^2$

$B^3_3 (t) = t^3$

Expansion of $[(1-t) + t]^3 = (1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3 \rightarrow$

$\Sigma_k B^3_k (t) = 1, \quad k = 0,1,2,3$

An affine combination of points

Bernstein Polynomials of Degree L

L + 1 control points

$$P(t) = \sum_{k=0}^{L} B_k^L(t) P_k \quad \text{where}$$

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

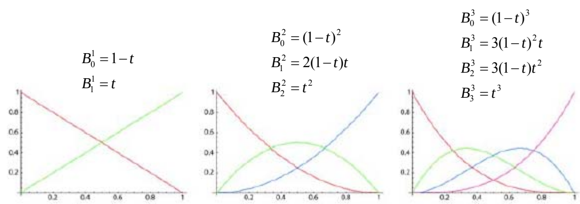
$$\binom{L}{k} = \frac{L!}{k!(L-k)!}, \quad \text{for } L \geq k$$

$$\sum_{k=0}^{L} B_k^L(t) = 1, \quad \text{for all } t$$

Expansion of $[(1-t) + t]^L$

Bernstein Polynomials

Common Bernstein Polynomials

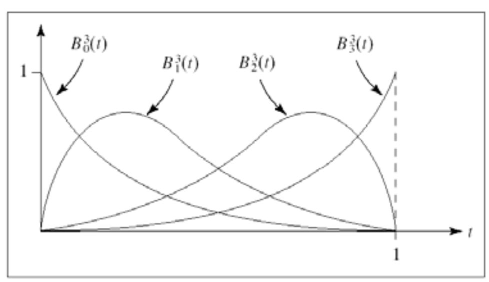


Bernstein Polynomials

Always positive

Zero only at t = 0 or t = 1

Degree 3



Bernstein Polynomials

Bernstein polynomials can also be defined recursively

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + t\, B_{i-1}^{n-1}(u)$$
$$B_0^0(t) = 1$$

Properties of Bezier Curves

- End point interpolation
- Affine Invariance: $T(P(t)) = \sum_{k=0}^{L} B_k^L(t) T(P)_k$
- Invariance under affine transformation of the parameter
- Convex Hull property for t in [0,1] $P = \sum_{k=0}^{L} a_k P_k$, where $\sum_{k=0}^{L} a_k = 1$ and $a_k > 0$
- Linear precision by collapsing convex hull
- Variation diminishing property: No straight line cuts the curve more times than it cuts the control polygon

Derivatives of Bezier Curves

It can be shown that for $P(t)=\sum_{k=0}^{L} B_k^L(t)P_k$

Velocity is also a Bezier curve of lower degree

$$P'(t)=L\sum_{k=0}^{L-1} B_k^{L-1}(t)\Delta P_k \text{ where } \Delta P_k = P_{k+1}-P_k$$

Acceleration:

$$P''(t)=L(L-1)\sum_{k=0}^{L-2} B_k^{L-2}(t)\Delta^2 P_k \text{ where } \Delta^2 P_k = \Delta P_{k+1}-\Delta P_k$$

Which Degree is Best?

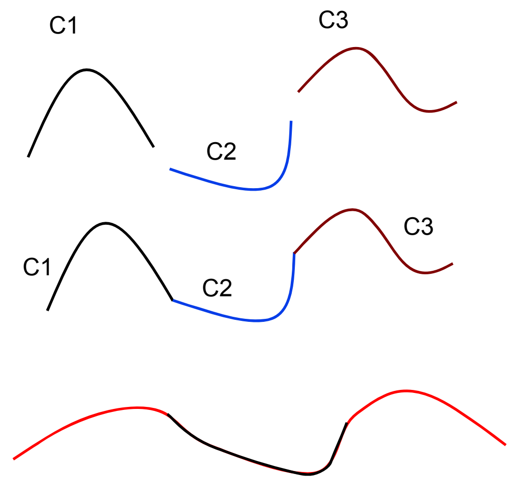
Cubic curves

- Lower than cubic order offers not enough flexibility
- Higher order has too many wiggles and is computationally expensive
- Cubic curves are lowest degree polynomial curves that are not necessarily planar in 3D

More complex curves?

