CMSC427
Parametric curves:
Hermite, Catmull-Rom, Bezier
Modeling

- Creating 3D objects
- How to construct complicated surfaces?
- Goal
  - Specify objects with few control points
  - Resulting object should be visually pleasing (smooth)
- Start with curves, then generalize to surfaces
Usefulness of curves

- Surface of revolution
Usefulness of curves

- Extruded/swept surfaces
Usefulness of curves

• Animation
  • Provide a “track” for objects
  • Use as camera path
Usefulness of curves

• Generalize to surface patches using “grids of curves”, next class
How to represent curves

• Specify every point along curve?
  • Hard to get precise, smooth results
  • Too much data, too hard to work with

• Idea: specify curves using small numbers of control points

• Mathematics: use polynomials to represent curves
Interpolating polynomial curves

http://en.wikipedia.org/wiki/Polynomial_interpolation

• Curve goes through all control points
• Seems most intuitive
• Surprisingly, not usually the best choice
  • Hard to predict behavior
  • Overshoots, wiggles
  • Hard to get “nice-looking” curves
Approximating polynomial curves

- Curve is “influenced” by control points

- Various types & techniques based on polynomial functions
  - Bézier curves, B-splines, NURBS

- Focus on Bézier curves
Mathematical definition

- A vector valued function of one variable $\mathbf{x}(t)$
  - Given $t$, compute a 3D point $\mathbf{x}=(x,y,z)$
  - May interpret as three functions $x(t)$, $y(t)$, $z(t)$
  - “Moving a point along the curve”
Tangent vector

- **Derivative**
  \[ \mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = (x'(t), y'(t), z'(t)) \]
- **A vector that** points in the direction of movement.
- **Length of** \( \mathbf{x}'(t) \) **corresponds to speed**
Piecewise polynomial curves

- Model complex shapes by sequence
- Use polyline to store control points
Continuity

• How piecewise curves join
• $C^k$ continuity – $k$th derivatives match
• $G^k$ continuity – $k$th derivatives are proportional
Hermite curves

- Cubic curve (here 2D)
  \[ x(t) = at^3 + bt^2 + ct + d \]
  \[ y(t) = et^3 + ft^2 + gt + h \]
- Interpolates end points P0 and P1
- Matches tangent at endpoints T0 and T1
  - (also dP0 and dP1 in these notes).

\[ P0 = (x_0, y_0) \]
\[ T0 = <dx_0, dy_0> \]
\[ P1 = (x_1, y_1) \]
\[ T1 = <dx_1, dy_1> \]
Computing coefficients $a$, $b$, $c$ and $d$

- Derivative of $x(t)$

$$x'(t) = 3at^2 + 2bt + c$$

- Set $t = 0$ and $1$ for endpoints

- Four constraints

$$x(0) = d$$
$$x'(0) = c$$
$$x(1) = a + b + c + d$$
$$x'(1) = 3a + 2b + c$$
Solve for a, b, c and d

\[ d = x_0 \]

\[ c = dx_0 \]

\[ b = -3x_0 + 3x_1 - 2dx_0 - dx_1 \]

\[ a = 2x_0 - 2x_1 + dx_0 + dx_1 \]
Matrix version

- **Constraints**
  
  \[
  \begin{align*}
  x(0) &= d \\
  x'(0) &= c \\
  x(1) &= a + b + c + d \\
  x'(1) &= 3a + 2b + c
  \end{align*}
  \]

