CSE 167

Discussion 7

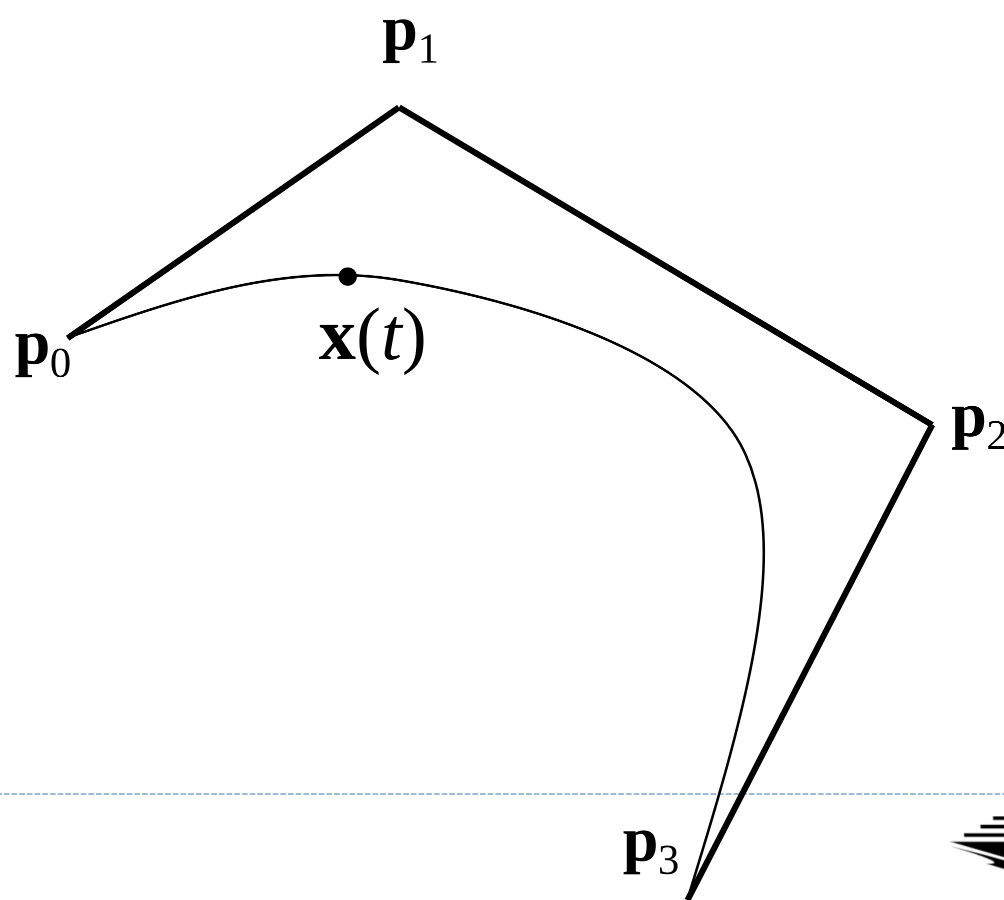
Jimmy
ft Kevin

Announcements

- Project 4 due Friday 2pm
- Late grading for Project 4 is extended an extra week due to Thanksgiving
- Start preparing for midterm + final project!

Cubic Bézier Curve

- Defined by four control points:
  - Two interpolated endpoints (points are on the curve)
  - Two points control the tangents at the endpoints



UCSD

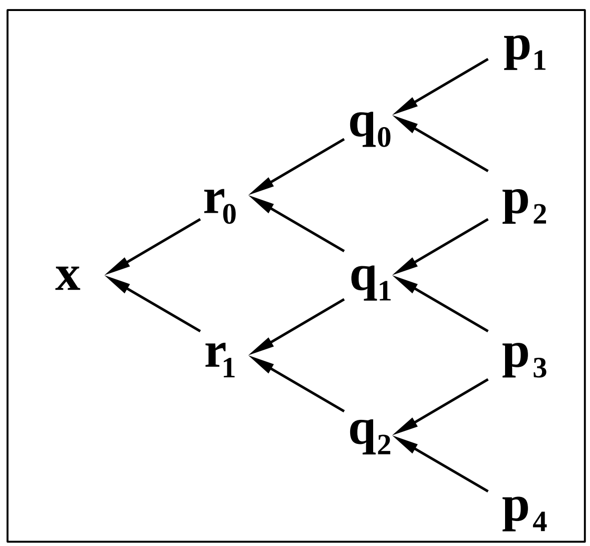
Recursive Linear Interpolation

$$\mathbf{x} = Lerp(t, \mathbf{r}_0, \mathbf{r}_1)$$

$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2)$$

$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3)$$

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



Equivalently…

$$\mathbf{x}(t) = \overbrace{\left(-t^3 + 3t^2 - 3t + 1\right)}^{B_0(t)}\mathbf{p}_0 + \overbrace{\left(3t^3 - 6t^2 + 3t\right)}^{B_1(t)}\mathbf{p}_1$$

$$+ \underbrace{\left(-3t^3 + 3t^2\right)}_{B_2(t)}\mathbf{p}_2 + \underbrace{\left(t^3\right)}_{B_3(t)}\mathbf{p}_3$$

