Computer Graphics
CS 543 Lecture 13b
Curves, Tesselation/Geometry Shaders & Level of Detail

Prof Emmanuel Agu

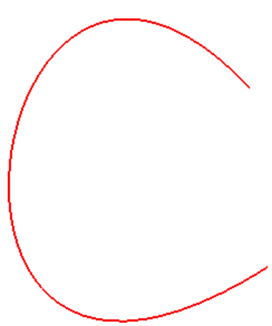
Computer Science Dept.

Worcester Polytechnic Institute (WPI)

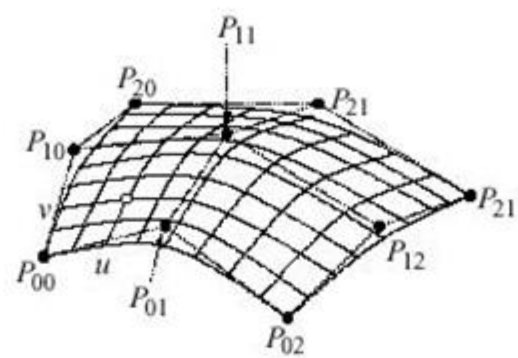
So Far…

Ref: Hill and Kelley, Computer Graphics Using OpenGL (3rd edition), Chapter 10

- Dealt with straight lines and flat surfaces
- But real world objects include curves, curved surfaces
- Need to develop:
  - Representations of curves (curved surfaces)
  - Tools to render curves (curved surfaces)



Curve



Curved Surface

Curve Representation: Explicit

- One variable expressed in terms of another
- Example:

$$z = f(x, y)$$

- Works if one x-value for each y value (unique pair)
- Example: does not work for a sphere (many x,y combinations = z)

$$z = \sqrt{x^2 + y^2}$$

- 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, 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
  - Determing if point is inside/outside a surface
- Parametric representation good for:
  - Dividing surface into small polygonal elements for rendering
  - Subdivide into smaller patches
- Sometimes possible to convert one representation into another

Continuity

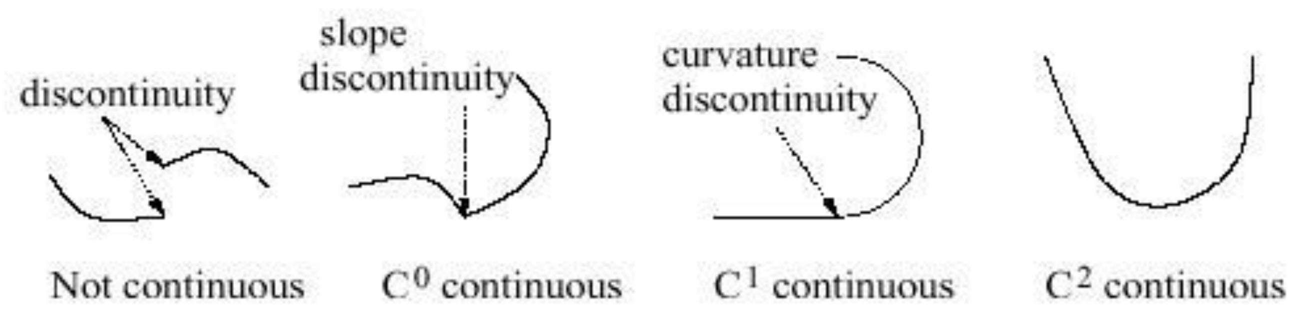
- Consider parametric curve

$$P(u) = (x(u), y(u), z(u))^T$$

- 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

- 0th order means curve is continuous
- 1st order means curve tangent vectors (1st derivative) vary continuously, etc



Interactive Curve Design

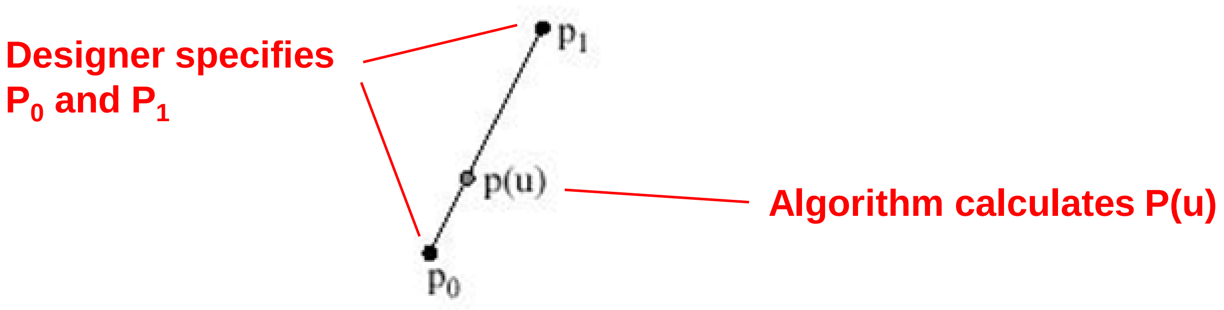
- Mathematical formula unsuitable for designers
- Prefer to interactively give sequence of points (control points)
- Write procedure:
  - **Input:** sequence of points
  - **Output:** parametric representation of curve