- Piecewise cubics

Piecewise Cubic Curves



Classifying the Continuity of Curves

***Parametric continuity* – C^k**

- Each coordinate function of the curve is differentiable *k* times
- And each is continuous through the k^{th} derivative

***Geometric continuity* – G^k**

- The curve itself is continuous up to order k independent of the parameterization
  - G^0 – two segments meet at the same point
  - G^1 – with the same tangent
  - G^2 – and the same curvature

These two kinds of continuity are* not *always equivalent

Cubic Curves in 3D

Consider coordinate functions that are cubic polynomials

$$x(u) = a_3u^3 + a_2u^2 + a_1u + a_0$$
$$y(u) = b_3u^3 + b_2u^2 + b_1u + b_0 \quad \text{where} \quad 0 \le u \le 1$$
$$z(u) = c_3u^3 + c_2u^2 + c_1u + c_0$$

Each is a linear combination of monomial terms

$$x(u) = \sum_{i=0}^{3} a_i u^i \qquad y(u) = \sum_{i=0}^{3} b_i u^i \qquad z(u) = \sum_{i=0}^{3} c_i u^i$$

For convenience, we can rewrite this in vector form

$$\mathbf{p}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \sum_{i=0}^{3} \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} u^i = \sum_{i=0}^{3} \mathbf{a}_i u^i \quad \text{where} \quad \mathbf{a}_i = \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix}$$

…and in an even more condensed matrix form

$$\mathbf{p}(u) = \mathbf{A}\mathbf{u} \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \end{bmatrix}$$

Rewriting with Geometric Constraints

A cubic is defined by 4 constraints

- We want to rewrite spline formulas in terms of these constraints, **not** the coefficients of the monomial terms

$$\mathbf{p}(u) = \mathbf{A}\mathbf{u} = \mathbf{G}\mathbf{M}\mathbf{u}$$

where

$$\mathbf{M} = \underbrace{\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}}_{\text{Basis Matrix}} \quad \text{and} \quad \mathbf{G} = \underbrace{\begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \mathbf{g}_4 \end{bmatrix}}_{\text{Geometry Matrix}}$$

Hermite Curves

Specify 4 geometric constraints

- Endpoints of the curve segment
- Tangent vectors at the endpoints

$$G = [\mathbf{p}_0 \quad \mathbf{p}_3 \quad \mathbf{r}_0 \quad \mathbf{r}_3]$$

where

$$\mathbf{p}_0 = \mathbf{p}(0)$$
$$\mathbf{p}_3 = \mathbf{p}(1)$$
$$\mathbf{r}_0 = \mathbf{p}'(0)$$
$$\mathbf{r}_3 = \mathbf{p}'(1)$$

It's easy to paste Hermite segments together

- Adjacent segments share endpoints and tangents
- Guarantees tangents are continuous — C^1 continuity

Deriving the Hermite Basis Matrix

These are the constraints we want

$$\mathbf{p}_0 = \mathbf{p}(0) = \mathbf{a}_0$$
$$\mathbf{p}_3 = \mathbf{p}(1) = \mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3$$
$$\mathbf{r}_0 = \mathbf{p}'(0) = \mathbf{a}_1$$
$$\mathbf{r}_3 = \mathbf{p}'(1) = \mathbf{a}_1 + 2\mathbf{a}_2 + 3\mathbf{a}_3$$

$$\mathbf{p}(u) = \sum_{i=0}^{3} \mathbf{a}_i u^i = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3$$

$$\mathbf{p}'(u) = \mathbf{a}_1 + 2\mathbf{a}_2 u + 3\mathbf{a}_3 u^2$$

We can rewrite these constraints as

$$G = A \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix}$$

Hence:

$$A = G \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix}^{-1} = G \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} = GM$$