Cubic Polynomial Form

Start with Bernstein form:

$$\mathbf{x}(t) = \left(-t^3 + 3t^2 - 3t + 1\right)\mathbf{p}_0 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_1 + \left(-3t^3 + 3t^2\right)\mathbf{p}_2 + \left(t^3\right)\mathbf{p}_3$$

$$\mathbf{x}(t) = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)t_3 + \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)t_2 + \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)t + \left(\mathbf{p}_0\right)1$$

$$\boxed{\mathbf{x}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d} \qquad \begin{aligned} \mathbf{a} &= \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right) \\ \mathbf{b} &= \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right) \\ \mathbf{c} &= \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right) \\ \mathbf{d} &= \left(\mathbf{p}_0\right) \end{aligned}}$$

- Good for fast evaluation
  - Precompute constant coefficients (**a**,**b**,**c**,**d**)
  - Can also write as a matrix, which is even faster

Global Parameterization

- Given N curve segments $\mathbf{x}_0(t)$, $\mathbf{x}_1(t)$, …, $\mathbf{x}_{N-1}(t)$
- Each is parameterized for t from 0 to 1
- Define a piecewise curve
  - Global parameter u from 0 to N

$$\mathbf{x}(u) = \begin{cases} \mathbf{x}_0(u), & 0 \le u \le 1 \\ \mathbf{x}_1(u-1), & 1 \le u \le 2 \\ \vdots & \vdots \\ \mathbf{x}_{N-1}(u-(N-1)), & N-1 \le u \le N \end{cases}$$

$$\mathbf{x}(u) = \mathbf{x}_i(u-i), \text{ where } i = \left\lfloor u \right\rfloor \quad (\text{and } \mathbf{x}(N) = \mathbf{x}_{N-1}(1))$$

- Alternate solution: u defined from 0 to 1

$$\mathbf{x}(u) = \mathbf{x}_i(Nu - i), \text{ where } i = \left\lfloor Nu \right\rfloor$$

Piecewise Bézier curve

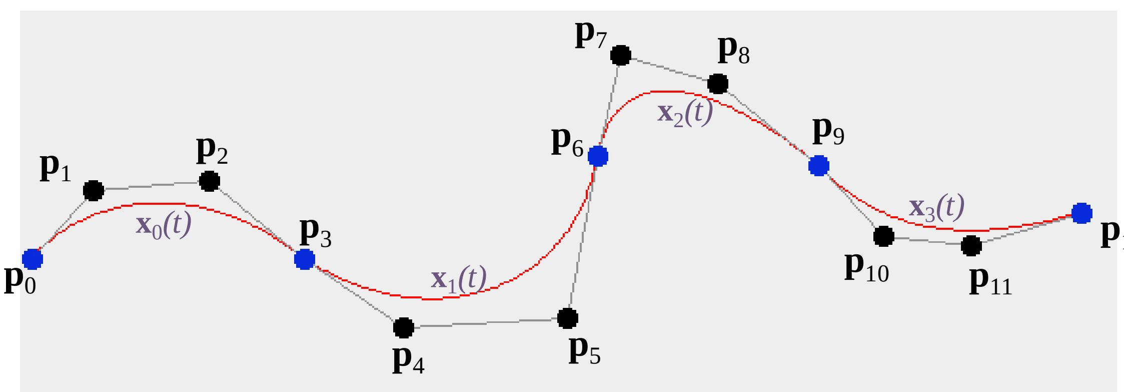
- Given $3N+1$ points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{3N}$
- Define N Bézier segments:

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

$$\mathbf{x}_1(t) = B_0(t)\mathbf{p}_3 + B_1(t)\mathbf{p}_4 + B_2(t)\mathbf{p}_5 + B_3(t)\mathbf{p}_6$$

$$\vdots$$

$$\mathbf{x}_{N-1}(t) = B_0(t)\mathbf{p}_{3N-3} + B_1(t)\mathbf{p}_{3N-2} + B_2(t)\mathbf{p}_{3N-1} + B_3(t)\mathbf{p}_{3N}$$

