• Return of midterm delayed to next Tuesday; scores will be on GradeSource on Monday

• Homework project 5 (scene graphs) due Monday 11/17

• Homework project 6 on-line as of today

• Course will get CAPE’d next Tuesday
Today

Curves

- Review of polynomial curves
- Bézier curves
- Piecewise-cubic Bézier curves
Polynomial functions

• **Linear:** \( f(t) = at + b \)  
  (1\textsuperscript{st} order)

• **Quadratic:** \( f(t) = at^2 + bt + c \)  
  (2\textsuperscript{nd} order)

• **Cubic:** \( f(t) = at^3 + bt^2 + ct + d \)  
  (3\textsuperscript{rd} order)
Linear interpolation

- Three different ways to write it
  - All equivalent
  - Different properties become apparent

1. Weighted sum of the control points

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

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 \textit{blending functions}
**Linear polynomial**

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

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

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

- 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 \[ x(t) = \begin{bmatrix} p_{0x} & p_{1x} \\ p_{0y} & p_{1y} \\ p_{0z} & p_{1z} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} \]
The derivative of a curve represents the tangent vector to the curve at some point.
Tangent of polynomial curves

- Easy to compute
  - Example: cubic curve

  \[ x(t) = \bar{a}t^3 + \bar{b}t^2 + \bar{c}t + d \quad \quad x'(t) = \frac{dx}{dt}(t) = 3\bar{a}t^2 + 2\bar{b} + \bar{c} \]

  \[
  x(t) = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} & d \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \quad \quad x'(t) = \frac{dx}{dt}(t) = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} & d \end{bmatrix} \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}
  \]

- Notice \( x'(t) \) is a vector
Today

Curves

- Review of polynomial curves
- Bézier curves
- Piecewise-cubic Bézier curves
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. Examples:
  - [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 four control points
- Two interpolated endpoints (points are on the curve)
- Two midpoints control the tangent at the endpoints
Cubic Bézier curve

- Define the point $x$ on the curve as a function of parameter $t$
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, developed at Citroen in 1959
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 $t \approx 0.25$
The de Casteljau Algorithm involves iteratively applying linear interpolation (Lerp) to find points along the curve defined by a set of control points.

Let's denote the control points as \( p_0, p_1, p_2, p_3 \) and the points along the curve as \( q_0, q_1, q_2 \). The algorithm proceeds as follows:

1. \( q_0(t) = \text{Lerp}(t, p_0, p_1) \)
2. \( q_1(t) = \text{Lerp}(t, p_1, p_2) \)
3. \( q_2(t) = \text{Lerp}(t, p_2, p_3) \)

Each \( q_i(t) \) is calculated by linearly interpolating between the control points, weighted by \( t \) for \( t \in [0, 1] \). This process is repeated until the point on the curve at \( t \) is found.
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

\[ x(t) = \text{Lerp}\left( t, r_0(t), r_1(t) \right) \]
de Casteljau algorithm

• Applets
  – Demo: http://www2.mat.dtu.dk/people/J.Gravesen/cagd/decast.html
Recursive linear interpolation

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

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

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

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

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

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

\[
p_0
\]

\[
p_1
\]

\[
p_2
\]

\[
p_3
\]

\[
p_4
\]
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

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

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

\[ x(t) = \underbrace{(-t^3 + 3t^2 - 3t + 1)p_0}_{B_0(t)} + \underbrace{(3t^3 - 6t^2 + 3t)p_1}_{B_1(t)} + \underbrace{(-3t^3 + 3t^2)p_2}_{B_2(t)} + \underbrace{(t^3)p_3}_{B_3(t)} \]
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, weights always add up to 1
- Endpoint interpolation, \( B_0 \) and \( B_3 \) go to 1
General Bernstein polynomials

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

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

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

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

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

n! = factorial of n
\( (n+1)! = n! \times (n+1) \)
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

- 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

![Diagram showing a convex hull with points p₀, p₁, p₂, and p₃]
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
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”
Affine invariance