Equation for the Hermite Curve

The curve is:

$$\mathbf{p}(u) = \mathbf{GMu} = [\mathbf{p}_0 \quad \mathbf{p}_3 \quad \mathbf{r}_0 \quad \mathbf{r}_3] \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$$

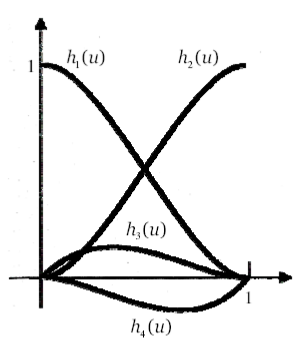
We can also regard it as a weighted sum of the constraints:

$$\begin{aligned}\mathbf{p}(u) &= (1-3u^2+2u^3)\mathbf{p}_0 + (3u^2-2u^3)\mathbf{p}_3 + (u-2u^2+u^3)\mathbf{r}_0 + (-u^2+u^3)\mathbf{r}_3 \\ &= h_1(u)\mathbf{p}_0 + h_2(u)\mathbf{p}_3 + h_3(u)\mathbf{r}_0 + h_4(u)\mathbf{r}_3\end{aligned}$$

- Each constraint is weighted by a ***blending function*** $h_i(u)$
- The coefficients of these blending functions are the rows of the basis matrix **M**

Hermite Blending Polynomials

$$\begin{aligned}h_1(u) &= 1 - 3u^2 + 2u^3 \\ h_2(u) &= 3u^2 - 2u^3 \\ h_3(u) &= u - 2u^2 + u^3 \\ h_4(u) &= -u^2 + u^3\end{aligned}$$



Matrix Form for Cubic Bézier Curve

$$\mathbf{p}(u) = (1-u)^3\mathbf{p}_0 + 3(1-u)^2u\mathbf{p}_1 + 3(1-u)u^2\mathbf{p}_2 + u^3\mathbf{p}_3$$
$$= (1-3u+3u^2-u^3)\mathbf{p}_0 + (3u-6u^2+3u^3)\mathbf{p}_1 + (3u^2-3u^3)\mathbf{p}_2 + u^3\mathbf{p}_3$$
$$= h_1(u)\mathbf{p}_0 + h_2(u)\mathbf{p}_1 + h_3(u)\mathbf{p}_2 + h_4(u)\mathbf{p}_3$$

Therefore:

$$\mathbf{p}(u) = \mathbf{GMu} = \begin{bmatrix}\mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\end{bmatrix}\begin{bmatrix}1 & -3 & 3 & -1\\ 0 & 3 & -6 & 3\\ 0 & 0 & 3 & -3\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\ u\\ u^2\\ u^3\end{bmatrix}$$

Bézier Continuity

Suppose that we are given two cubic Bézier control polygons

$$\mathbf{p}_0 \quad \mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3$$
$$\mathbf{q}_0 \quad \mathbf{q}_1 \quad \mathbf{q}_2 \quad \mathbf{q}_3$$

where the two curves p(u) and q(u) should join consecutively

What constraints on these points are necessary to guarantee C^1 continuity between them?

Bézier Tangents

For a Bézier curve

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(u) \qquad 0 \le u \le 1$$

The derivatives at the endpoints are

$$\mathbf{p}'(0) = n(\mathbf{p}_1 - \mathbf{p}_0)$$
$$\mathbf{p}'(1) = n(\mathbf{p}_n - \mathbf{p}_{n-1})$$

So in the cubic case we have

$$\mathbf{p}'(0) = 3(\mathbf{p}_1 - \mathbf{p}_0)$$
$$\mathbf{p}'(1) = 3(\mathbf{p}_3 - \mathbf{p}_2)$$