- **Give**

  \[
  \begin{bmatrix}
  0 & 0 & 0 & 1 \\
  1 & 1 & 1 & 1 \\
  0 & 0 & 1 & 0 \\
  3 & 2 & 1 & 0
  \end{bmatrix}
  \begin{bmatrix}
  a \\
  b \\
  c \\
  d
  \end{bmatrix}
  =
  \begin{bmatrix}
  x_0 \\
  x_1 \\
  dx_0 \\
  dx_1
  \end{bmatrix}
  \]
• Since we have $MA = G$
• We can solve with $A = M^{-1}G$
• And get Hermite basis matrix $M^{-1}$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ dx0 \\ dx1 \end{bmatrix}$$
To include $x$, $y$ and $z$, rewrite with vectors $P0$, $P1$ and tangents $T0$ and $T1$

\[
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix} = \begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}\begin{bmatrix}
P0 \\
P1 \\
T0 \\
T1
\end{bmatrix}
\]

Coefficients $a$, $b$, $c$ and $d$ are now vectors
- Rewrite polynomial as dot product

\[ P(t) = [t^3 \ t^2 \ t \ 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \]

\[ = [t^3 \ t^2 \ t \ 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P0 \\ P1 \\ T0 \\ T1 \end{bmatrix} \]
Blending functions

• Instead of polynomial in \( t \), look at curve as weighted sum of \( P_0, P_1, T_0 \) and \( T_1 \)

\[
x(t) = (2x_0 - 2x_1 + dx_0 + dx_1)t^3
\]

\[
+ (-3x_0 + 3x_1 - 2dx_0 - dx_1)t^2
\]

\[
+ (dx_0)t
\]

\[
+ x_0
\]
• Instead of polynomial in $t$, look at curve as weighted sum of $P_0$, $P_1$, $T_0$ and $T_1$

• $x(t) =$
• $(2t^3 - 3t^2 + 1)x_0$
• $+(-2t^3 + 3t^2)x_1$
• $+(t^3 - 2t^2 + t)dx_0$
• $+(t^3 - t^2)dx_1$
Blending functions

\[ h_{00}(t) = (2t^3 - 3t^2 + 1) \]
\[ h_{01}(t) = (-2t^3 + 3t^2) \]
\[ h_{10}(t) = (t^3 - 2t^2 + t) \]
\[ h_{11}(t) = (t^3 - t^2) \]
Computing Hermite tangents

- Have $P(-1)$, $P0$, $P1$ and $P2$ as input
- Compute tangent with $H$ matrix

\[
\begin{bmatrix}
  x_0 \\
  x_1 \\
  dx_0 \\
  dx_1 \\
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  -1 & 1 & 0 & 0 & 0 \\
  0 & 0 & -1 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_{-1} \\
  x_2 \\
\end{bmatrix}
\]
Combine with Hermite basis

- Unify notation

\[
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix} = \begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & -1 & 0 \\
0 & -1 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_0 \\
x_1 \\
x_{-1} \\
x_2
\end{bmatrix}
\]

- Final matrix

\[
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix} = \begin{bmatrix}
3 & -3 & -1 & 1 \\
-5 & 4 & 2 & -1 \\
1 & 0 & -1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
x_0 \\
x_1 \\
x_{-1} \\
x_2
\end{bmatrix}
\]
Catmull-Rom curves

- Hermite – problem with C1 continuity
Catmull-Rom curves

• Catmull-Rom – make tangent symmetric
• Define by two adjacent points
• Here $T_3 = P_4 - P_2$
Catmull-Rom curves

- Need to change H matrix
- $\frac{1}{2}$ traditional for C-R curves

\[
\begin{bmatrix}
  x_0 \\
  x_1 \\
  dx_0' \\
  dx_1'
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  1/2 & 0 & -1/2 & 0 \\
  -1/2 & 0 & 0 & 1/2
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_{-1} \\
  x_2
\end{bmatrix}
\]
Catmull-Rom curves

- Which gives

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 \\ -1/2 & 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_{-1} \\ x_2 \end{bmatrix}$$

- Or

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & -0.5 & 0.5 \\ -3.5 & 3 & 1 & -0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_{-1} \\ x_2 \end{bmatrix}$$
Bézier curves

http://en.wikipedia.org/wiki/B%C3%A9zier_curve

• A particularly intuitive way to define control points for polynomial curves

• Developed for CAD (computer aided design) and manufacturing
  • Before games, before movies, CAD was the big application for CG

