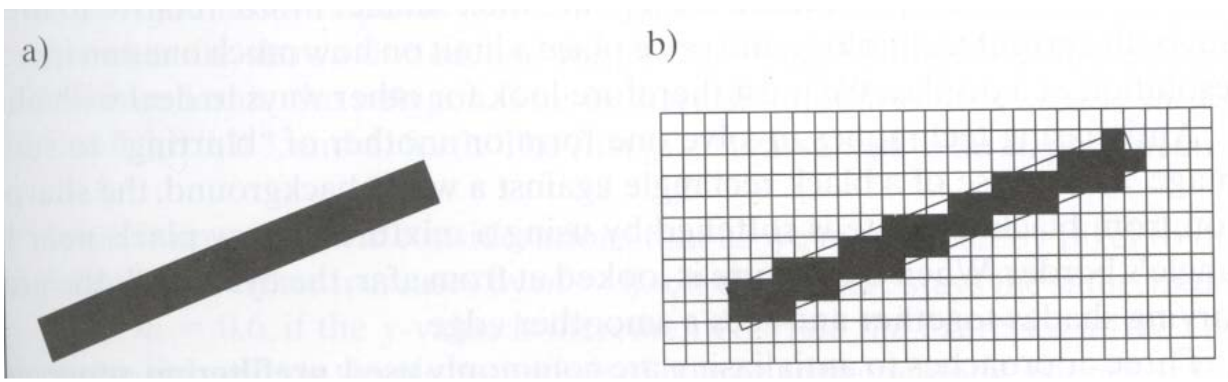
Recall: Antialiasing

- Raster displays have pixels as rectangles
- Aliasing: Discrete nature of pixels introduces "jaggies"



Recall: Antialiasing

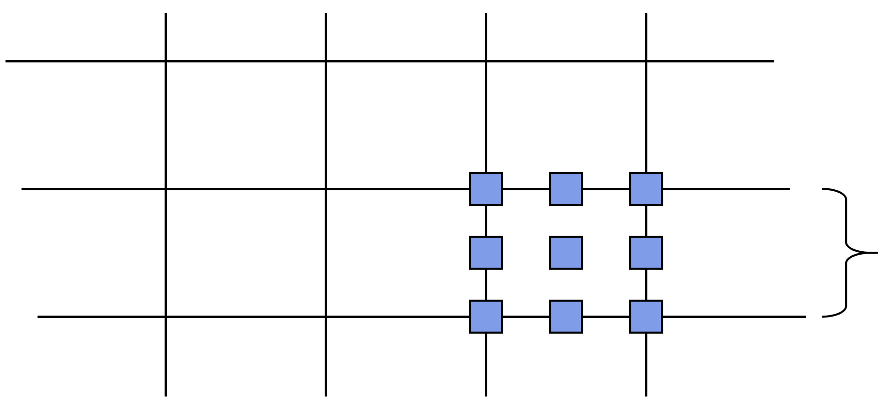
- Aliasing effects:
  - Distant objects may disappear entirely
  - Objects can blink on and off in animations
- Antialiasing techniques involve some form of blurring to reduce contrast, smoothen image
- Three antialiasing techniques:
  - Prefiltering
  - Postfiltering
  - Supersampling

Prefiltering

- Basic idea:
  - compute area of polygon coverage
  - use proportional intensity value
- Example: if polygon covers ¼ of the pixel
  - Pixel color = ¼ polygon color + ¾ adjacent region color
- Cons: computing polygon coverage can be time consuming

