Today
Curves
• Introduction
• Polynomial curves
• Bézier curves

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
• Specify parameter values over time
• 2D curve editor

How to represent curves
• Specify every point along a curve?
  - Hard to get precise, smooth results
  - Too much data, too hard to work with
• Specify a curve using a small number of “control points”
  - Known as a spline curve or just spline

Approximating splines
• Curve is “influenced” by control points
  • Various types & techniques
  • Most common: polynomial functions
    - Bézier
    - (B-spline, NURBS)
  • Focus on Bézier splines

Interpolating splines
• 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

Mathematical definition
• A vector valued function of one variable $x(t)$
  - Given $t$, compute a 3D point $x=(x,y,z)$
  - 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 corresponds to speed

**Questions?**

**Today**
Curves
- Introduction
- Polynomial curves
- Bézier curves

**Polynomial functions**
- **Linear**: \( f(t) = at + b \) (1st order)
- **Quadratic**: \( f(t) = at^2 + bt + c \) (2nd order)
- **Cubic**: \( f(t) = at^3 + bt^2 + ct + d \) (3rd order)

**Polynomial curves**
- **Linear**: \( \mathbf{x}(t) = at + b \)
  \( \mathbf{x} = (x, y, z) \), \( \mathbf{a} = (a_x, a_y, a_z) \), \( \mathbf{b} = (b_x, b_y, b_z) \)
- Evaluated as
  \( x(t) = a_xt + b_x \)
  \( y(t) = a_yt + b_y \)
  \( z(t) = a_zt + b_z \)

- **Quadratic**: \( \mathbf{x}(t) = at^2 + bt + c \) (2nd order)
- **Cubic**: \( \mathbf{x}(t) = at^3 + bt^2 + ct + d \) (3rd order)

- We usually define the curve for \( 0 \leq t \leq 1 \)
Control points

- Polynomial coefficients \(a, b, c, d\) can be interpreted as control points
  - Remember \(a, b, c, d\) have \(x, y, z\) components each
- Unfortunately, don’t intuitively describe shape of curve
- Main objective of curve representation is to come up with intuitive control points

Control points

- How many control points?
  - Two points define a line (1st order)
  - Three points define a quadratic curve (2nd order)
  - Four points define a cubic curve (3rd order)
  - \(k+1\) points define a \(k\)-order curve
- Let’s start with a line...

First order curve

- Based on linear interpolation (LERP)
  - Weighted average between two values
  - “Value” could be a number, vector, color, ...
- Interpolate between points \(p_0\) and \(p_1\) with parameter \(t\)
  - Defines a “curve” that is straight (first-order spline)
  - \(t=0\) corresponds to \(p_0\)
  - \(t=1\) corresponds to \(p_1\)
  - \(t=0.5\) corresponds to midpoint

\[
x(t) = Lerp(t, p_0, p_1) = (1-t)p_0 + tp_1
\]

Linear interpolation

- Three different ways to write it
  - All equivalent
  - Different properties become apparent
1. Weighted sum of the control points
   \[
x(t) = p_0(1-t) + p_1t
\]
2. Polynomial in \(t\)
   \[
x(t) = (p_1 - p_0)t + p_0
\]
3. Matrix form
   \[
x(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}
\]

Weighted average

\[
x(t) = (1-t)p_0 + tp_1
\]
\[
= B_0(t)p_0 + B_1(t)p_1, \text{ where } B_0(t) = 1-t \text{ and } B_1(t) = t
\]

- Weights are a function of \(t\)
  - Sum is always 1, for any value of \(t\)
  - Also known as blending functions

Linear polynomial

- Curve is based at point \(p_0\)
- Add the vector, scaled by \(t\)
Matrix form

\[ x(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = GBT \]
- Geometry matrix \( G = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \)
- Geometric basis \( B = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \)
- Polynomial basis \( T = \begin{bmatrix} t \\ 1 \end{bmatrix} \)
- In components \( x(t) = \begin{bmatrix} p_{0x} & p_{1x} \\ p_{0y} & p_{1y} \\ p_{0z} & p_{1z} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} \)

Tangent

- For a straight line, the tangent is constant \( x'(t) = p_1 - p_0 \)
- Weighted average \( x'(t) = (-1)p_0 + (+1)p_1 \)
- Polynomial \( x'(t) = 0t + (p_1 - p_0) \)
- Matrix form \( x'(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \)

Questions?