• Pierre Bézier (1962), design of auto bodies for Peugeot, http://en.wikipedia.org/wiki/Pierre_B%C3%A9zier

• Paul de Casteljau (1959), for Citroen
Bézier curves

• Higher order extension of linear interpolation
• Control points $p_0, p_1, ...$
Bézier curves

• Intuitive control over curve given control points
  • Endpoints are interpolated, intermediate points are approximated
  • Convex Hull property
  • Variation-diminishing property
Cubic Bézier curve

• Cubic polynomials, most common case
• Defined by 4 control points
• Two interpolated **endpoints**
• Two **midpoints** control the tangent at the endpoints
Bézier Curve formulation

• Three alternatives, analogous to linear case
  1. Weighted average of control points
  2. Cubic polynomial function of $t$
  3. Matrix form

• Algorithmic construction
  • de Casteljau algorithm
de Casteljau Algorithm


• A recursive series of linear interpolations
  • Works for any order, not only cubic

• Not terribly efficient to evaluate
  • Other forms more commonly used

• Why study it?
  • Intuition about the geometry
  • Useful for subdivision (later today)
• Given the control points
• A value of $t$
• Here $t \approx 0.25$
de Casteljau Algorithm

\[ q_0(t) = \text{Lerp}(t, p_0, p_1) \]
\[ q_1(t) = \text{Lerp}(t, p_1, p_2) \]
\[ q_2(t) = \text{Lerp}(t, p_2, p_3) \]
de Casteljau Algorithm

\[ \mathbf{r}_0(t) = \text{Lerp}(t, \mathbf{q}_0(t), \mathbf{q}_1(t)) \]
\[ \mathbf{r}_1(t) = \text{Lerp}(t, \mathbf{q}_1(t), \mathbf{q}_2(t)) \]
\[ \mathbf{x}(t) = \text{Lerp}(t, \mathbf{r}_0(t), \mathbf{r}_1(t)) \]
de Casteljau algorithm

• Applets
  • http://www2.mat.dtu.dk/people/J.Gravesen/cagd/decast.html
  • http://www.caffeineowl.com/graphics/2d/vectorial/bezierintro.html
de Casteljau Algorithm

Recursive linear interpolation

\[ r_0 = L, r_1 = L, q_0 = L, q_1 = L, q_2 = L \]

\[ p_0, p_1, p_2, p_3, p_4 \]
Recursive linear interpolation

\[
\begin{align*}
q_0 &= \text{Lerp}(t, p_0, p_1) \\
q_1 &= \text{Lerp}(t, p_1, p_2) \\
q_2 &= \text{Lerp}(t, p_2, p_3)
\end{align*}
\]
Recursive linear interpolation

\[
\begin{align*}
    r_0 &= \text{Lerp}(t, q_0, q_1) \\
    r_1 &= \text{Lerp}(t, q_1, q_2)
\end{align*}
\]

\[
\begin{align*}
    q_0 &= \text{Lerp}(t, p_0, p_1) \\
    q_1 &= \text{Lerp}(t, p_1, p_2) \\
    q_2 &= \text{Lerp}(t, p_2, p_3)
\end{align*}
\]
Recursive linear interpolation

\[
x = \text{Lerp}(t, r_0, r_1)
\]

\[
r_0 = \text{Lerp}(t, q_0, q_1)
\]

\[
r_1 = \text{Lerp}(t, q_1, q_2)
\]

\[
q_0 = \text{Lerp}(t, p_0, p_1)
\]

\[
q_1 = \text{Lerp}(t, p_1, p_2)
\]

\[
q_2 = \text{Lerp}(t, p_2, p_3)
\]