Supersampling

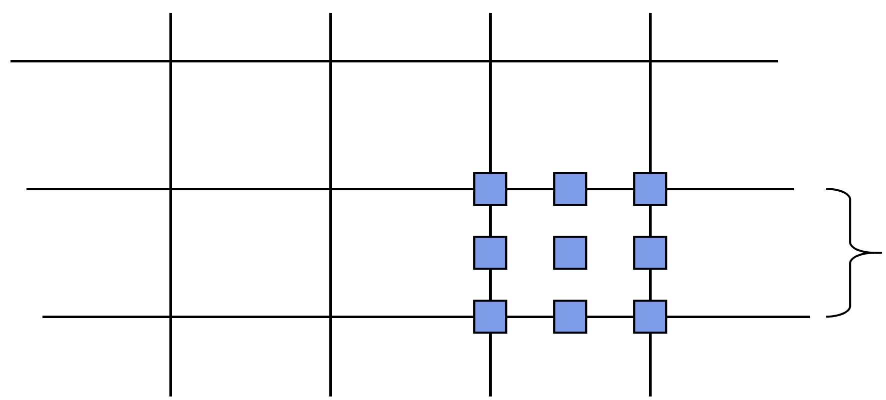
- Assumes we can compute color of any location (x,y) on screen
- Sample (x,y) in fractional (e.g. ½) increments, average samples
- Example: Double sampling = increments of ½ = 9 color values averaged for each pixel



Average 9 (x, y) values to find pixel color

Postfiltering

- Supersampling weights all samples equally
- Post-filtering: use unequal weighting of samples
- Compute pixel value as weighted average
- Samples close to pixel center given more weight



Sample weighting

1 / 16	1 / 16	1 / 16
1 / 16	1 / 2	1 / 16
1 / 16	1 / 16	1 / 16

Antialiasing in OpenGL

- Many alternatives
- Simplest: accumulation buffer
- **Accumulation buffer:** extra storage, similar to frame buffer
- Samples are accumulated
- When all slightly perturbed samples are done, copy results to frame buffer and draw

Antialiasing in OpenGL

- First initialize:
  - `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_ACCUM | GLUT_DEPTH);`
- Zero out accumulation buffer
  - `glClear(GLUT_ACCUM_BUFFER_BIT);`
- Add samples to accumulation buffer using
  - `glAccum( )`

Antialiasing in OpenGL

- Sample code
- jitter[] stores randomized slight displacements of camera,
- factor, f controls amount of overall sliding

```
glClear(GL_ACCUM_BUFFER_BIT);
for(int i=0;i < 8; i++)
{
   cam.slide(f*jitter[i].x, f*jitter[i].y, 0);
   display( );
   glAccum(GL_ACCUM, 1/8.0);
}
glAccum(GL_RETURN, 1.0);
```

jitter.h

-0.3348, 0.4353

0.2864, -0.3934

……

Computer Graphics
CS 4731 Lecture 26
Curves

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

So Far…

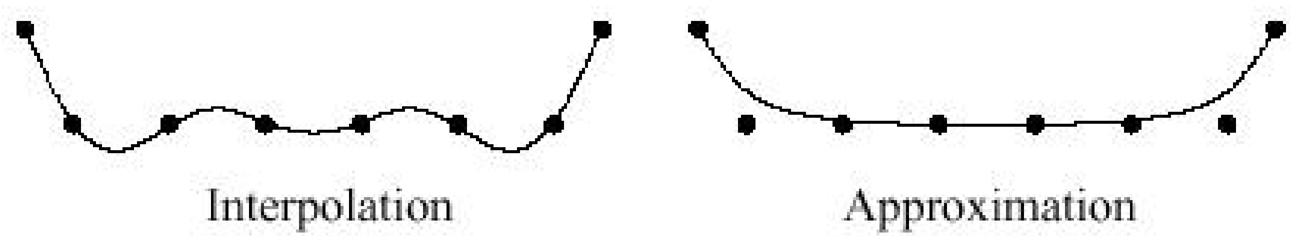
- Dealt with straight lines and flat surfaces
- Real world objects include curves
- Need to develop:
  - Representations of curves (mathematical)
  - Tools to render curves

