So Far…

- Dealt with straight lines and flat surfaces
- Real world objects include curves
- Need to develop:
  - Representations of curves (mathematical)
  - Tools to render curves
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

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

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

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

\[ = (1-u)^2 b_0 + (2u(1-u)) b_1 + u^2 b_2 \]

$b_{02}(u)$  \quad $b_{12}(u)$  \quad $b_{22}(u)$

Blending functions for degree 2 Bezier curve

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

Note: blending functions, non-negative, sum to 1
De Casteljau Algorithm

- Extend to 4 control points P0, P1, P2, P3

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

- 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

\[ b_{03}(u) = (1-u)^3 \]
\[ b_{13}(u) = 3u(1-u)^2 \]
\[ b_{23}(u) = 3u^2(1-u) \]
\[ b_{33}(u) = u^3 \]
Subdividing Bezier Curves

- OpenGL renders flat objects
- To render curves, approximate with small linear segments
- Subdivide surface to polygonal patches
- Bezier Curves can either be straightened or curved recursively in this way
Bezier 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 $v$

\[
p(u, v) = (1 - v)((1 - u)p_{00} + up_{01}) + v((1 - u)p_{10} + up_{11})
\]
Problems with Bezier Curves

- Bezier curves elegant but 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

- Non-uniform Rational B-splines (NURBS)
- Rational function means 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:

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

Previously: Pre-generate mesh versions offline

Tessellation shader unit new to GPU in DirectX 10 (2007)

- Subdivide faces on-the-fly to yield finer detail, generate new vertices, primitives

- Mesh simplification/tessellation on GPU = Real time LoD
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

Diagram:
- Vertex Shader
  - Primitive Assembly
  - Tessellation Control Shader
    - Tessellation Primitive Generator
      - Tessellation Evaluation Shader
      - Primitive Assembly
  - Geometry Shader
    - Primitive Assembly
    - Rasterizer
    - Fragment Shader

Legend:
- Orange = Fixed Function
- Green = Programmable
Geometry Shader

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

- Hill and Kelley, chapter 11