Diagram:

```
   x
 / \
 r_0 /  \
 | /   \
 q_0 | /   \
   r_1 | /   \
 | /   q_1
 p_2 / \
   r_1   q_2
 p_3 /  \
 |  /   \
 p_2 |  /   \
 |  |  /   \
 p_4
```
Expand the LERPs

\[
\begin{align*}
q_0(t) &= Lerp(t, p_0, p_1) = (1 - t)p_0 + tp_1 \\
q_1(t) &= Lerp(t, p_1, p_2) = (1 - t)p_1 + tp_2 \\
q_2(t) &= Lerp(t, p_2, p_3) = (1 - t)p_2 + tp_3
\end{align*}
\]
Expand the LERPs

\[ q_0(t) = \text{Lerp}(t,p_0,p_1) = (1-t)p_0 + tp_1 \]
\[ q_1(t) = \text{Lerp}(t,p_1,p_2) = (1-t)p_1 + tp_2 \]
\[ q_2(t) = \text{Lerp}(t,p_2,p_3) = (1-t)p_2 + tp_3 \]

\[ r_0(t) = \text{Lerp}(t,q_0(t),q_1(t)) \]
\[ r_1(t) = \text{Lerp}(t,q_1(t),q_2(t)) \]
Expand the LERPs

\[ q_0(t) = Lerp(t, p_0, p_1) = (1 - t)p_0 + tp_1 \]
\[ q_1(t) = Lerp(t, p_1, p_2) = (1 - t)p_1 + tp_2 \]
\[ q_2(t) = Lerp(t, p_2, p_3) = (1 - t)p_2 + tp_3 \]

\[ r_0(t) = Lerp(t, q_0(t), q_1(t)) = (1 - t)((1 - t)p_0 + tp_1) + t((1 - t)p_1 + tp_2) \]
\[ r_1(t) = Lerp(t, q_1(t), q_2(t)) = (1 - t)((1 - t)p_1 + tp_2) + t((1 - t)p_2 + tp_3) \]
Expand the LERPs

\[ q_0(t) = Lerp(t, p_0, p_1) = (1-t)p_0 + tp_1 \]
\[ q_1(t) = Lerp(t, p_1, p_2) = (1-t)p_1 + tp_2 \]
\[ q_2(t) = Lerp(t, p_2, p_3) = (1-t)p_2 + tp_3 \]

\[ r_0(t) = Lerp(t, q_0(t), q_1(t)) = (1-t)((1-t)p_0 + tp_1) + t((1-t)p_1 + tp_2) \]
\[ r_1(t) = Lerp(t, q_1(t), q_2(t)) = (1-t)((1-t)p_1 + tp_2) + t((1-t)p_2 + tp_3) \]

\[ x(t) = Lerp(t, r_0(t), r_1(t)) \]
Expand the LERPs

\[ q_0(t) = Lerp(t, p_0, p_1) = (1-t)p_0 + tp_1 \]
\[ q_1(t) = Lerp(t, p_1, p_2) = (1-t)p_1 + tp_2 \]
\[ q_2(t) = Lerp(t, p_2, p_3) = (1-t)p_2 + tp_3 \]

\[ r_0(t) = Lerp(t, q_0(t), q_1(t)) = (1-t)((1-t)p_0 + tp_1) + t((1-t)p_1 + tp_2) \]
\[ r_1(t) = Lerp(t, q_1(t), q_2(t)) = (1-t)((1-t)p_1 + tp_2) + t((1-t)p_2 + tp_3) \]

\[ x(t) = Lerp(t, r_0(t), r_1(t)) \]
\[ = (1-t)(((1-t)((1-t)p_0 + tp_1) + t((1-t)p_1 + tp_2)) \]
\[ + t((1-t)((1-t)p_1 + tp_2) + t((1-t)p_2 + tp_3))) \]
Weighted average of control points