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) called **De Casteljau’s algorithm**

De Casteljau Algorithm

- Consider smooth curve that approximates sequence of control points [p0,p1,….]

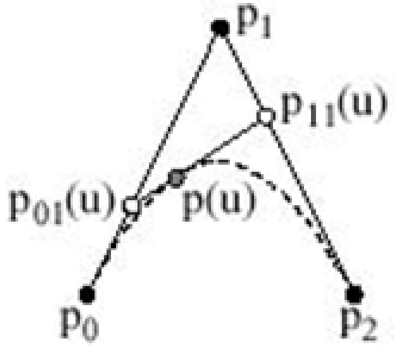


$$p(u) = (1-u)p_0 + up_1 \qquad 0 \leq u \leq 1$$

- Blending functions: *u* and *(1 – u)* are non-negative and sum to one

De Casteljau Algorithm

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



- Algorithm first calculates P_{01} and P_{11}

$$p_{01}(u) = (1-u)\,p_0 + up_1 \qquad p_{11}(u) = (1-u)\,p_1 + up_2$$

De Casteljau Algorithm

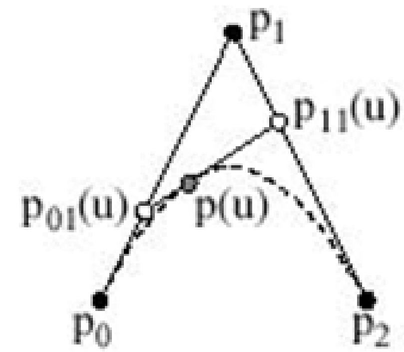
Then calculates P(u) by

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

$$p(u) = (1-u)\,p_{01} + u p_{11}(u)$$

$$= (1-u)^2\,p_0 + (2u(1-u))\,p_1 + u^2\,p_2$$

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



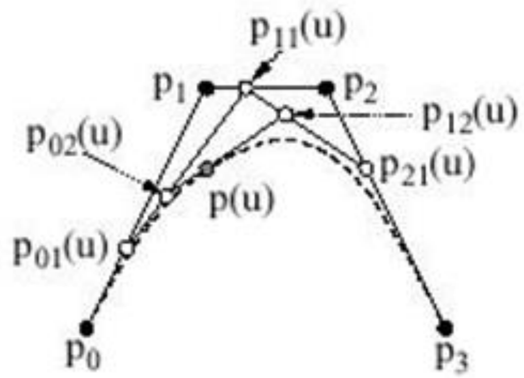
Blending functions for degree 2 Bezier curve

$$b_{02}(u) = (1-u)^2 \qquad b_{12}(u) = 2u(1-u) \qquad b_{22}(u) = u^2$$

Note: blending functions, non-negative, sum to 1

De Casteljau Algorithm

- Similarly, extend to 4 control points P0, P1, P2, P3



$$p(u) = (1-u)^3 \boxed{p_0} + (3u(1-u)^2)\boxed{p_1} + (3u^2(1-u))\boxed{p_2} + u^3$$

$b_{03}(u)$ $\quad$ $b_{13}(u)$ $\quad$ $b_{23}(u)$ $\quad$ $b_{33}(u)$

- Final result above is Bezier curve of degree 3

De Casteljau Algorithm

$$p(u) = (1-u)^3 \boxed{p_0} + (3u(1-u)^2)\boxed{p_1} + (3u^2(1-u))\boxed{p_2} + u^3$$