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

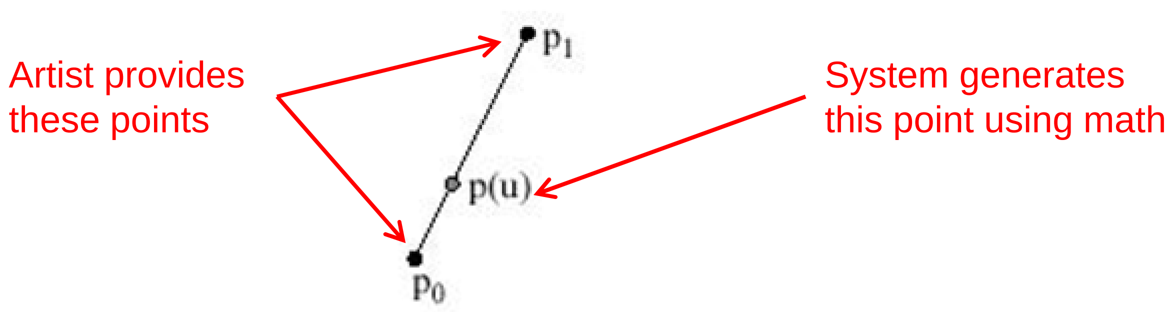
- 1 approach: curves pass through control points (interpolate)
- **Example:** Lagrangian Interpolating Polynomial
- Difficulty with this approach:
  - Polynomials always have "wiggles"
  - For straight lines wiggling is a problem
- Our approach: approximate control points (Bezier, B-Splines)



De Casteljau Algorithm

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

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

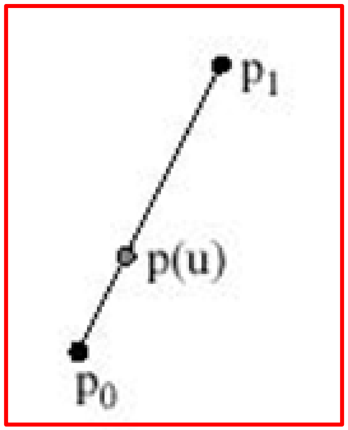


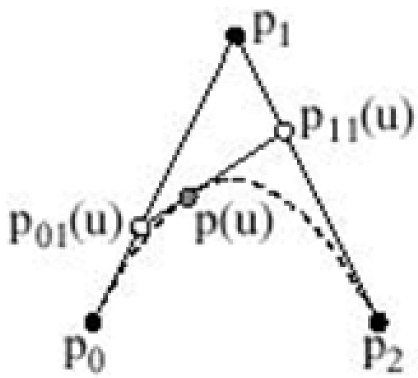
- Blending functions: *u* and *(1 – u)* are non-negative and sum to one

De Casteljau Algorithm

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

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





De Casteljau Algorithm

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

$$p(u) = (1-u)p_{01} + up_{11}(u)$$
$$= \underbrace{(1-u)^2}_{b_{02}(u)} \boxed{p_0} + \underbrace{(2u(1-u))}_{b_{12}(u)} \boxed{p_1} + \underbrace{u^2}_{b_{22}(u)} \boxed{p_2}$$

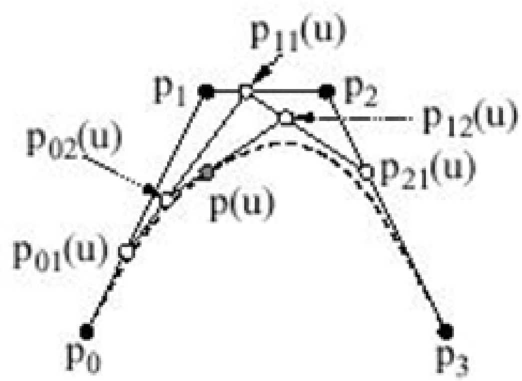
Blending functions for degree 2 Bezier curve

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

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

De Casteljau Algorithm

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



$$p(u) = \underbrace{(1-u)^3}_{b_{03}(u)} \boxed{p_0} + \underbrace{(3u(1-u)^2)}_{b_{13}(u)} \boxed{p_1} + \underbrace{(3u^2(1-u))}_{b_{23}(u)} \boxed{p_2} + \underbrace{u^3}_{b_{33}(u)}$$