- Regroup

\[
x(t) = (1 - t)((1 - t)(1 - t)p_0 + tp_1) + t((1 - t)p_1 + tp_2))
+ t(1 - t)((1 - t)p_1 + tp_2) + t((1 - t)p_2 + tp_3))
\]
Weighted average of control points

- Regroup

\[ x(t) = (1 - t)((1 - t)((1 - t)p_0 + tp_1) + t((1 - t)p_1 + tp_2) + t((1 - t)p_1 + tp_2) + t((1 - t)p_2 + tp_3)) \]

\[ x(t) = (1 - t)^3 p_0 + 3(1 - t)^2 tp_1 + 3(1 - t)t^2 p_2 + t^3 p_3 \]
Weighted average of control points

- Regroup

\[
x(t) = (1 - t) \left( (1 - t) \left( (1 - t)p_0 + tp_1 \right) + t \left( (1 - t)p_1 + tp_2 \right) \right)
\]

\[
+ t \left( (1 - t) \left( (1 - t)p_1 + tp_2 \right) + t \left( (1 - t)p_2 + tp_3 \right) \right)
\]

\[
x(t) = (1 - t)^3 p_0 + 3(1 - t)^2 tp_1 + 3(1 - t)t^2 p_2 + t^3 p_3
\]

\[
x(t) = \underbrace{-t^3 + 3t^2 - 3t + 1}_{B_0(t)} p_0 + \underbrace{3t^3 - 6t^2 + 3t}_{B_1(t)} p_1
\]

\[
+ \underbrace{-3t^3 + 3t^2}_{B_2(t)} p_2 + \underbrace{t^3}_{B_3(t)} p_3
\]

Bernstein polynomials
Cubic Bernstein polynomials

\[ x(t) = B_0(t)p_0 + B_1(t)p_1 + B_2(t)p_2 + B_3(t)p_3 \]

The cubic *Bernstein polynomials*:

\[ B_0(t) = -t^3 + 3t^2 - 3t + 1 \]
\[ B_1(t) = 3t^3 - 6t^2 + 3t \]
\[ B_2(t) = -3t^3 + 3t^2 \]
\[ B_3(t) = t^3 \]

\[ \sum B_i(t) = 1 \]

- **Partition of unity**, at each \( t \) always add to 1
- **Endpoint interpolation**, \( B_0 \) and \( B_3 \) go to 1
General Bernstein polynomials

\[ B_0^1(t) = -t + 1 \]
\[ B_1^1(t) = t \]
General Bernstein polynomials

\[ B_0^1(t) = -t + 1 \]
\[ B_1^1(t) = t \]
\[ B_0^2(t) = t^2 - 2t + 1 \]
\[ B_1^2(t) = -2t^2 + 2t \]
\[ B_2^2(t) = t^2 \]
General Bernstein polynomials

\[
\begin{align*}
B_0^1(t) &= -t + 1 \\
B_1^1(t) &= t \\
B_0^2(t) &= t^2 - 2t + 1 \\
B_1^2(t) &= -2t^2 + 2t \\
B_2^2(t) &= t^2 \\
B_0^3(t) &= -t^3 + 3t^2 - 3t + 1 \\
B_1^3(t) &= 3t^3 - 6t^2 + 3t \\
B_2^3(t) &= -3t^3 + 3t^2 \\
B_3^3(t) &= t^3
\end{align*}
\]
General Bernstein polynomials

\[ B_0^1(t) = -t + 1 \]
\[ B_1^1(t) = t \]

\[ B_0^2(t) = t^2 - 2t + 1 \]
\[ B_1^2(t) = -2t^2 + 2t \]
\[ B_2^2(t) = t^2 \]

\[ B_0^3(t) = -t^3 + 3t^2 - 3t + 1 \]
\[ B_1^3(t) = 3t^3 - 6t^2 + 3t \]
\[ B_2^3(t) = -3t^3 + 3t^2 \]
\[ B_3^3(t) = t^3 \]

