Announcements

- Homework assignment 5 due tomorrow, Nov 8 at 1:30pm
- Late submissions for assignment 4 will be accepted
CSE 169: Computer Animation

- Most recent course web site is from 2009:
  - http://graphics.ucsd.edu/courses/cse169_w09
- PixelActive’s CityScape:
  - http://www.youtube.com/watch?v=yrqm9qK_Mlo
CSE 190: Shader Programming

- Instructor: Wolfgang Engel, CEO and Co-Founder of Confetti Interactive

Lecture topics:
- Introduction to DirectX 11.1 Compute
- Simple Compute Case Studies
- DirectCompute performance optimization
- Direct3D 11.1 Graphics Pipeline
- Physically Based Lighting
- Deferred Lighting, AA
- Shadows
- Order-Independent Transparency
- Global Illumination Algorithms in Games
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves
Modeling

- Creating 3D objects
- How to construct complex surfaces?
- Goal
  - Specify objects with control points
  - Objects should be visually pleasing (smooth)
- Start with curves, then generalize to surfaces

- Next: What can curves be used for?
Curves

- Surface of revolution
Curves

- Extruded/swept surfaces
Curves

- **Animation**
  - Provide a “track” for objects
  - Use as camera path
Video

- Bezier Curves
  - [http://www.youtube.com/watch?v=hlDYJNEiYvU](http://www.youtube.com/watch?v=hlDYJNEiYvU)
Curves

- Can be generalized to surface patches
Curve Representation

- Specify many points along a curve, connect with lines?
  - Difficult to get precise, smooth results across magnification levels
  - Large storage and CPU requirements
  - How many points are enough?
- Specify a curve using a small number of “control points”
  - Known as a spline curve or just spline
Spline: Definition

- **Wikipedia:**
  - Term comes from flexible spline devices used by shipbuilders and draftsmen to draw smooth shapes.
  - Spline consists of a long strip fixed in position at a number of points that relaxes to form a smooth curve passing through those points.
Interpolating Control Points

- “Interpolating” means that curve goes through all control points
- Seems most intuitive
- Surprisingly, not usually the best choice
  - Hard to predict behavior
  - Hard to get aesthetically pleasing curves
Approximating Control Points

- Curve is “influenced” by control points

- Various types
- Most common: polynomial functions
  - Bézier spline (our focus)
  - B-spline (generalization of Bézier spline)
  - NURBS (Non Uniform Rational Basis Spline): used in CAD tools
A vector valued function of one variable $\mathbf{x}(t)$

- Given $t$, compute a 3D point $\mathbf{x}=(x,y,z)$
- Could be interpreted as three functions: $x(t)$, $y(t)$, $z(t)$
- Parameter $t$ “moves a point along the curve”
Tangent Vector

- Derivative $\mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = (x'(t), y'(t), z'(t))$
- Vector $\mathbf{x}'$ points in direction of movement
- Length corresponds to speed
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions

- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves
Polynomial Functions

- **Linear:** $f(t) = at + b$
  
  $(1^{st} \text{ order})$

- **Quadratic:** $f(t) = at^2 + bt + c$
  
  $(2^{nd} \text{ order})$

- **Cubic:** $f(t) = at^3 + bt^2 + ct + d$
  
  $(3^{rd} \text{ order})$
Polynomial Curves

- **Linear** \( x(t) = at + b \)
  \[ x = (x, y, z), \ a = (a_x, a_y, a_z), \ b = (b_x, b_y, b_z) \]

  - Evaluated as:
    \[ x(t) = a_x t + b_x \]
    \[ y(t) = a_y t + b_y \]
    \[ z(t) = a_z t + b_z \]
Polynomial Curves

- **Quadratic**: \( x(t) = at^2 + bt + c \)
  (2\(^{nd}\) order)

- **Cubic**: \( x(t) = at^3 + bt^2 + ct + d \)
  (3\(^{rd}\) 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, they do not intuitively describe the shape of the curve
- Goal: intuitive control points
Control Points

- How many control points?
  - Two points define a line (1\textsuperscript{st} order)
  - Three points define a quadratic curve (2\textsuperscript{nd} order)
  - Four points define a cubic curve (3\textsuperscript{rd} 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 $\mathbf{p}_0$ and $\mathbf{p}_1$ with parameter $t$
  - Defines a “curve” that is straight (first-order spline)
  - $t=0$ corresponds to $\mathbf{p}_0$
  - $t=1$ corresponds to $\mathbf{p}_1$
  - $t=0.5$ corresponds to midpoint

\[
x(t) = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) = (1 - t)\mathbf{p}_0 + t \mathbf{p}_1
\]
Linear Interpolation