Transforming Bézier curves

• Two ways
  - Transform the control points, then compute resulting spline point
  - Compute spline point, then transform it

• Either way, we get the same point!
  - Curve is defined via affine combination of points
  - Invariant under affine transformations
  - Convex hull property always remains
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 \( t \):

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

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

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

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

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

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

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

- 3 parallel equations, in \( x, y \) 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

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

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

\[
x(t) = C T
\]

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

• Generally no low-level support for drawing curves
Drawing Bézier curves

- Generally no low-level support for drawing curves
- Draw \textit{line segments} or individual pixels
- Approximate the curve as a series of line segments (\textit{tessellation})
  - Uniform sampling
  - Adaptive sampling
  - Recursive subdivision
Uniform sampling

- Approximate curve with N straight segments
  - N chosen in advance
  - Evaluate $x_i = x(t_i)$ where $t_i = \frac{i}{N}$ for $i = 0, 1, \ldots, N$

\[
x_i = \bar{a} \frac{i^3}{N^3} + \bar{b} \frac{i^2}{N^2} + \bar{c} \frac{i}{N} + d
\]
  - 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
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 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 up 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

- For each half
  - If flat enough: draw line segment
  - Else: recurse

- Curve is flat enough if hull is flat enough

- Test how far the handles are from a straight segment
  - If it’s about a pixel, the hull is flat
Today

Curves

- Review of polynomial curves
- Bézier curves
- Piecewise-cubic 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 simple (low-order) curves, end-to-end
  - Known as a *piecewise polynomial curve*
- Sequence of line segments
  - *Piecewise linear* curve

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

- 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
Global parameterization

• Given \( N \) curve segments \( x_0(t), x_1(t), \ldots, x_{N-1}(t) \)
• Each is parameterized for \( t \) from 0 to 1
• Define a piecewise curve
  - Global parameter \( u \) from 0 to \( N \)
    \[
    x(u) = \begin{cases} 
    x_0(u), & 0 \leq u \leq 1 \\
    x_1(u-1), & 1 \leq u \leq 2 \\
    \vdots & \\
    x_{N-1}(u-(N-1)), & N-1 \leq u \leq N
    \end{cases}
    \]

  \[x(u) = x_i(u - i), \text{ where } i = \lfloor u \rfloor \quad \text{(and } x(N) = x_{N-1}(1))\]

• Alternate: \( u \) also goes from 0 to 1
  \[
  x(u) = x_i(Nu - i), \text{ where } i = \lfloor Nu \rfloor
  \]
Piecewise-linear curve

- Given $N+1$ points $p_0, p_1, \ldots, p_N$
- Define curve

$$x(u) = \text{Lerp}(u - i, p_i, p_{i+1}), \quad i \leq u \leq i + 1$$
$$= (1 - u + i)p_i + (u - i)p_{i+1}, \quad i = \lfloor u \rfloor$$

- $N+1$ points define $N$ linear segments
- $x(i) = p_i$
- $C^0$ continuous by construction
- $C^1$ at $p_i$ when $p_i - p_{i-1} = p_{i+1} - p_i$
**Piecewise Bézier curve**

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

  $x_0(t) = B_0(t)p_0 + B_1(t)p_1 + B_2(t)p_2 + B_3(t)p_3$
  
  $x_1(t) = B_0(t)p_3 + B_1(t)p_4 + B_2(t)p_5 + B_3(t)p_6$
  
  $\vdots$
  
  $x_{N-1}(t) = B_0(t)p_{3N-3} + B_1(t)p_{3N-2} + B_2(t)p_{3N-1} + B_3(t)p_{3N}$
Piecewise Bézier curve

- $3N+1$ points define $N$ Bézier segments 
- $x(3i)=p_{3i}$ 
- $C^0$ continuous by construction 
- $C^1$ continuous at $p_{3i}$ when $p_{3i} - p_{3i-1} = p_{3i+1} - p_{3i}$ 
- $C^2$ is harder to get
Piecewise 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, others approximate
  - 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
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
Next Time

• Extension to 2D:
  Bezier surface patches