Order \( n \):

\[ B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \]

\[ \binom{n}{i} = \frac{n!}{i!(n-i)!} \]

\[ \sum B_i^n(t) = 1 \]

Partition of unity, endpoint interpolation
General Bézier curves

- $n$th-order Bernstein polynomials form $n$th-order Bézier curves
- Bézier curves are weighted sum of control points using $n$th-order Bernstein polynomials

Bernstein polynomials of order $n$:

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i$$

Bézier curve of order $n$:

$$x(t) = \sum_{i=0}^{n} B_i^n(t)p_i$$
Bézier curve properties

- Convex hull property
- Variation diminishing property
- Affine invariance
Convex hull, convex combination

- **Convex hull** of a set of points
  - Smallest polyhedral volume such that
    (i) all points are in it
    (ii) line connecting any two points in the volume lies completely inside it (or on its boundary)

- **Convex combination** of the points
  - Weighted average of the points, where weights all between 0 and 1, sum up to 1

- Any convex combination always lies within the convex hull
Convex hull property

• Bézier curve is a **convex combination** of the control points
  • Bernstein polynomials add to 1 at each value of $t$
• Curve is always **inside the convex hull** of control points
• Makes curve predictable
• Allows efficient culling, intersection testing, adaptive tessellation
Variation diminishing property

• If the curve is in a plane, this means no straight line intersects a Bézier curve more times than it intersects the curve's control polyline
• “Curve is not more wiggly than control polyline”

Yellow line: 7 intersections with control polyline
3 intersections with curve
Affine invariance

• Two ways to transform Bézier curves
  1. Transform the control points, then compute resulting point on curve
  2. Compute point on curve, then transform it

• Either way, get the same transform point!
  • Curve is defined via affine combination of points (convex combination is special case of an affine combination)
  • Invariant under affine transformations
  • Convex hull property always remains
Start with Bernstein form:

\[
x(t) = (-t^3 + 3t^2 - 3t + 1)p_0 + (3t^3 - 6t^2 + 3t)p_1 + (-3t^3 + 3t^2)p_2 + (t^3)p_3
\]
Cubic polynomial form

Start with Bernstein form:

\[ x(t) = \left( -t^3 + 3t^2 - 3t + 1 \right) p_0 + \left( 3t^3 - 6t^2 + 3t \right) p_1 + \left( -3t^3 + 3t^2 \right) p_2 + \left( t^3 \right) p_3 \]

Regroup into coefficients of \( t \):

\[ x(t) = \left( -p_0 + 3p_1 - 3p_2 + p_3 \right) t^3 + \left( 3p_0 - 6p_1 + 3p_2 \right) t^2 + \left( -3p_0 + 3p_1 \right) t + \left( p_0 \right) t^1 \]
Cubic polynomial form

Start with Bernstein form:

\[ x(t) = (-t^3 + 3t^2 - 3t + 1)p_0 + (3t^3 - 6t^2 + 3t)p_1 + (-3t^3 + 3t^2)p_2 + (t^3)p_3 \]

Regroup into coefficients of \( t \):

\[ x(t) = (-p_0 + 3p_1 - 3p_2 + p_3)t^3 + (3p_0 - 6p_1 + 3p_2)t^2 + (-3p_0 + 3p_1)t + (p_0) \]

\[
\begin{align*}
\text{a} &= (-p_0 + 3p_1 - 3p_2 + p_3) \\
\text{b} &= (3p_0 - 6p_1 + 3p_2) \\
\text{c} &= (-3p_0 + 3p_1) \\
\text{d} &= (p_0)
\end{align*}
\]

- Good for fast evaluation, precompute constant coefficients \((a,b,c,d)\)
Cubic polynomial form

Start with Bernstein form:

\[ x(t) = (-t^3 + 3t^2 - 3t + 1)p_0 + (3t^3 - 6t^2 + 3t)p_1 + (-3t^3 + 3t^2)p_2 + (t^3)p_3 \]