- Three equivalent ways to write it
  - Expose different properties

1. Regroup for points $p$
   \[ x(t) = p_0(1 - t) + p_1 t \]

2. Regroup for $t$
   \[ x(t) = (p_1 - p_0)t + p_0 \]

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

\[ x(t) = (p_1 - p_0) \cdot t + p_0 \]

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

\[ x(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = \text{GBT} \]

- 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
  \[ x'(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves
Bézier Curves

- Are a higher order extension of linear interpolation
Bézier Curves

- Give intuitive control over curve with control points
  - Endpoints are interpolated, intermediate points are approximated
  - Convex Hull property
- Many demo applets online, for example:
  - Demo: http://www.cs.princeton.edu/~min/cs426/jar/bezier.html
  - http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCEXamples/Bezier/bezier.html
Cubic Bézier Curve

- Most commonly used case
- Defined by four control points:
  - Two interpolated endpoints (points are on the curve)
  - Two points control the tangents at the endpoints
- Points \( x \) on curve defined as function of parameter \( t \)
Algorithmic Construction

- Algorithmic construction
  - *De Casteljau* algorithm, developed at Citroen in 1959, named after its inventor Paul de Casteljau (pronounced “Cast-all-’Joe”)
  - Developed independently from Bézier’s work: Bézier created the formulation using blending functions, Casteljau devised the recursive interpolation algorithm
De Casteljau Algorithm

- A recursive series of linear interpolations
  - Works for any order Bezier function, not only cubic
- Not very efficient to evaluate
  - Other forms more commonly used
- But:
  - Gives intuition about the geometry
  - Useful for subdivision
De Casteljau Algorithm

- **Given:**
  - Four 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

\[ r_0(t) = \text{Lerp}(t, q_0(t), q_1(t)) \]
\[ r_1(t) = \text{Lerp}(t, q_1(t), q_2(t)) \]
De Casteljau Algorithm

\[ \mathbf{x}(t) = \text{Lerp}\left(t, \mathbf{r}_0(t), \mathbf{r}_1(t)\right) \]
De Casteljau Algorithm

Applets
- Demo: [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

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

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

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

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

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

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

\[
q_0(t) = \text{Lerp}(t, p_0, p_1) = (1-t)p_0 + tp_1
\]

\[
q_1(t) = \text{Lerp}(t, p_1, p_2) = (1-t)p_1 + tp_2
\]

\[
q_2(t) = \text{Lerp}(t, p_2, p_3) = (1-t)p_2 + tp_3
\]

\[
r_0(t) = \text{Lerp}(t, q_0(t), q_1(t)) = (1-t)((1-t)p_0 + tp_1) + t((1-t)p_1 + tp_2)
\]

\[
r_1(t) = \text{Lerp}(t, q_1(t), q_2(t)) = (1-t)((1-t)p_1 + tp_2) + t((1-t)p_2 + tp_3)
\]

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

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

\[
+ t((1-t)((1-t)p_1 + tp_2) + t((1-t)p_2 + tp_3))
\]
Weighted Average of Control Points

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

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

  \[
x(t) = \underbrace{(-t^3 + 3t^2 - 3t + 1)}_{B_0(t)} p_0 + \underbrace{(3t^3 - 6t^2 + 3t)}_{B_1(t)} p_1
    + \underbrace{(-3t^3 + 3t^2)}_{B_2(t)} p_2 + \underbrace{(t^3)}_{B_3(t)} p_3
  \]
Cubic Bernstein Polynomials

\[ \mathbf{x}(t) = B_0(t)\mathbf{p}_0 + B_1(t)\mathbf{p}_1 + B_2(t)\mathbf{p}_2 + B_3(t)\mathbf{p}_3 \]

The cubic Bernstein polynomials:

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

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

- Weights \( B_i(t) \) add up to 1 for any value of \( t \)
General Bernstein Polynomials

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

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

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

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

n! = factorial of n
(n+1)! = n! x (n+1)
General Bézier 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

Overview:

- Convex Hull property
- Affine Invariance
Definitions

- **Convex hull** of a set of points:
  - Polyhedral volume created such that all lines connecting any two points lie completely inside it (or on its boundary)

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

- **Any convex combination of a set of points lies within the convex hull**
Convex Hull Property

- A Bézier curve is a convex combination of the control points (by definition, see Bernstein polynomials)
- A Bézier curve is always inside the convex hull
  - Makes curve predictable
  - Allows culling, intersection testing, adaptive tessellation
Affine Invariance

Transforming Bézier curves

Two ways to transform:

- Transform the control points, then compute resulting spline points
- Compute spline points, then transform them

Either way, we get the same points

- Curve is defined via affine combination of points
- Invariant under affine transformations (i.e., translation, scale, rotation, shear)
- Convex hull property remains true
Cubic Polynomial Form