Lissajou curves

Today

Curves
- Introduction
- Polynomial curves
- Bézier curves

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

- Higher order extension of linear interpolation

![Linear, Quadratic, Cubic Bézier curves]

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

- Many demo applets online
  - [http://www.cs.unc.edu/~mantler/research/bezier/](http://www.cs.unc.edu/~mantler/research/bezier/)
  - [http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCExamples/Bezier/bezier.html](http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCExamples/Bezier/bezier.html)

Cubic Bézier curve

- Most common case
- Defined by 4 control points
- Two interpolated endpoints
- Two midpoints control the tangent at the endpoints

![Cubic Bézier curve]

- Define the point $x$ on the curve as a function of parameter $t$

![Cubic Bézier curve]

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)
**de Casteljau Algorithm**

- Given the control points
- A value of $t$
- Here $t \approx 0.25$

Let $p_0, p_1, p_2, p_3$ be the control points.

$q_0(t) = \text{Lerp}(t, p_0, p_1)$
$q_1(t) = \text{Lerp}(t, p_1, p_2)$
$q_2(t) = \text{Lerp}(t, p_2, p_3)$

$r_0(t) = \text{Lerp}(t, q_0(t), q_1(t))$
$r_1(t) = \text{Lerp}(t, q_1(t), q_2(t))$

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

**de Casteljau Algorithm**

- Applets
  - [http://www2.mat.dtu.dk/people/J.Gravesen/cagd/decast.html](http://www2.mat.dtu.dk/people/J.Gravesen/cagd/decast.html)

**Recursive linear interpolation**

$x = \text{Lerp}(t, r_0, r_1)$
$q_0 = \text{Lerp}(t, p_0, p_1)$
$q_1 = \text{Lerp}(t, p_1, p_2)$
$q_2 = \text{Lerp}(t, p_2, p_3)$

$x = \text{Lerp}(t, r_0, r_1)$
Expand the LERPs

\[ q_i(t) = Lerp(t, p_i, p_{i+1}) = (1-t)p_i + tp_{i+1}, \]
\[ q_i(t) = Lerp(t, p_i, p_{i+1}) = (1-t)p_i + tp_{i+1}, \]
\[ q_i(t) = Lerp(t, p_i, p_{i+1}) = (1-t)p_i + tp_{i+1}. \]

\[ r_i(t) = Lerp(t, q_{i-1}(t), q_i(t)) = (1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2}) \]
\[ r_i(t) = Lerp(t, q_{i-1}(t), q_i(t)) = (1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2}). \]

\[ x(t) = Lerp(t, r_i(t), r_{i+1}(t)) = (1-t)((1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2})) + t((1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2})). \]

Weighted average of control points

- Regroup

\[ x(t) = (1-t)((1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2})) + t((1-t)((1-t)p_i + tp_{i+1}) + t((1-t)p_{i+1} + tp_{i+2})). \]

\[ x(t) = (1-t)(p_i + 3(1-t)^2p_{i+1} + 3(1-t)^3p_{i+2} + tp_{i+3}). \]

\[ x(t) = \frac{\sum_{i=0}^{n} B_i(t) p_i}{\sum_{i=0}^{n} B_i(t)} + t \frac{\sum_{i=0}^{n} B_i(t) p_i}{\sum_{i=0}^{n} B_i(t)} + t \frac{\sum_{i=0}^{n} B_i(t) p_i}{\sum_{i=0}^{n} B_i(t)} \]

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

\[ B_i(t) + B_{i+1}(t) + \cdots + B_0(t) = 1. \]

Cubic Bernstein polynomials

- Partition of unity, weights always add to 1
- Endpoint interpolation, \( B_0 \) and \( B_3 \) go to 1

General Bernstein polynomials

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

General Béziers curves

- \( n \)-th order Bernstein polynomials form \( n \)-th order Bézier curves

\[ B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \]
\[ x(t) = \sum_{i=0}^{n} B_i^n(t) p_i. \]

Bézier curve properties

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

- **Convex hull of a set of points**
  - Polyhedral volume such that line connecting any two points 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 diagram]

Convex hull property

- Bézier curve is a convex combination of the control points
- Curve is always inside the convex hull
  - Makes curve predictable
  - Allows culling, intersection testing, adaptive tessellation

![Convex hull property diagram]

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”

![Variation diminishing property diagram]

Affine invariance

Transforming Bézier curves

- **Two ways**
  - Transform the control points, then compute resulting spline point
  - Compute spline point, then transform it
- Either way, get the same point!
  - Curve is defined via affine combination of points
  - Invariant under affine transformations
  - Convex hull property always remains

![Affine invariance diagram]

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 \( t \):

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

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

![Cubic polynomial form diagram]

Cubic matrix form

\[
x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \[t^3] & \end{bmatrix}
\]

\[
x(t) = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \[1] & \end{bmatrix}
\]

- Other cubic splines use different basis matrix \( B \)
  - Hermite, Catmull-Rom, B-Spline, ...
Cubic matrix form

- 3 parallel equations, in x, y and z:

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

Matrix form

- Bundle into a single matrix

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

- Efficient evaluation
  - Precompute C
  - Take advantage of existing 4x4 matrix hardware support

Next time

- Curves with multiple segments
- Extending to curves to surfaces