Regroup into coefficients of \( t \):

\[ x(t) = (-p_0 + 3p_1 - 3p_2 + p_3)t^3 + (3p_0 - 6p_1 + 3p_2)t^2 + (-3p_0 + 3p_1)t + (p_0)1 \]

\[
\begin{align*}
\begin{array}{l}
x(t) = at^3 + bt^2 + ct + d \\
\begin{align*}
a & = (-p_0 + 3p_1 - 3p_2 + p_3) \\
b & = (3p_0 - 6p_1 + 3p_2) \\
c & = (-3p_0 + 3p_1) \\
d & = (p_0)
\end{align*}
\end{array}
\end{align*}
\]

- Good for fast evaluation, precompute constant coefficients \((a,b,c,d)\)
- Not much geometric intuition
Cubic matrix form

\[ \mathbf{x}(t) = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} & \bar{d} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \]

\[ \bar{a} = (-p_0 + 3p_1 - 3p_2 + p_3) \]
\[ \bar{b} = (3p_0 - 6p_1 + 3p_2) \]
\[ \bar{c} = (-3p_0 + 3p_1) \]
\[ \bar{d} = (p_0) \]

\[ \mathbf{x}(t) = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \]

\[
\mathbf{G}_{bez} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}
\]
\[
\mathbf{B}_{bez} = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}
\]

- Can construct other cubic curves by just using different basis matrix \( \mathbf{B} \)
- Hermite, Catmull-Rom, B-Spline, ...
Cubic matrix form

- 3 parallel equations, in \( x, y \) and \( z \):

\[
\begin{align*}
\mathbf{x}_x(t) &= \begin{bmatrix} p_{0x} & p_{1x} & p_{2x} & p_{3x} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \\
\mathbf{x}_y(t) &= \begin{bmatrix} p_{0y} & p_{1y} & p_{2y} & p_{3y} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \\
\mathbf{x}_z(t) &= \begin{bmatrix} p_{0z} & p_{1z} & p_{2z} & p_{3z} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}
\end{align*}
\]
Matrix form

• Bundle into a single matrix

\[
x(t) = \begin{bmatrix} p_{0x} & p_{1x} & p_{2x} & p_{3x} \\ p_{0y} & p_{1y} & p_{2y} & p_{3y} \\ p_{0z} & p_{1z} & p_{2z} & p_{3z} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}
\]

\[
x(t) = G_{Bez} B_{Bez} T
\]

\[
x(t) = C \ T
\]

• Efficient evaluation
  • Precompute \( C \)
  • Take advantage of existing 4x4 matrix hardware support
Drawing Bézier curves

• Generally no low-level support for drawing smooth curves
  • I.e., GPU draws only straight line segments

• Need to break curves into line segments or individual pixels

• Approximating curves as series of line segments called tessellation

• Tessellation algorithms
  • Uniform sampling
  • Adaptive sampling
  • Recursive subdivision
• Approximate curve with $N$-1 straight segments
  • $N$ chosen in advance
  • Evaluate
    \[ x_i = x(t_i) \text{ where } t_i = \frac{i}{N} \text{ for } i = 0, 1, \ldots, N \]
  • Connect the points with lines

• Too few points?
  • Bad approximation
  • “Curve” is faceted

• Too many points?
  • Slow to draw too many line segments
  • Segments may draw on top of each other

Uniform sampling
Adaptive Sampling

• Use only as many line segments as you need
  • Fewer segments where curve is mostly flat
  • More segments where curve bends
  • Segments never smaller than a pixel

• Various schemes for sampling, checking results, deciding whether to sample more
Recursive Subdivision

• Any cubic (or $k$-th order) curve segment can be expressed as a cubic (or $k$-th order) Bézier curve

“Any piece of a cubic (or $k$-th order) curve is itself a cubic (or $k$-th order) curve”

