MCS 585/480
Computer Graphics I

Curve Drawing Algorithms
Week 4, Lecture 8

William Regli and Maxim Peysakhov
Geometric and Intelligent Computing Laboratory
Department of Computer Science
Drexel University
http://gicl.mcs.drexel.edu

Outline

• Drawing of 2D Curves
  – De Casteljau algorithm
  – Subdivision algorithm
  – Drawing parametric curves
• Introduction and discussion of homework #4

The de Casteljau Algorithm

• How do define a smooth curve that approximates a sequence 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

Historic Analogy: The Universe of Aristotle and Ptolemy

• Earth is in the center
• Planetary motion described as Epicycles
• For some planets Epicycles had to be placed on Epicycles
• In the similar manner we will have linear interpolation of the linear interpolating points

Recall: Linear Interpolation

• Simple example
  – Interpolating along the line between two points
  – (really an affine combination of points a and b)

Properties of Piecewise Linear Interpolations

• Given
  – continuous curve, C
  – piecewise linear interpolant of C, PLC
  – and an arbitrary plane, P
• Then:
  The number of crossings of P by PLC is no greater than those of C
Linear Interpolation: Example 1
- Constructing a parabola using three control points
- Special case of the three tangent theorem from analytic geometry
  \[ \text{ratio} \left( b_i, b_i \right) = \frac{\text{ratio}(b_i, b_i)}{\text{ratio}(b_i, b_i)} = \frac{f_i - 1}{f_i - 1} \]

Linear Interpolation: Example 2
- Constructing a Bézier curve with four control points
- Point \( b_i(t) \) is obtained by repeated linear interpolation
- Shown: cubic case, \( n=3 \) and \( t=1/3 \)

The de Casteljau Algorithm
- Basic case, with two points:
  - Plotting a curve via repeated linear interpolation
  - Given \( p_0, p_1 \), 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 \( p_0p_1 \) and \( p_1p_2 \)
  - Interpolate along the resulting points
    \[ p_u(u) = (1-u)p_0 + up_1 \]
    \[ p_u(u) = (1-u)p_2 + up_2 \]

The de Casteljau Algorithm
- The complete solution from the algorithm for three iterations:
  \[ p_0(u) = (1-u)p_0 + up_1 \]
  \[ p_1(u) = (1-u)p_1 + up_2 \]
  \[ p_2(u) = (1-u)p_2 + up_3 \]
- The solution after four iterations:
The de Casteljau Algorithm

- Input: \( b_0, b_1, b_2, \ldots, b_n \in \mathbb{R}^k, t \in \mathbb{R} \)
- Iteratively set:
  \[
  b'_i(t) = (1-t)b_{i+1} + t \sum_{j=0}^{i-1} \binom{n-i}{j} b_{i+j+1} \quad \text{for } i = 0, \ldots, n-1
  \]
  and \( b'_n(t) = b_n \)

Then \( b'_n(t) \) is the point with parameter value \( t \) on the Bézier curve \( b^n \).
Hence: \( b^n(t) = b'_n(t) \)

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

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
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

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
- Lots of motivation for this op.

Bézier Curve Subdivision

- Subdivision allows display of curves at different 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, avec de Casteljau

- Calculate the value of $x(u)$ at $u = \frac{1}{2}$
- This creates a new control point for subdividing the curve
- Use the two new edges to form a triangle for two new Bézier curves
Bézier Curve Subdivision (degree elevation)

- Given a control polygon
- Generate successive refinements of the control polygon
- In the limit, this converges to the uniform B-spline curve defined by the original control polygon

Subdivision: Arc Segment

Subdivision: Cubic Curve

Subdivision: Wave Curve

Subdivision: Loop Curve

Drawing Parametric Curves

Two basic ways:

- **Iterative evaluation** of $x(t)$, $y(t)$, $z(t)$ for incrementally spaced values of $t$
  - a la Bresenham, but for curves

- **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

\[ d(P, L) = \frac{\left| (y_2 - y_1)x + (x_2 - x_1)y + (x_1y_2 - x_2y_1) \right|}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}} \]

**Programming Assignment 4**

- Output B/W XPM, input PostScript like file.
- Create data structure to hold polygons.
- Implement translation, rotation and scaling.
- Implement polygon clipping.
- Implement scanline polygon filling. *(You can not use flood filling algorithms)*