Bézier to Hermite Conversion

This gives us a direct connection to Hermite splines

$$\text{Hermite}\left\{\begin{array}{l} \mathbf{p}_0 = \mathbf{p}_0 \\ \mathbf{p}_3 = \mathbf{p}_3 \\ \mathbf{r}_0 = 3(\mathbf{p}_1 - \mathbf{p}_0) \\ \mathbf{r}_3 = 3(\mathbf{p}_3 - \mathbf{p}_2) \end{array}\right\}\text{Bézier}$$

Which we can write in matrix form

$$[\mathbf{p}_0 \quad \mathbf{p}_3 \quad \mathbf{r}_0 \quad \mathbf{r}_3] = [\mathbf{p}_0 \quad \mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3]\begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix}$$

Converting Between Cubic Spline Types

We saw the specific example of Bézier-Hermite conversion

Suppose we want to convert between two arbitrary splines

$$\mathbf{G}_a \mathbf{M}_a \mathbf{u} = \mathbf{G}_b \mathbf{M}_b \mathbf{u}$$

Given geometry matrix G_a, find the equivalent G_b for the other spline

$$\mathbf{G}_b = \mathbf{G}_a \mathbf{M}_a \mathbf{M}_b^{-1}$$

Catmull-Rom Splines

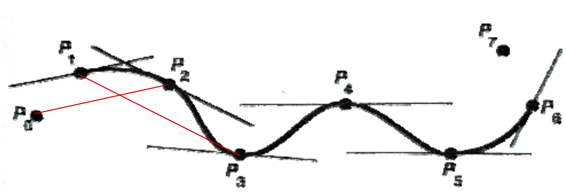
Given a set of points in space, suppose we want a spline that

- Interpolates the points (rules out Bézier)
- With C^1 continuity (Hermite: Lots of tweaking)

This is a common situation in animation

We start with the given set of points $p_0,\ldots,p_n$

Define tangents $r_i = s\,(p_{i+1} - p_{i-1})$



Catmull-Rom Splines

Typically we choose s = ½ *and we can derive a spline equation*

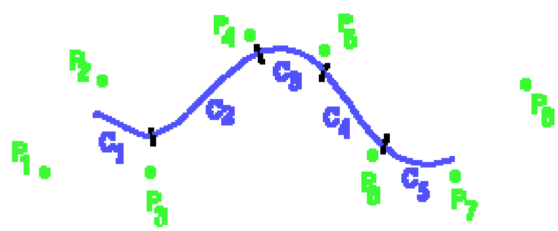
$$\mathbf{p}(u) = \frac{1}{2}\begin{bmatrix}\mathbf{p}_{i-3} & \mathbf{p}_{i-2} & \mathbf{p}_{i-1} & \mathbf{p}_i\end{bmatrix}\begin{bmatrix}0 & -1 & 2 & -1\\ 2 & 0 & -5 & 3\\ 0 & 1 & 4 & -3\\ 0 & 0 & -1 & 1\end{bmatrix}\begin{bmatrix}1\\ u\\ u^2\\ u^3\end{bmatrix}$$

More generally we can use a tension *parameter* s

$$\mathbf{p}(u) = \begin{bmatrix}\mathbf{p}_{i-3} & \mathbf{p}_{i-2} & \mathbf{p}_{i-1} & \mathbf{p}_i\end{bmatrix}\begin{bmatrix}0 & -s & 2s & -s\\ 1 & 0 & s-3 & 2-s\\ 0 & s & 3-2s & s-2\\ 0 & 0 & -s & s\end{bmatrix}\begin{bmatrix}1\\ u\\ u^2\\ u^3\end{bmatrix}$$

B-Splines

Like Catmull-Rom splines, start with sequence of points $\mathbf{p}_0,\ldots,\mathbf{p}_n$