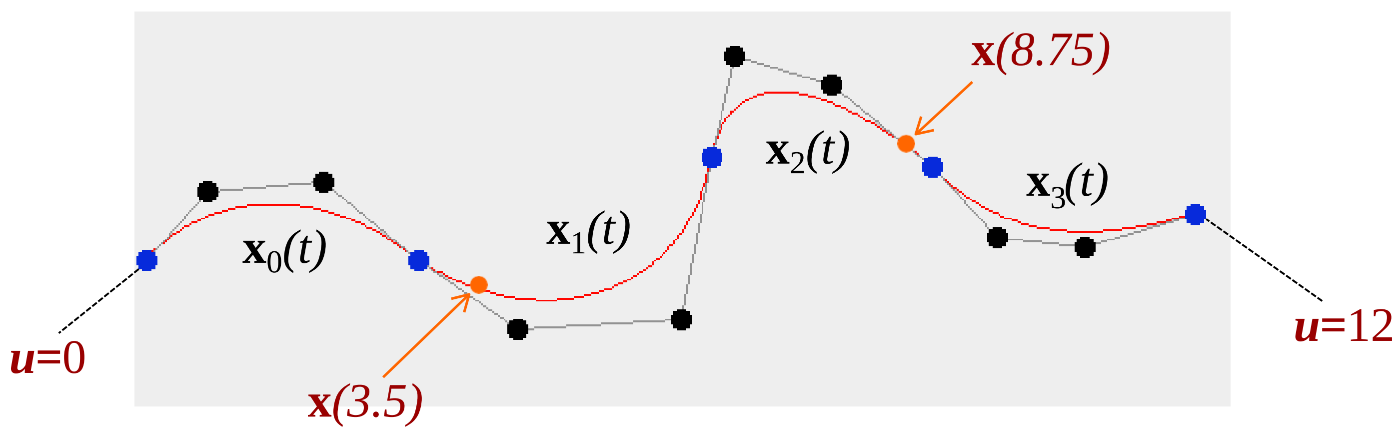


Piecewise Bézier Curve

- Parameter in 0<=u<=3N

$$\mathbf{x}(u) = \begin{cases} \mathbf{x}_0(\frac{1}{3}u), & 0 \le u \le 3 \\ \mathbf{x}_1(\frac{1}{3}u - 1), & 3 \le u \le 6 \\ \vdots & \vdots \\ \mathbf{x}_{N-1}(\frac{1}{3}u - (N-1)), & 3N - 3 \le u \le 3N \end{cases}$$

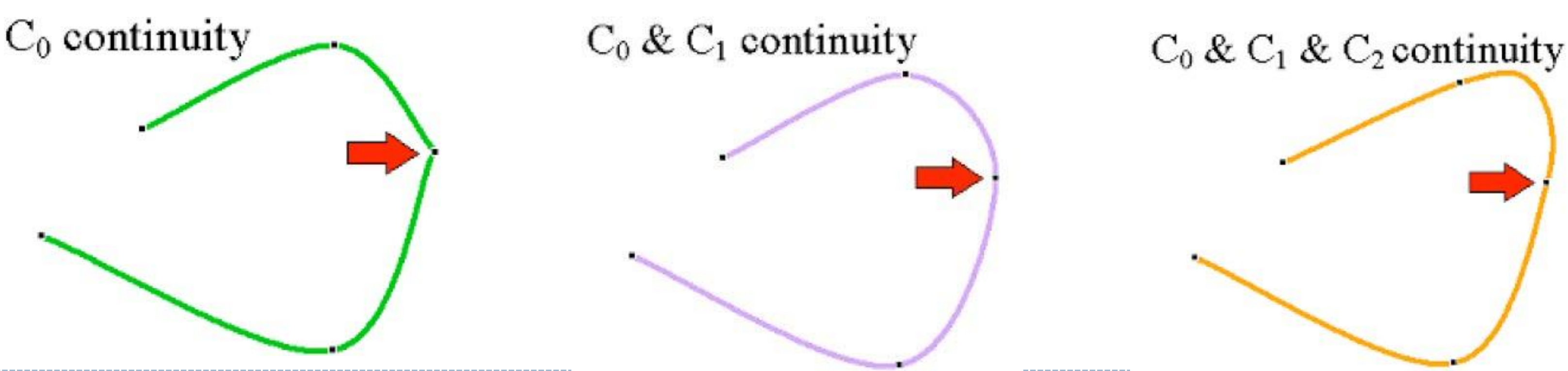
$$\mathbf{x}(u) = \mathbf{x}_i\left(\tfrac{1}{3}u - i\right), \text{ where } i = \left\lfloor \tfrac{1}{3}u \right\rfloor$$



