Computer Graphics
CS 543 Lecture 12 (Part 1)
Curves

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
So Far...

- Dealt with straight lines and flat surfaces
- Real world objects include curves
- Need to develop:
  - Representations of curves
  - Tools to render curves
Curve Representation: Explicit

- One variable expressed in terms of another
- Example:

\[ z = f(x, y) \]

- Works if one x-value for each y value
- Example: does not work for a sphere

\[ z = \sqrt{x^2 + y^2} \]

- Rarely used in CG because of this limitation
Curve Representation: Implicit

- Represent 2D curve or 3D surface as zeros of a formula
- Example: sphere representation
  \[ x^2 + y^2 + z^2 - 1 = 0 \]
- May limit classes of functions used
- Polynomial: function which can be expressed as linear combination of integer powers of x, y, z
- Degree of algebraic function: highest power in function
- Example: \( mx^4 \) has degree of 4
Curve Representation: Parametric

- Represent 2D curve as 2 functions, 1 parameter

  \[(x(u), y(u))\]

- 3D surface as 3 functions, 2 parameters

  \[(x(u, v), y(u, v), z(u, v))\]

- Example: parametric sphere

  \[x(\theta, \phi) = \cos \phi \cos \theta\]
  \[y(\theta, \phi) = \cos \phi \sin \theta\]
  \[z(\theta, \phi) = \sin \phi\]
Choosing Representations

- Different representation suitable for different applications

- Implicit representations good for:
  - Computing ray intersection with surface
  - Determining if point is inside/outside a surface

- Parametric representation good for:
  - Breaking surface into small polygonal elements for rendering
  - Subdivide into smaller patches

- Sometimes possible to convert one representation into another
Continuity

- Consider parametric curve
  \[ P(u) = (x(u), y(u), z(u))^T \]

- We would like smoothest curves possible
- Mathematically express smoothness as continuity (no jumps)

**Defn:** if kth derivatives exist, and are continuous, curve has kth order parametric continuity denoted \( C^k \)
Continuity

- $0^{th}$ order means curve is continuous
- $1^{st}$ order means curve tangent vectors vary continuously, etc
Interactive Curve Design

- Mathematical formula unsuitable for designers
- Prefer to interactively give sequence of points (control points)
- Write procedure:
  - Input: sequence of points
  - Output: parametric representation of curve
Interactive Curve Design

- 1 approach: curves pass through control points (interpolate)
- **Example**: Lagrangian Interpolating Polynomial
- Difficulty with this approach:
  - Polynomials always have “wiggles”
  - For straight lines wiggling is a problem
- Our approach: approximate control points (Bezier, B-Splines)
De Casteljau Algorithm

- Consider smooth curve that approximates sequence of control points \([p_0, p_1, \ldots]\)

\[
p(u) = (1 - u)p_0 + up_1 \quad 0 \leq u \leq 1
\]

- Blending functions: \(u\) and \((1 - u)\) are non-negative and sum to one
De Casteljau Algorithm

- Now consider 3 points
- 2 line segments, P0 to P1 and P1 to P2

\[
p_{01}(u) = (1-u)p_0 + up_1 \quad \quad p_{11}(u) = (1-u)p_1 + up_2
\]
De Casteljau Algorithm

Substituting known values of $p_{01}(u)$ and $p_{11}(u)$

\[ p(u) = (1-u)p_{01} + up_{11}(u) \]

\[ = (1-u)^2p_0 + (2u(1-u))p_1 + u^2p_2 \]

\[ b_{02}(u) \quad b_{12}(u) \quad b_{22}(u) \]

Blending functions for degree 2 Bezier curve

\[ b_{02}(u) = (1-u)^2 \quad b_{12}(u) = 2u(1-u) \quad b_{22}(u) = u^2 \]

\textbf{Note:} blending functions, non-negative, sum to 1
De Casteljau Algorithm

- Extend to 4 control points $P_0, P_1, P_2, P_3$

$$p(u) = (1-u)^3 p_0 + (3u(1-u)^2) p_1 + (3u^2(1-u)) p_2 + u^3$$

- Final result above is Bezier curve of degree 3
De Casteljau Algorithm

\[ p(u) = (1-u)^3 p_0 + (3u(1-u)^2) p_1 + (3u^2(1-u)) p_2 + u^3 \]

- Blending functions are polynomial functions called Bernstein’s polynomials

\[ b_{03}(u) = (1-u)^3 \]
\[ b_{13}(u) = 3u(1-u)^2 \]
\[ b_{23}(u) = 3u^2(1-u) \]
\[ b_{33}(u) = u^3 \]
De Casteljau Algorithm

\[ p(u) = (1-u)^3 p_0 + (3u(1-u)^2) p_1 + (3u^2(1-u)) p_2 + u^3 \]

- Writing coefficient of blending functions gives Pascal’s triangle

\[
\begin{array}{cccc}
1 & & & \\
1 & 1 & & \\
1 & 2 & 1 & \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
\end{array}
\]

3 control points
4 control points
5 control points
De Casteljau Algorithm

- In general, blending function for $k$ Bezier curve has form

\[ b_{ik}(u) = \binom{k}{i} (1-u)^{k-i} u^i \]

