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

Pics/Math courtesy of G. Farin @ ASU
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 \( p_0, p_1 \)

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

\[
p_{ir}(t) = (1 - t)p_{i(r-1)}(t) + t \ p_{(i+1)(r-1)}(t) \quad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n - r \end{cases}
\]

and \( p_{i0}(t) = p_i \)

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

Pics/Math courtesy of G. Farin @ ASU
De Casteljau: Arc Segment 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!
Subdivision

- Common in many areas of graphics, CAD, CAGD, vision
- Basic idea
  - primitives def’d 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

Pics/Math courtesy of G. Farin @ ASU
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

\[
\begin{align*}
&\quad \mathbf{p}_0 \\
&\quad \mathbf{p}_1 \quad \mathbf{p}_{01} \\
&\quad \mathbf{p}_2 \quad \mathbf{p}_{11} \quad \mathbf{p}_{02} \\
&\quad \mathbf{p}_3 \quad \mathbf{p}_{21} \quad \mathbf{p}_{12} \quad \mathbf{p}(u)
\end{align*}
\]
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

Pics/Math courtesy of Dave Mount @ UMD-CP
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 w/ 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:

- DrawCurveRecSub(curve, e)
  - If straight(curve, e) then DrawLine(curve)
  - Else
    - SubdivideCurve(curve, LeftCurve, RightCurve)
    - DrawCurveRecSub(LeftCurve, e)
    - DrawCurveRecSub(RightCurve, e)
Subdivision: Wave Curve
Bézier Curve: Degree Elevation

- Given a control polygon
- Generate additional control points
- Keep the curve the same
- In the limit, this converges to the curve defined by the original control polygon
Bézier Curve: Degree Elevation

- Start with a curve with n+1 control points
  - $P_0, P_1, P_2, \ldots, P_n$
- Compute n+2 control points

\[
Q_i = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i \quad 1 \leq i \leq n
\]

- $Q_0 = P_0 \quad Q_{n+1} = P_n$
Bézier Curve: Degree Elevation

\[
\begin{align*}
Q_1 &= \frac{1}{n+1} P_0 + \left( 1 - \frac{1}{n+1} \right) P_1 \\
Q_2 &= \frac{2}{n+1} P_1 + \left( 1 - \frac{2}{n+1} \right) P_2 \\
Q_3 &= \frac{3}{n+1} P_2 + \left( 1 - \frac{3}{n+1} \right) P_3 \\
&\vdots \\
Q_{n-1} &= \frac{n-1}{n+1} P_{n-2} + \left( 1 - \frac{n-1}{n+1} \right) P_{n-1} \\
Q_n &= \frac{n}{n+1} P_{n-1} + \left( 1 - \frac{n}{n+1} \right) P_n
\end{align*}
\]
Bézier Curve: Degree Elevation
Beziers 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
Programming Assignment 3

• Input PostScript-like file.
• Output B/W PBM.
• Implement viewports.
• Use HW2 for polygon clipping.
• Implement scanline polygon filling. (*You can* not use flood filling algorithms*)
Extra Credit Assignment

- Input Curveto PostScript-like file
  - Read in control points for a number of curves
- Support HW3 options
- Interpret points as the control points of a cubic Bezier curve
- Evaluate curve to produce an approximating polyline
- Draw lines into frame buffer
- Output B/W PBM