- Final result above is Bezier curve of degree 3

De Casteljau Algorithm

$$p(u) = \underbrace{(1-u)^3}_{b_{03}(u)} \boxed{p_0} + \underbrace{(3u(1-u)^2)}_{b_{13}(u)} \boxed{p_1} + \underbrace{(3u^2(1-u))}_{b_{23}(u)} \boxed{p_2} + \underbrace{u^3}_{b_{33}(u)}$$

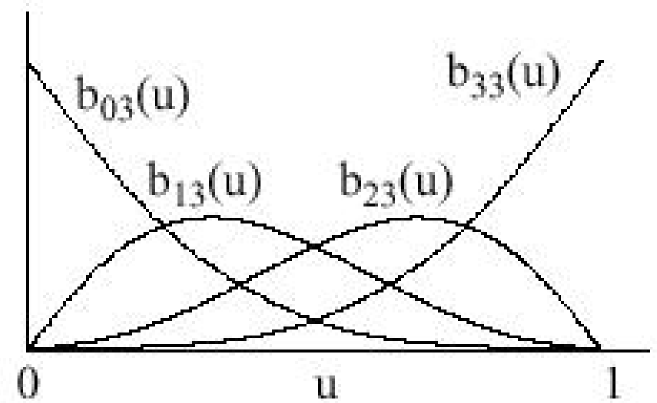
- Blending functions are polynomial functions called **Bernstein's polynomials**