UCSD

Parametric Continuity

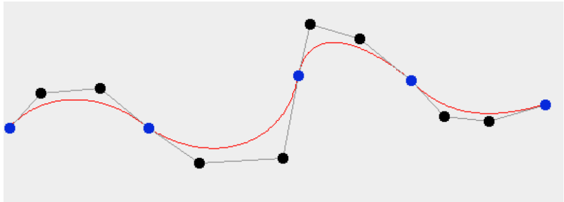
- C^0 continuity:
  - Curve segments are connected
- C^1 continuity:
  - C^0 & 1st-order derivatives agree
  - Curves have same tangents
  - Relevant for smooth shading
- C^2 continuity:
  - C^1 & 2nd-order derivatives agree
  - Curves have same tangents and curvature
  - Relevant for high quality reflections

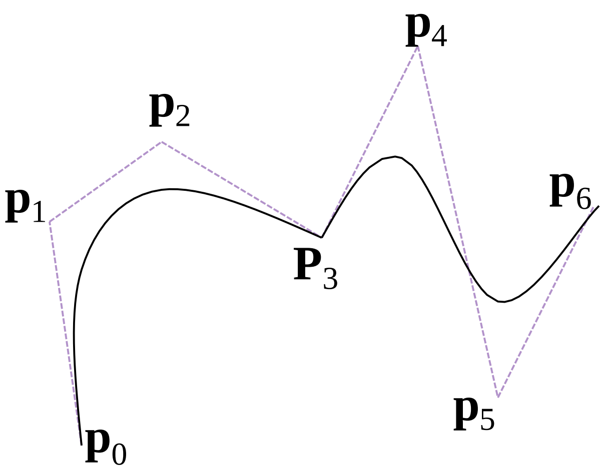


UCSD

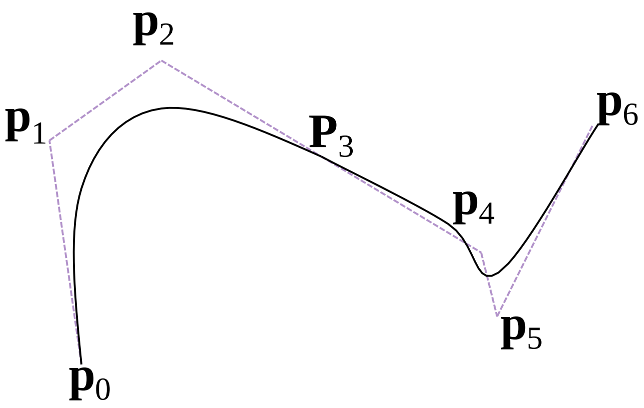
Piecewise Bézier Curve

- $3N+1$ points define N Bézier segments
- $\mathbf{x}(3i)=\mathbf{p}_{3i}$
- C_0 continuous by construction
- C_1 continuous at $\mathbf{p}_{3i}$ when $\mathbf{p}_{3i} - \mathbf{p}_{3i-1} = \mathbf{p}_{3i+1} - \mathbf{p}_{3i}$
- C_2 is harder to achieve and rarely necessary





C_1 discontinuous



C_1 continuous

Recommended Structure

- Use your scene graph code from Project 3, and implement some new Geometry subclasses:

- BezierCurve
  - Has a GetPoint(t) method
  - Should draw N sampled points from the curve (project requires N >= 150)
  - Should also draw its own control points

- Track
  - Contains 8 children BezierCurves
  - Supports keyboard controls for editing control points
  - Should draw control handles: lines through related control points, which are not all owned by any single BezierCurve

More tips

- We can precompute the sampled points inside each BezierCurve, and only update them when that curve is updated.

- Draw lines/points by passing GL_LINE_STRIP/GL_POINTS instead of GL_TRIANGLES to glDrawElements/glDrawArrays
  - see docs – GL_LINE_STRIP draws a line for each adjacent pair, GL_LINES draws a lines for the pairs (0,1), (2,3), …

- A clean way to enforce C1 continuity is to implement more Geometry types
  - Example 1: AnchorPoint and TangentPoint subclasses of Geometry
  - Example 2: ControlHandle subclass of Geometry

Sphere Movement

- We want the sphere to move at a constant velocity *and* stay on the track.
- Pick any point on the track (e.g. a control point) as the initial location. Always keep track of what line segment we're on.
- Calculate the distance to travel in the current frame (frame_distance = velocity * delta_time)
- If traveling this distance keeps the point on the same line segment, we're done.

Sphere Movement

- Otherwise, travel to the end of the current line segment. Subtract the distance traveled from frame_distance. Then move on to the next line segment (which we're now on the initial point of).

- Repeat until frame_distance = 0.

- You also need to handle the case where the sphere moves across different pieces of the track. It's conceptually exactly the same (two adjacent line segments) but requires a bit of extra bookkeeping if you structure your code using BezierCurve objects.