- Example

\[ b_{03}(u) = \binom{3}{0} (1-u)^{3-0} u^0 = (1-u)^3 \]
De Casteljau Algorithm

- Can express cubic parametric curve in matrix form

\[ p(u) = [1, u, u^2, u^3] M_B \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \]

where

\[ M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \]
Subdividing Bezier Curves

- OpenGL renders flat objects
- To render curves, approximate with small linear segments
- Subdivide surface to polygonal patches
- Bezier curves useful for elegant, recursive subdivision
Subdividing Bezier Curves

- Let \((P0..., P3)\) denote original sequence of control points
- Recursively interpolate with \(u = \frac{1}{2}\) as below
- Sequences \((P00,P01,P02,P03)\) and \((P03,P12,P21,30)\) define Bezier curves also
- Bezier Curves can either be straightened or curved recursively in this way
Beziers Surfaces

- Bezier surfaces: interpolate in two dimensions
- This called Bilinear interpolation
- Example: 4 control points, P00, P01, P10, P11, 2 parameters $u$ and $v$
- Interpolate between
  - P00 and P01 using $u$
  - P10 and P11 using $u$
  - P00 and P10 using $v$
  - P01 and P11 using $v$

$$p(u, v) = (1-v)((1-u)p_{00} + up_{01}) + v((1-u)p_{10} + up_{11})$$
Beziers Surfaces

- Expressing in terms of blending functions

\[ p(u,v) = b_{01}(v)b_{01}(u)p_{00} + b_{01}(v)b_{11}b_{01}(u)p_{01} + b_{11}(v)b_{11}(u)p_{11} \]

Generalizing

\[ p(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_{i,3}(v)b_{j,3}(u)p_{i,j} \]
Problems with Bezier Curves

- Bezier curves are elegant but too many control points
- To achieve smoother curve
  - = more control points
  - = higher order polynomial
  - = more calculations
- **Global support problem:** All blending functions are non-zero for all values of $u$
- All control points contribute to all parts of the curve
- Means after modelling complex surface (e.g. a ship), if one control point is moves, recalculate everything!
B-Splines

- B-splines designed to address Bezier shortcomings
- B-Spline given by blending control points
- **Local support**: Each spline contributes in limited range
- Only non-zero splines contribute in a given range of $u$

\[
p(u) = \sum_{i=0}^{m} B_i(u) p_i
\]

B-spline blending functions, order 2
NURBS

- Encompasses both Bezier curves/surfaces and B-splines
- Non-uniform Rational B-splines (NURBS)
- Rational function is ratio of two polynomials
- Some curves can be expressed as rational functions but not as simple polynomials
- No known exact polynomial for circle
- Rational parametrization of unit circle on xy-plane:

  \[
  x(u) = \frac{1-u^2}{1+u^2} \\
  y(u) = \frac{2u}{1+u^2} \\
  z(u) = 0
  \]
NURBS

- We can apply homogeneous coordinates to bring in $w$

\[
\begin{align*}
  x(u) &= 1 - u^2 \\
  y(u) &= 2u \\
  z(u) &= 0 \\
  w(u) &= 1 + u^2
\end{align*}
\]

- Using $w$, we get we cleanly integrate rational parametrization
- Useful property of NURBS: preserved under transformation
Tessellation

- **Previously**: Pre-generate mesh versions offline
- Tessellation shader unit new to GPU in DirectX 10 (2007)
  - Subdivide faces to yield finer detail, generate new vertices, primitives
- Mesh simplification/tessellation on GPU = Real time LoD
- Tessellation: [Demo](#)
Tessellation Shaders

- Can subdivide curves, surfaces on the GPU

Lines

Triangles

Quads (subsequently broken into triangles)
Where Does Tessellation Shader Fit?

- Fixed number of vertices in/out

- Can change number of vertices
Step 1: Application Code

- Tessellation shader can generate new primitive called a patch
  - User-defined number of vertices per patch
- In application, use `glPatchParameteri` set number of vertices per patch
- Example: 2 patches, each with 4 vertices

```
GLfloat vertices [][] = {
    {-0.75, -0.25}, {-0.25, -0.25}, {-0.25, 0.25}, {-0.75, 0.25},
    { 0.25, -0.25}, { 0.75, -0.25}, { 0.75, 0.25}, { 0.25, 0.25}
};

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glVertexAttribPointer(vPos, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));
glPatchParameteri(GL_PATCH_VERTICES, 4);
glDrawArrays(GL_PATCHES, 0, 8);
```

8 vertices total, 4 vertices per patch
Step 2: Tessellation Control Shader

- Generates output vertices
  - Sometimes pass-through: same no. of vertices as input
- Set how much each patch should be tessellated
  - Outer: how many segments exterior edges broken into
  - Inner: How many inner regions (horizontal & vertical)
Step 3: Tessellation Evaluation Shader

- Positions vertices output from TCS
- A function e.g. curve equation can be used to position vertices
- Teapot made up of tessellated patches
  - Eqn to determine tessellated vertex location from control points

\[ \tilde{p}(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} B(i, u)B(j, v)\vec{v}_{ij} \]
Geometry Shader

- After Tessellation shader. Can
  - Handle whole primitives
  - Generate new primitives
  - Generate no primitives (cull)
References

- Hill and Kelley, chapter 11