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

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

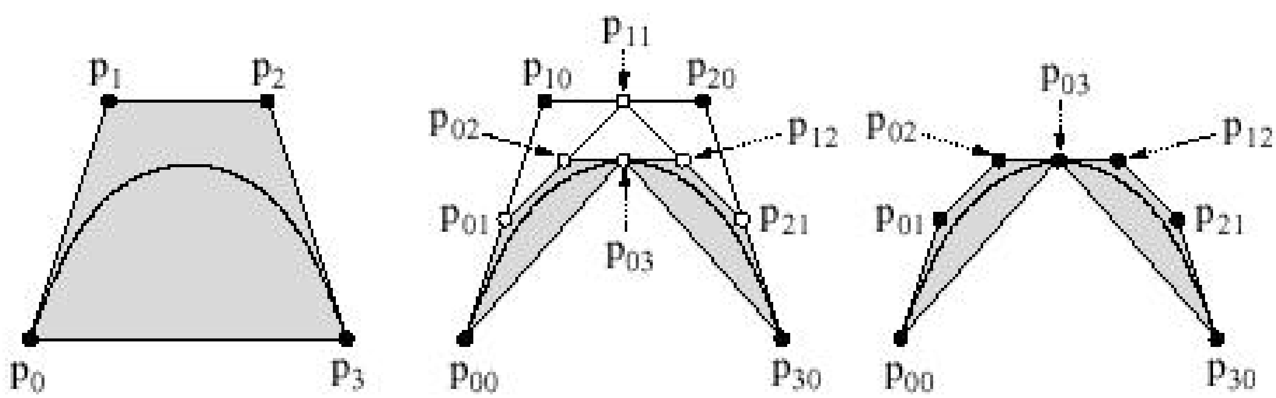
$$b_{23}(u) = 3u^2(1-u)$$

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



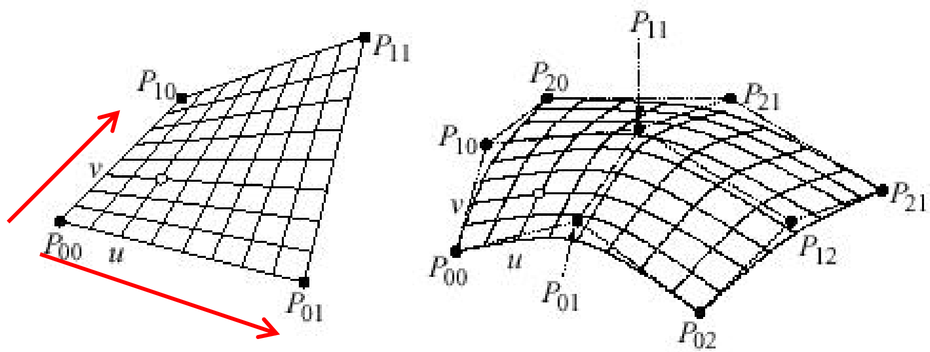
Subdividing Bezier Curves

- 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

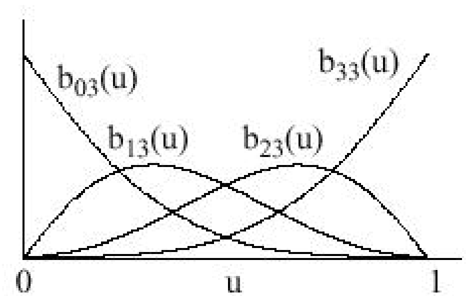
- Bezier surfaces: interpolate in two dimensions
- This called Bilinear interpolation
- Example: 4 control points, P00, P01, P10, P11,
  - 2 parameters **u** and **v**
- Interpolate between
  - P00 and P01 using u
  - P10 and P11 using u
  - P00 and P10 using v
  - P01 and P11 using v



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

Problems with Bezier Curves

- Bezier curves elegant but to achieve smoother curve
  - = more control points
  - = higher order polynomial
  - = more calculations

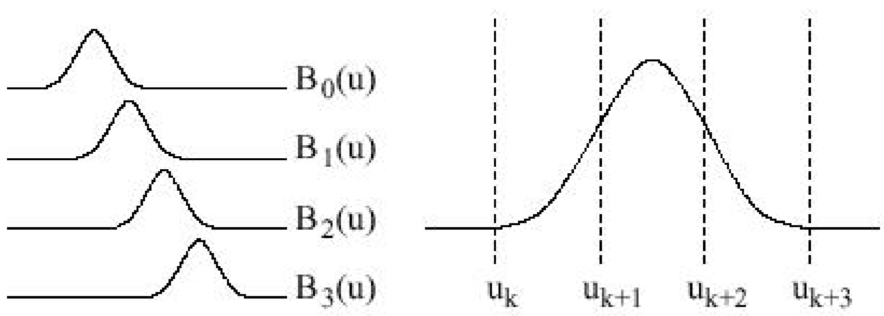


- **Global support problem:** All blending functions are non-zero for all values of *u*
- All control points contribute to all parts of the curve
- Means after modelling complex surface (e.g. a ship), if one control point is moves, recalculate everything!