$b_{03}(u)$ $b_{13}(u)$ $b_{23}(u)$ $b_{33}(u)$

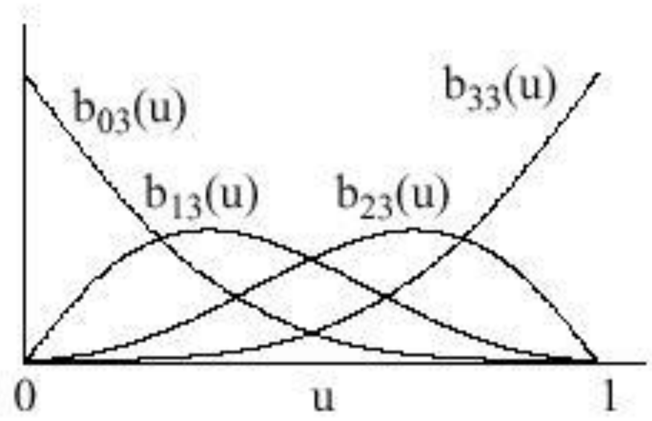
- Blending functions are polynomial functions called **Bernstein's polynomials**

$$b_{03}(u) = (1-u)^3$$

$$b_{13}(u) = 3u(1-u)^2$$

$$b_{23}(u) = 3u^2(1-u)$$

$$b_{33}(u) = u^3$$



De Casteljau Algorithm

$$p(u) = (1-u)^3 \boxed{p_0} + (3u(1-u)^2)\boxed{p_1} + (3u^2(1-u))\boxed{p_2} + u^3$$

1 3 3 1

- Coefficients of blending functions gives Pascal's triangle

1

1 1

1 2 1 ← 3 control points

1 3 3 1 ← 4 control points

1 4 6 4 1 ← 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

$$p(u) = (1-u)^3 \boxed{p_0} + (3u(1-u)^2)\boxed{p_1} + (3u^2(1-u))\boxed{p_2} + u^3$$

$b_{03}(u)$ $b_{13}(u)$ $b_{23}(u)$ $b_{33}(u)$

- Blending function b_{03} can be represented using (i = 0, k = 3)

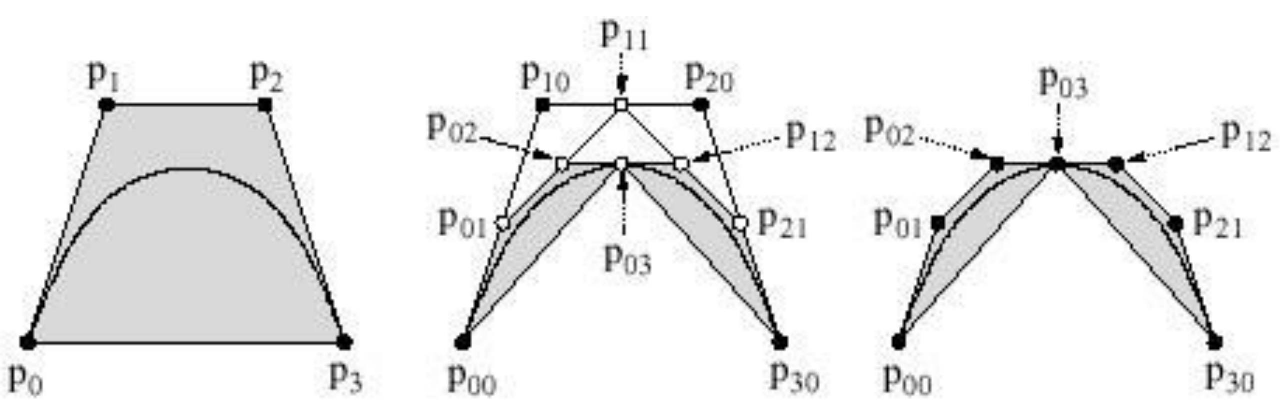
$$b_{03}(u) = \binom{3}{0}(1-u)^{3-0}u^{0} = (1-u)^3$$

Subdividing Bezier Curves

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

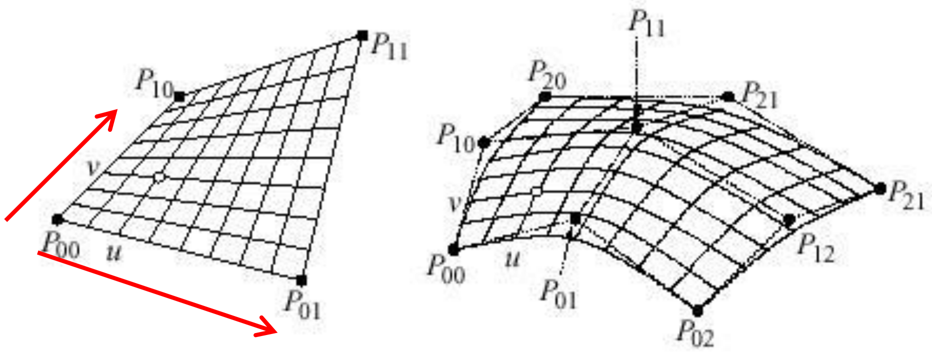
Subdividing Bezier Curves

- Let (P0… P3) denote original sequence of control points
- Recursively interpolate with u = ½ 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