• Therefore, any Bézier curve can be subdivided into smaller Bézier curves
• de Casteljau construction points are the control points of two Bézier sub-segments $(p_0, q_0, r_0, x)$ and $(x, r_1, q_2, p_3)$.
Adaptive subdivision algorithm

1. Use de Casteljau construction to split Bézier segment in middle ($t=0.5$)

2. For each half
   - If “flat enough”: draw line segment
   - Else: recurse from 1. for each half

   • Curve is flat enough if hull is flat enough
   • Test how far away midpoints are from straight segment connecting start and end
     • If about a pixel, then hull is flat enough
Curves

• Introduction
• Polynomial curves
• Bézier curves
• Drawing Bézier curves
• Piecewise curves
More control points

• Cubic Bézier curve limited to 4 control points
  • Cubic curve can only have one inflection
  • Need more control points for more complex curves

• \(k-1\) order Bézier curve with \(k\) control points
  
  - Hard to control and hard to work with
    • Intermediate points don’t have obvious effect on shape
    • Changing any control point changes the whole curve

• Want **local support**
  • Each control point only influences nearby portion of curve
Piecewise curves (splines)

- Sequence of simple (low-order) curves, end-to-end
  - Piecewise polynomial curve, or splines

- Sequence of line segments
  - Piecewise linear curve (linear or first-order spline)

- Sequence of cubic curve segments
  - Piecewise cubic curve, here piecewise Bézier (cubic spline)
Piecewise cubic Bézier curve

- Given $3N + 1$ points $p_0, p_1, \ldots, p_{3N}$
- Define $N$ Bézier segments:

$$x_0(t) = B_0(t)p_0 + B_1(t)p_1 + B_2(t)p_2 + B_3(t)p_3$$

$$x_1(t) = B_0(t)p_3 + B_1(t)p_4 + B_2(t)p_5 + B_3(t)p_6$$

$$\vdots$$

$$x_{N-1}(t) = B_0(t)p_{3N-3} + B_1(t)p_{3N-2} + B_2(t)p_{3N-1} + B_3(t)p_{3N}$$
• **Global parameter** \( u, \ 0 \leq u \leq 3N \)

\[
x(u) = \begin{cases} 
  x_0(\frac{1}{3}u), & 0 \leq u \leq 3 \\
  x_1(\frac{1}{3}u - 1), & 3 \leq u \leq 6 \\
  \vdots & \\
  x_{N-1}(\frac{1}{3}u - (N - 1)), & 3N - 3 \leq u \leq 3N 
\end{cases}
\]

\[
x(u) = x_i \left( \frac{1}{3}u - i \right), \text{ where } i = \left\lfloor \frac{1}{3}u \right\rfloor
\]
• Want smooth curves
• $C^0$ continuity
  • No gaps
  • Segments match at the endpoints
• $C^1$ continuity: first derivative is well defined
  • No corners
  • Tangents/normals are $C^0$ continuous (no jumps)
• $C^2$ continuity: second derivative is well defined
  • Tangents/normals are $C^1$ continuous
  • Important for high quality reflections on surfaces
Piecewise cubic Bézier curve

- $C^0$ continuous by construction
- $C^1$ continuous at segment endpoints $p_{3i}$ if $p_{3i} - p_{3i-1} = p_{3i+1} - p_{3i}$
- $C^2$ is harder to get
Piecewise cubic Bézier curves

• Used often in 2D drawing programs

• Inconveniences
  • Must have 4 or 7 or 10 or 13 or ... (1 plus a multiple of 3) control points
  • Some points interpolate (endpoints), others approximate (handles)
  • Need to impose constraints on control points to obtain $C^1$ continuity
  • $C^2$ continuity more difficult

• Solutions
  • User interface using “Bézier handles”
  • Generalization to B-splines, next time
Bézier handles

• Segment end points (interpolating) presented as curve control points
• Midpoints (approximating points) presented as “handles”
• Can have option to enforce $C^1$ continuity