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

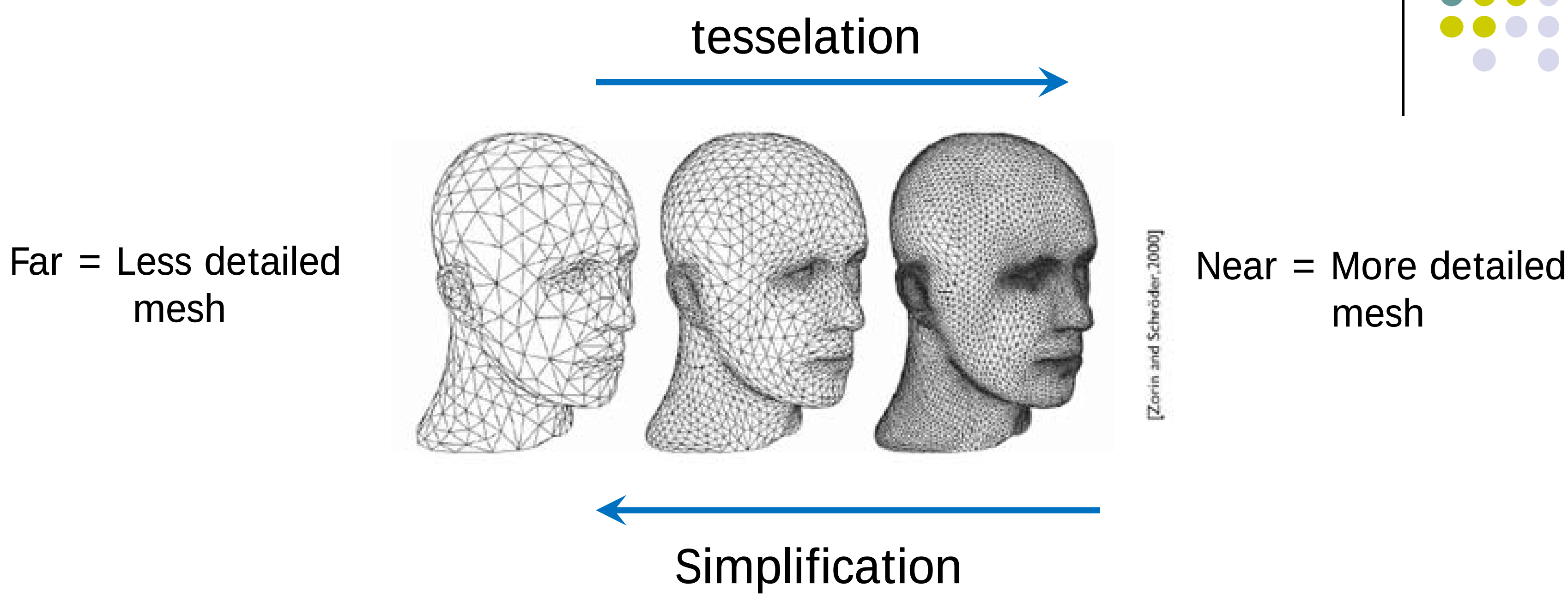
- Non-uniform Rational B-splines (NURBS)
- Rational function means ratio of two polynomials
- Some curves can be expressed as rational functions but not as simple polynomials
- No known exact polynomial for circle
- Rational parametrization of unit circle on xy-plane:

$$x(u) = \frac{1-u^2}{1+u^2}$$

$$y(u) = \frac{2u}{1+u^2}$$

$$z(u) = 0$$

Tesselation

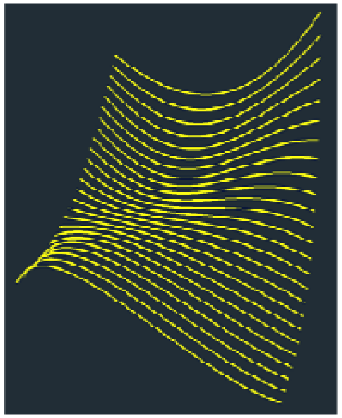


- **Previously:** Pre-generate mesh versions offline
- Tesselation shader unit new to GPU in DirectX 10 (2007)
  - Subdivide faces **on-the-fly** to yield finer detail, generate new vertices, primitives
- Mesh simplification/tesselation on GPU = Real time LoD