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



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

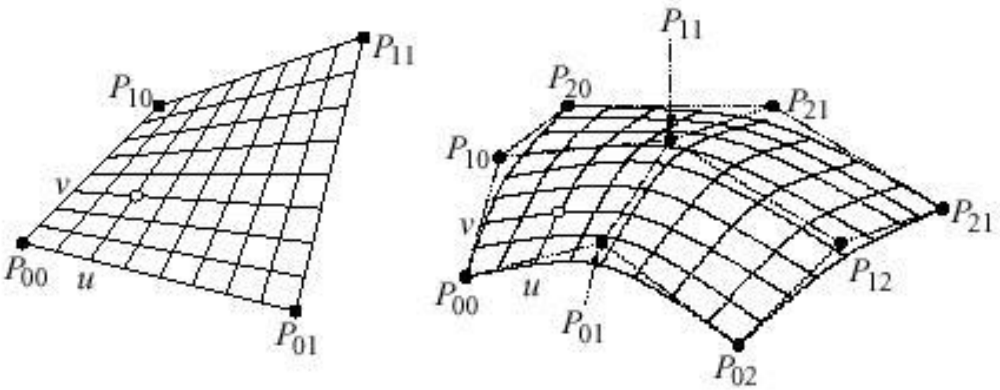
Bezier Surfaces

- Expressing in terms of blending functions

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

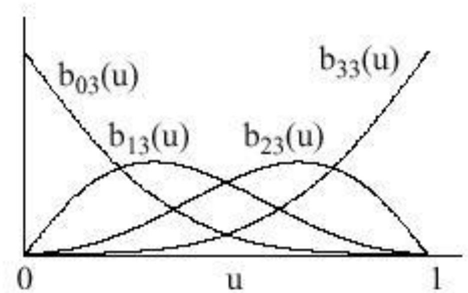
Generalizing

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_{i,3}(v)b_{j,3}(u)\,p_{i,j}$$



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



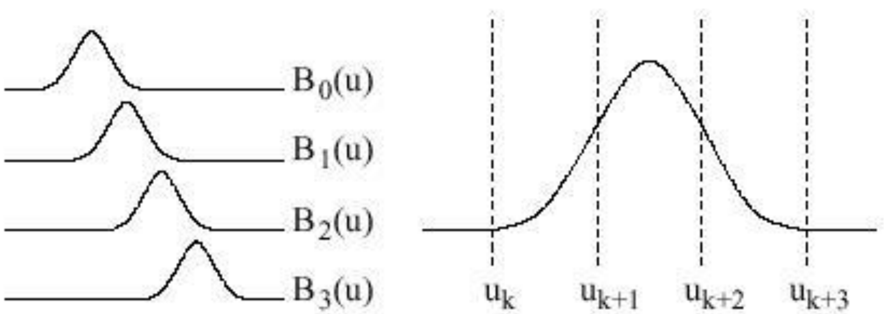
- **Global support problem:** All blending functions are non-zero for all values of *u*
- All control points contribute to all parts of the curve
- Means after modelling complex surface (e.g. a ship), if one control point is moved, must recalculate everything!

B-Splines

- B-splines designed to address Bezier shortcomings
- B-Spline given by blending control points

$$p(u) = \sum_{i=0}^{m} B_i(u)\, p_i$$

- **Local support:** Each spline contributes in limited range of *u*
- Only non-zero splines contribute in a given range of *u*



B-spline blending functions, order 2

Non-Uniform Rational B-Splines (NURBS)

- Encompasses both Bezier curves/surfaces and B-splines
- A rational function is ratio of two polynomials
- Some curves can be expressed as rational functions but not as simple polynomials
- E.g. No known exact polynomial for circle
- Rational form 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*