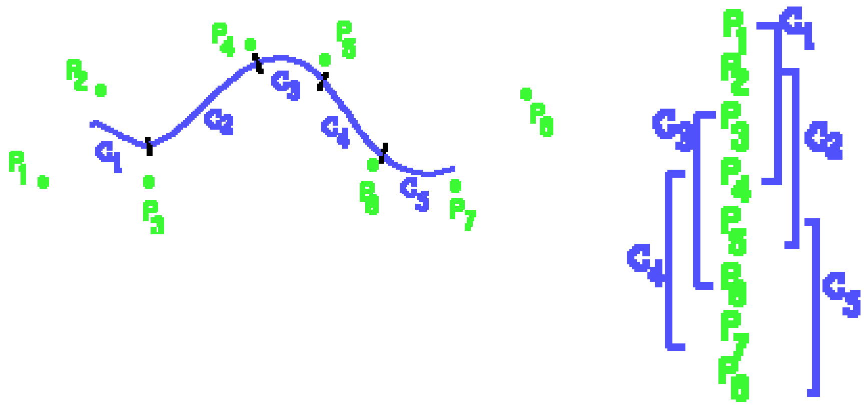
Curves no longer interpolate control points

- Points where segments actually meet are called ***knots***
- For Hermite and others, the knots were always at control points

Lack of interpolation isn't a big problem for interactive design

- But it's hard to predict the position of the curve for any parameter value *u* just based on the coordinates of the control points

B-Splines



B-Spline Basis Functions

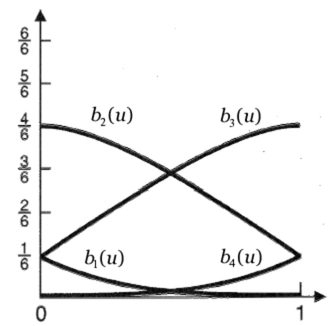
$$\mathbf{p}(u) = b_1(u)\mathbf{p}_{i-3} + b_2(u)\mathbf{p}_{i-2} + b_3(u)\mathbf{p}_{i-1} + b_4(u)\mathbf{p}_i$$

$$b_1(u) = \frac{1}{6}(1-u)^3$$

$$b_2(u) = \frac{1}{6}(3u^3 - 6u^2 + 4)$$

$$b_3(u) = \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1)$$

$$b_4(u) = \frac{1}{6}u^3$$



Non-negative functions

- Implies convex hull property

Matrix Form for B-Splines

$$\mathbf{p}(u) = \frac{1}{6}\begin{bmatrix}\mathbf{p}_{i-3} & \mathbf{p}_{i-2} & \mathbf{p}_{i-1} & \mathbf{p}_i\end{bmatrix}\begin{bmatrix}1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 \\ u \\ u^2 \\ u^3\end{bmatrix}$$

B-Spline Properties

C^2 continuous!

Convex hull property

NO invariace under perspective projection

NURBS: Nonuniform Rational B-splines

Invariance under perspective projection

Can create exact conic sections

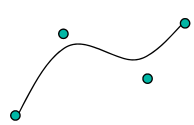
x(u) = X(u) / W(u)

y(u) = Y(u) / W(u)

z(u) = Z(u) / W(u)

In General

$P_0, \ldots, P_L \rightarrow$ Curve generation $\rightarrow P(u)$



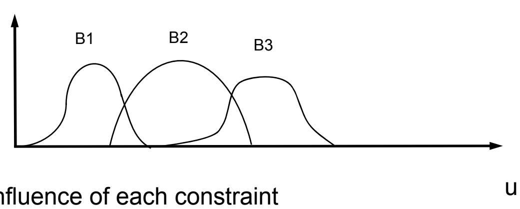
$$P(u) = \sum_{k=0}^{L} B_k(u) P_k$$

where

$P_k, \quad k = 1, \ldots, L$: Constraints

$B_k(u)$: Blending functions

$u \in [a, b]$



The Blending functions weight the influence of each constraint (e.g., control point) on the curve created

Wish List for Blending Functions

- They should have sufficient smoothness
- They should be easy to compute and stable
- They should sum to unity for every u in [a,b]
- They should “have support” over a portion of [a,b]
- They could interpolate certain control points