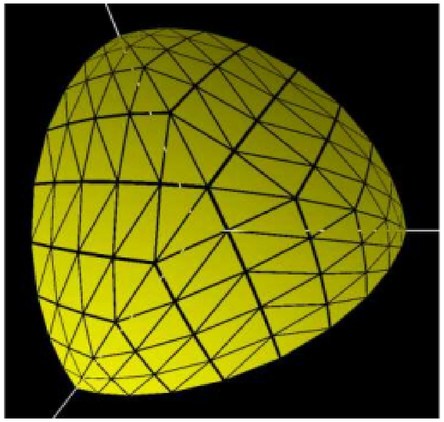
Tessellation Shaders

- Can subdivide curves, surfaces on the GPU

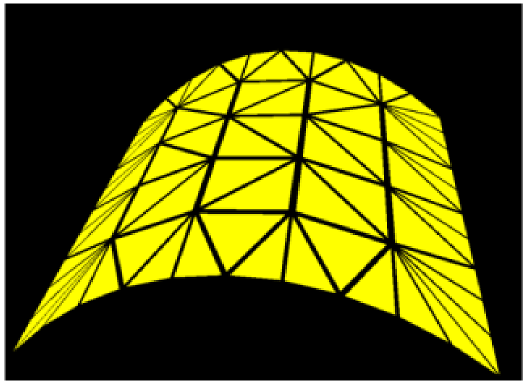
Lines



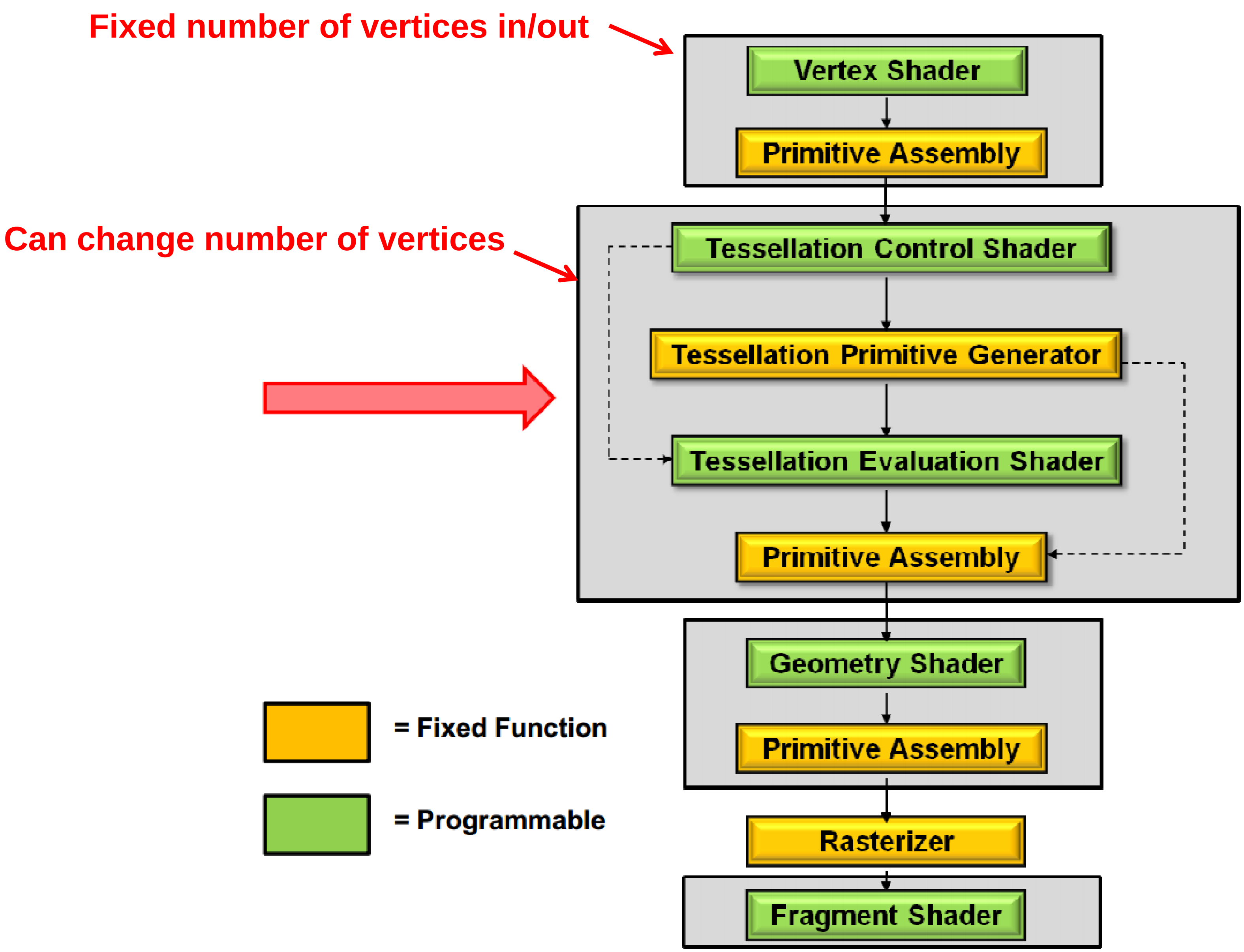
Triangles



Quads (subsequently broken into triangles)