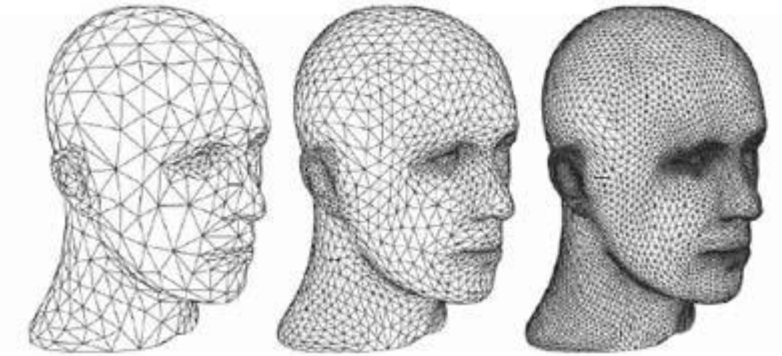
$$x(u) = 1 - u^2$$
$$y(u) = 2u$$
$$z(u) = 0$$
$$w(u) = 1 + u^2$$

- Useful property of NURBS: preserved under transformation
  - E.g. Rotate sphere defined as NURBS, after rotation still a sphere

Tesselation

tesselation →

Far = Less detailed mesh



[Zorin and Schröder, 2000]

Near = More detailed mesh

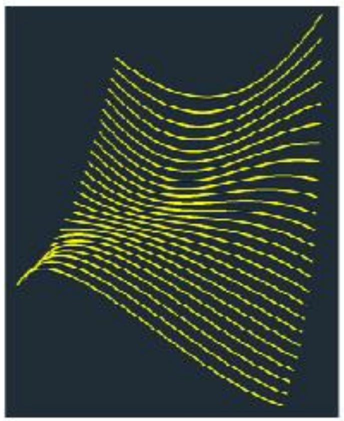
← Simplification

- **Previously:** Had to pre-generate mesh versions offline
- Tesselation shader unit new to GPU in DirectX 10 (2007)
  - Subdivide faces to yield finer detail, generate new vertices, primitives
- Mesh simplification/tesselation on GPU = Real time LoD
- Tesselation: Demo

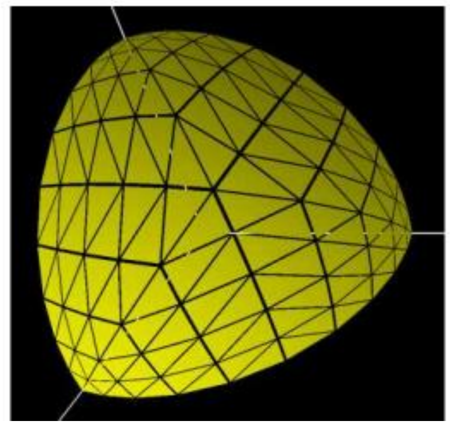
Tessellation Shaders

- Operates on/sub-divides **primitives (Lines, triangles, quads)**
- Can subdivide curves, surfaces on the GPU