Start with Bernstein form:

\[ x(t) = (\begin{array}{c}
-t^3 + 3t^2 - 3t + 1\\
3t^3 - 6t^2 + 3t\\
-3t^3 + 3t^2\\
t^3
end{array})p_0 + (\begin{array}{c}
3t^3 - 6t^2 + 3t\\
-3t^3 + 3t^2\\
(t^3)
end{array})p_1 + (\begin{array}{c}
-3t^3 + 3t^2\\
(t^3)
end{array})p_2 + (\begin{array}{c}
(t^3)
end{array})p_3 \]

Regroup into coefficients of \( t \):

\[ x(t) = (\begin{array}{c}
-p_0 + 3p_1 - 3p_2 + p_3\\
3p_0 - 6p_1 + 3p_2\\
-3p_0 + 3p_1\\
p_0
end{array})t^3 + (\begin{array}{c}
3p_0 - 6p_1 + 3p_2\\
-3p_0 + 3p_1\\
p_0
end{array})t^2 + (\begin{array}{c}
-p_0 + 3p_1 - 3p_2 + p_3\\
3p_0 - 6p_1 + 3p_2\\
-3p_0 + 3p_1\\
p_0
end{array})t + (\begin{array}{c}
p_0
end{array}) \]

\[ x(t) = at^3 + bt^2 + ct + d \]

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

- Good for fast evaluation
  - Precompute constant coefficients (\( a,b,c,d \))
- Not much geometric intuition
Cubic Matrix Form

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

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

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

\[
\bar{c} = (-3p_0 + 3p_1)
\]

\[
d = (p_0)
\]

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

\[
G_{Bez} \quad B_{Bez} \quad T
\]

- Other types of cubic splines use different basis matrices \( B_{Bez} \)
Cubic Matrix Form

- In 3D: 3 equations for \( x, y \) and \( z \):

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

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

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

- **Bundle into a single matrix**

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

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

\[
x(t) = C T
\]

- **Efficient evaluation**
  - Pre-compute \( C \)
  - Take advantage of existing 4x4 matrix hardware support
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions

- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves
Drawing Bézier Curves

- Draw *line segments* or individual pixels
- Approximate the curve as a series of line segments (*tessellation*)
  - Uniform sampling
  - Adaptive sampling
  - Recursive subdivision
Uniform Sampling

- Approximate curve with \( N \) straight segments
  - \( N \) chosen in advance
  - Evaluate
    \[
    x_i = x(t_i) \quad \text{where} \quad t_i = \frac{i}{N} \quad \text{for} \quad i = 0, 1, \ldots, N
    \]
    
    \[
    x_i = \tilde{a} \frac{i^3}{N^3} + \tilde{b} \frac{i^2}{N^2} + \tilde{c} \frac{i}{N} + d
    \]
  - Connect the points with lines

- Too few points?
  - Poor approximation
  - “Curve” is faceted

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

- Any cubic curve segment can be expressed as a Bézier curve
- Any piece of a cubic curve is itself a cubic curve
- Therefore:
  - Any Bézier curve can be broken down into smaller Bézier curves
De Casteljau Subdivision

- De Casteljau construction points are the control points of two Bézier sub-segments.
Adaptive Subdivision Algorithm

- Use De Casteljau construction to split Bézier segment in half
- For each half
  - If “flat enough”: draw line segment
  - Else: recurse
- Curve is flat enough if hull is flat enough
  - Test how far the approximating control points are from a straight segment
    - If less than one pixel, the hull is flat enough
Drawing Bézier Curves With OpenGL

- Indirect OpenGL support for drawing curves:
  - Define evaluator map (`glMap`)
  - Draw line strip by evaluating map (`glEvalCoord`)
  - Optimize by pre-computing coordinate grid (`glMapGrid` and `glEvalMesh`)

- More details about OpenGL implementation:
  - [http://www.cs.duke.edu/courses/fall09/cps124/notes/12_curves/opengl_nurbs.pdf](http://www.cs.duke.edu/courses/fall09/cps124/notes/12_curves/opengl_nurbs.pdf)
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves
More Control Points

- Cubic Bézier curve limited to 4 control points
  - Cubic curve can only have one inflection (point where curve changes direction of bending)
  - 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

- Sequence of line segments
  - Piecewise linear curve

- Sequence of simple (low-order) curves, end-to-end
  - Known as a piecewise polynomial curve

- Sequence of cubic curve segments
  - Piecewise cubic curve (here piecewise Bézier)
Parametric Continuity

- **C⁰ continuity:**
  - Curve segments are connected

- **C¹ continuity:**
  - C⁰ & 1st-order derivatives agree
  - Curves have same tangents
  - Relevant for smooth shading

- **C² continuity:**
  - C¹ & 2nd-order derivatives agree
  - Curves have same tangents and curvature
  - Relevant for high quality reflections