CS 536
Computer Graphics

Bezier Curve Drawing Algorithms
Week 2, Lecture 3

David Breen, William Regli and Maxim Peysakhov
Department of Computer Science
Drexel University
Outline

• Drawing of 2D Curves
  – De Casteljau algorithm
  – Subdivision algorithm
  – Drawing parametric curves
The de Casteljau Algorithm

• How to compute a sequence of points that approximates a smooth curve given a set of control points?
• Developed by Paul de Casteljau at Citroën in the late 1950s
• Idea: recursively subdivide the curve and add points to refine the number of control points
Recall: Linear Interpolation

• Simple example
  – interpolating along the line between two points
  – (really an affine combination of points a and b)
  – \( x(t) = a + (b-a)t \)
Properties of Piecewise Linear Interpolations

• Given
  – continuous curve, C
  – piecewise linear interpolant (PLI) of C
  – and an arbitrary plane, P

• Then:
The number of crossings of P by PLI is no greater than those of C
Linear Interpolation: Example 1

- Constructing a parabola using three control points
- From analytic geometry

\[
\text{ratio}(u, v, w) = \frac{v - u}{w - u}
\]

\[
\text{ratio}(b_0, b_0^1, b_1) = \text{ratio}(b_1, b_1^1, b_2) = \text{ratio}(b_0^1, b_0^2, b_1^1) = t
\]
The de Casteljau Algorithm

Basic case, with two points:

- Plotting a curve via *repeated linear interpolation*
  - Given \( \langle p_0, p_1, \ldots \rangle \)
    a sequence of control points
  - Simple case: Mapping a parameter \( u \) to the line \( \overline{p_0, p_1} \)

\[
p(u) = (1 - u)p_0 + up_1 \quad \text{for } 0 \leq u \leq 1.
\]
The de Casteljau Algorithm

- Generalizing to three points
  - Interpolate \( \overline{p_0p_1} \) and \( \overline{p_1p_2} \)
  - Interpolate along the resulting points

\[
\begin{align*}
p_{01}(u) &= (1-u)p_0 + up_1 \\
p_{11}(u) &= (1-u)p_1 + up_2.
\end{align*}
\]
The de Casteljau Algorithm

• The complete solution from the algorithm for three iterations:

\[
p_{01}(u) = (1-u)p_0 + up_1
\]
\[
p_{11}(u) = (1-u)p_1 + up_2.
\]
\[
p(u) = (1-u)p_{01}(u) + up_{11}(u)
\]
The de Casteljau Algorithm

- The solution after four iterations:
The de Casteljau Algorithm

- **Input:** \( p_0, p_1, p_2 \ldots p_n \in \mathbb{R}^3 \), \( t \in \mathbb{R} \)
- **Iteratively set:**
  \[
  p_{ir}(t) = (1 - t)p_{i(r-1)}(t) + t \ p_{(i+1)(r-1)}(t)
  \]
  \[
  \text{and } p_{i0}(t) = p_i
  \]
  \[
  \begin{cases}
  r = 1, \ldots, n \\
  i = 0, \ldots, n - r
  \end{cases}
  \]

Then \( p_{0n}(t) \) is the point with parameter value \( t \) on the Bézier curve defined by the \( p_i \)'s
The de Casteljau Algorithm: Example Results

• Quartic curve (degree 4)
• 50 points computed on the curve
  – black points
• All intermediate control points shown
  – gray points
The de Casteljau Algorithm: Example Results

- A degree 6 curve
- 60 points computed on the curve
  - the black points
- Intermediate control points
  - the gray points
De Casteljau: Arc Segment Animation
De Casteljau: Cubic Curve Animation
De Casteljau: Loop Curve Animation
The de Casteljau Algorithm: Some Observations

- Interpolation along the curve is based only on \( u \)
- Drawing the curve’s pixels requires iterating over \( u \) at sufficient refinement
- What is the right increment?  
  - It’s not constant!
- Compute points and define a polyline
Subdivision

- Common in many areas of graphics, CAD, CAGD, vision
- Basic idea
  - primitives defined by control polygons
  - set of control points is not unique
    - more than one way to compute a curve
  - subdivision refines representation of an object by introducing more control points
- Allows for local modification
- Subdivide to pixel resolution
Bézier Curve Subdivision

• Subdivision allows display of curves at different/adaptive levels of resolution
• Rendering systems (OpenGL, ActiveX, etc) only display polygons or lines
• Subdivision generates the lines/facets that approximate the curve/surface
  – output of subdivision sent to renderer
Bézier Curve Subdivision, with de Casteljau

- Calculate the value of $x(u)$ at $u = 1/2$
- This creates a new control point for subdividing the curve
- Use the two new edges to form control polygon for two new Bezier curves
Bézier Curve Subdivision

- Observe subdivision:
  - does not affect the shape of the curve
  - partitions one curve into several curved pieces with (collectively) the same shape
Drawing Parametric Curves

Two basic ways:

• *Iterative evaluation* of $x(t)$, $y(t)$, $z(t)$ for incrementally spaced values of $t$
  – can’t easily control segment lengths and error

• *Recursive Subdivision*
  via de Casteljau, that stops when control points get sufficiently close to the curve
  – i.e. when the curve is nearly a straight line

• Use Bresenham to draw each line segment
Drawing Parametric Curves via Recursive Subdivision

- Idea: stop subdivision when segment is flat enough to be drawn with a straight line
- **Curve Flatness Test:**
  - based on the convex hull
  - if $d_2$ and $d_3$ are both less than some $\varepsilon$, then the curve is declared flat
FYI: Computing the Distance from a Point to a Line

- Line is defined with two points
- Basic idea:
  - Project point $P$ onto the line
  - Find the location of the projection

$$d(P, L) = \frac{(y_0 - y_1)x + (x_1 - x_0)y + (x_0y_1 - x_1y_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}$$
Drawing Parametric Curves via Recursive Subdivision

The Algorithm:

- \text{DrawCurveRecSub}(\text{curve}, e)
  - If \text{straight}(\text{curve}, e) then \text{DrawLine}(\text{curve})
  - Else
    - \text{SubdivideCurve}(\text{curve}, \text{LeftCurve}, \text{RightCurve})
    - \text{DrawCurveRecSub}(\text{LeftCurve}, e)
    - \text{DrawCurveRecSub}(\text{RightCurve}, e)
Subdivision: Wave Curve

Animated by Max Peysakhov @ Drexel University
Bézier Curve: Degree Elevation

- Given a control polygon
- Generate additional control points, i.e. increase the degree of the curve
- Keep the curve the same
- In the limit, this converges to the curve defined by the original control polygon
Bezzer Curve Drawing

• Given control points you can either …
  – Iterate through $t$ and evaluate formula
  – Iterate through $t$ and use de Casteljau Algorithm
    • Successive interpolation of control polygon edges
  – Recursively subdivide de Casteljau polygons until they are approximately flat
  – Generate more control points with degree elevation until control polygon approximates curve
General Form of Bezier Curve

\[ Q(u) = \sum_{i=0}^{k} P_{i+1} \binom{k}{i} (1 - u)^{k-i} u^i \]

Control points: \( P_1, P_2, \ldots, P_{k+1}; \quad 0 \leq u \leq 1 \)

Produces a point on curve \( Q \) at parameter value \( u \)
Programming Assignment 1

- Process command-line arguments
- Read in 3D control points
- Iterate through parameter space by du
- At each u value evaluate Bezier curve formula to produce a sequence of 3D points
- Output points by printing them to the console as a polyline and control points as spheres in Open Inventor format