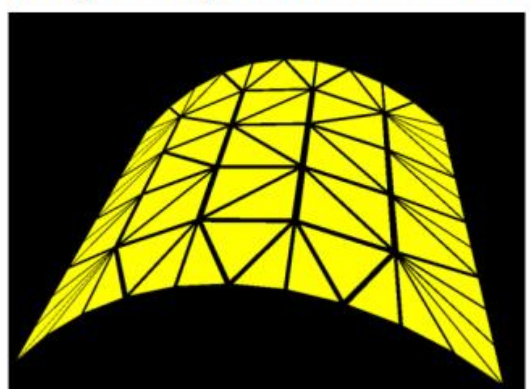
Lines



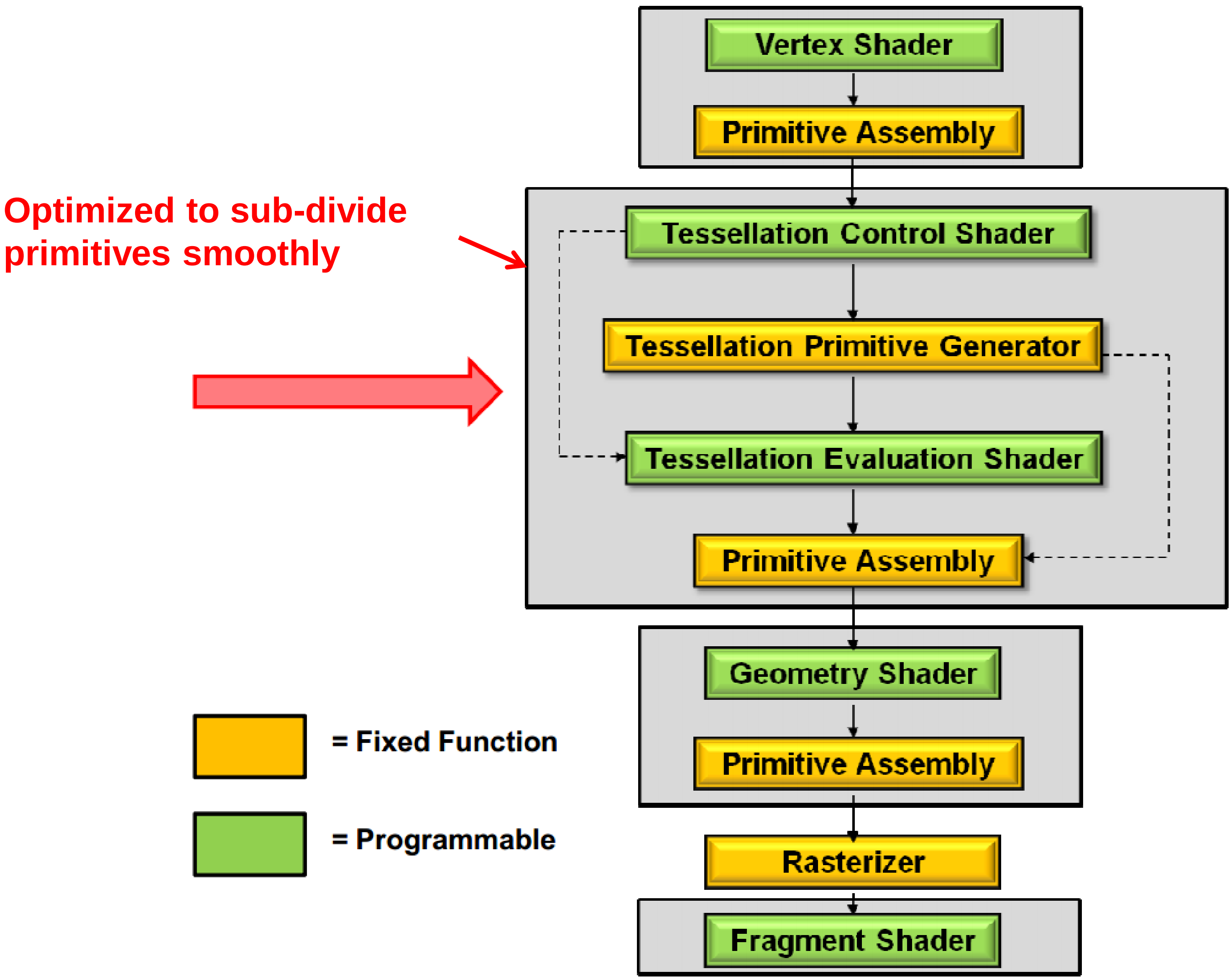
Triangles



Quads (subsequently broken into triangles)

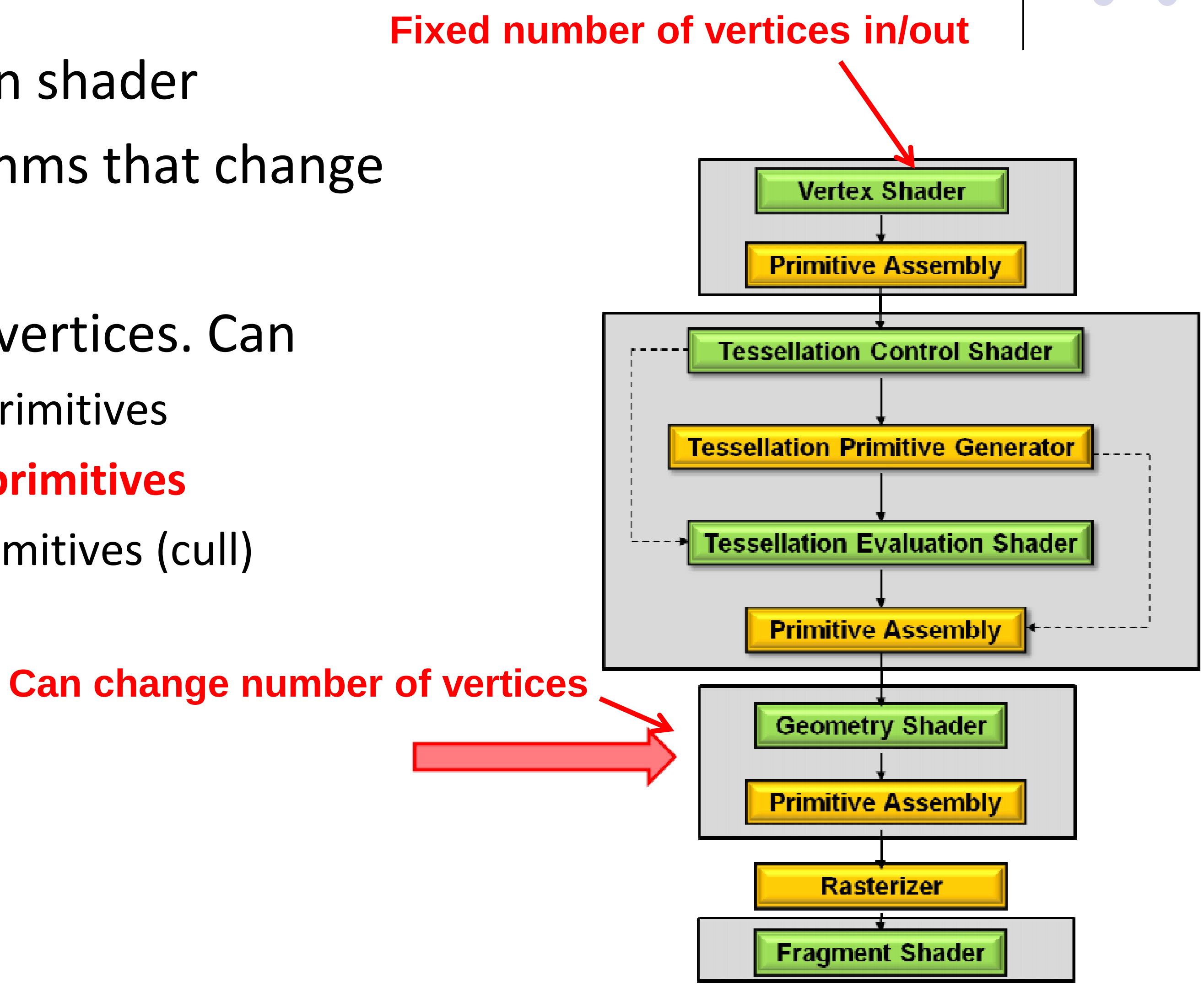


Where Does Tesselation Shader Fit?



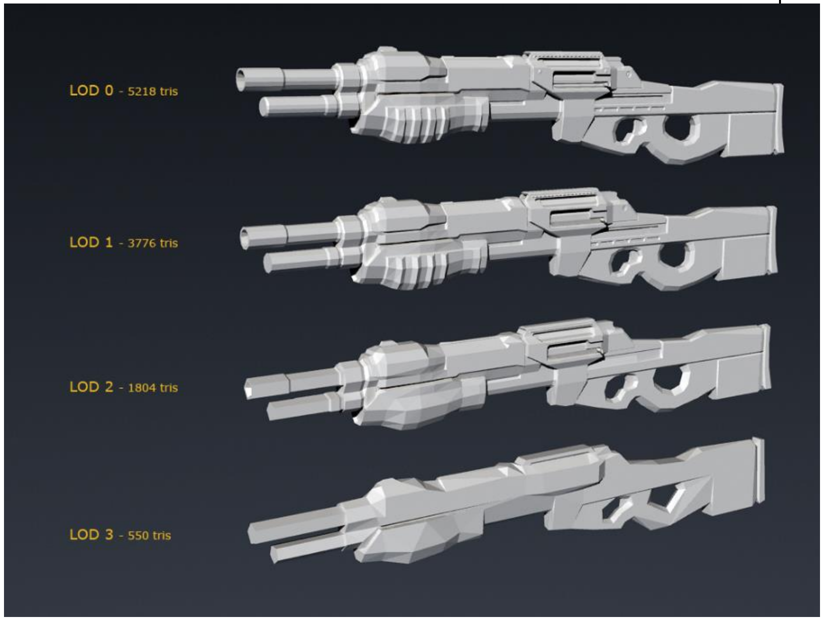
Geometry Shader

- After Tesselation shader
- Used for algorithms that change no. of vertices
- Modifies no. of vertices. Can
  - Handle whole primitives
  - Generate **new primitives**
  - Generate no primitives (cull)



Level of Detail (LoD)

- Use simpler versions of objects if they make smaller contributions to the image

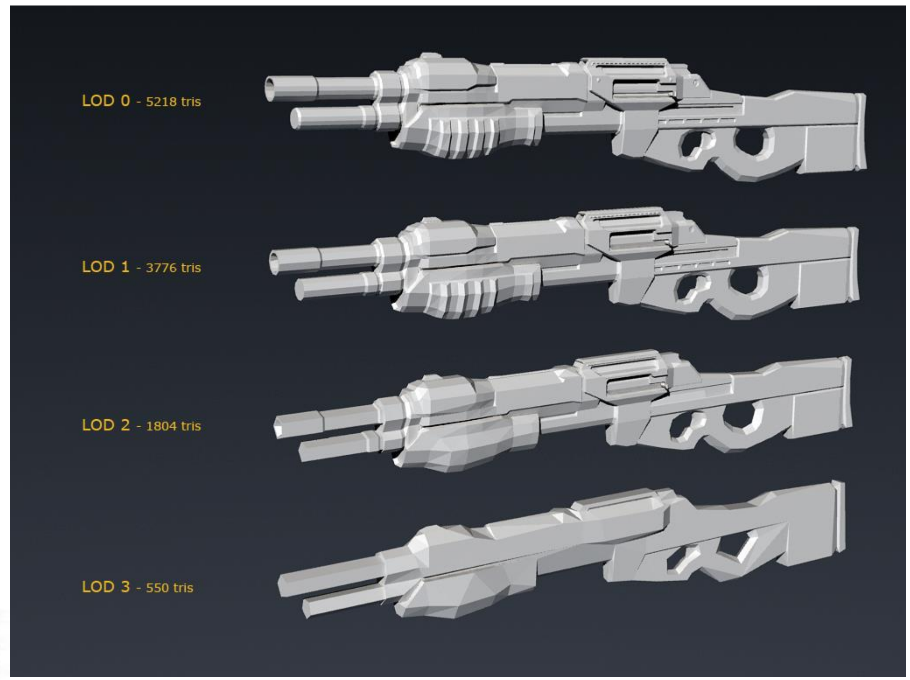


- LOD algorithms have three parts:
  - Generation: Models of different details are generated
  - Selection: Chooses which model should be used depending on criteria
  - Switching: Changing from one model to another
- Can be used for models, textures, shading and more

Level of Detail (LoD)

1.5 million triangles **1100 triangles**

Figure 14.21. On the left, the original model consists of 1.5 million triangles. On the right, the model has 1100 triangles, with surface details stored as heightfield textures and rendered using relief mapping. *(Image courtesy of Natalya Tatarchuk, ATI Research, Inc.)*

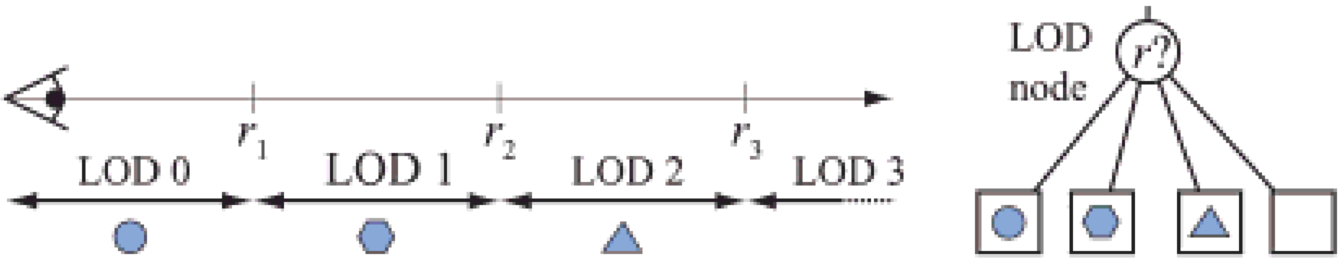


LOD Switching

- Discrete Geometry LODs
  - LOD is switched **suddenly** from one frame to the next
- Blend LODs
  - Two LODs are blended together over time (several frames)
    - Fade out LoD 1 by decreasing alpha value (1 to 0)
    - Fade in new LoD 2 by increasing alpha value (0 to 1)
  - More expensive than rendering one LOD
- Alpha LOD: Object’s alpha value decreased as distance increases

LOD Selection

- Determining which LOD to render and which to blend
- Range-Based (depending on object distance):
  - LOD choice based on distance



Time-Critical LOD Rendering

- Using LOD to ensure constant frame rates
- Select LoD of scene that hardware can render at 25 FPS

- Predictive algorithm
  - Selects the LOD based on which objects are visible

- Heuristics:
  - Maximize $$\sum_{S} \text{Benefit}(O, L)$$
  - Constraint: $$\sum_{S} \text{Cost}(O, L) \leq \text{TargetFrameTime}.$$

References

- Hill and Kelley, chapter 11
- Angel and Shreiner, Interactive Computer Graphics, 6th edition, Chapter 10
- Shreiner, OpenGL Programming Guide, 8th edition