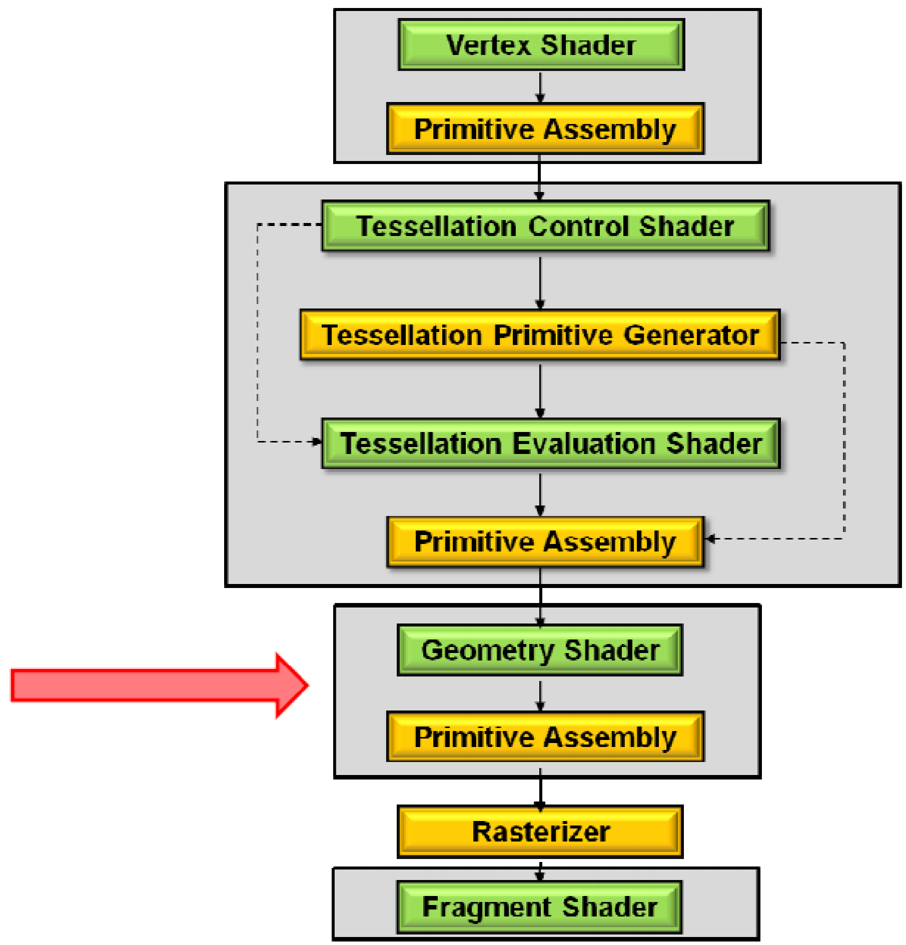


Where Does Tesselation Shader Fit?



Geometry Shader

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



References

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

Computer Graphics (CS 4731)
Lecture 26: Image Manipulation

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Image Processing

- Graphics concerned with creating artificial scenes from geometry and shading descriptions
- Image processing
  - Input is an image
  - Output is a modified version of input image
- Image processing operations include altering images, remove noise, super-impose images

Image Processing

- Example: Sobel Filter

Original Image

Sobel Filter

- Image Proc in OpenGL: Fragment shader invoked on each element of texture
  - Performs calculation, outputs color to pixel in color buffer

Luminance

- Luminance of a color is its **overall brightness (grayscale)**
- Compute it luminance from RGB as

Luminance = **R** * 0.2125 + **G** * 0.7154 + **B** * 0.0721

Image Negative

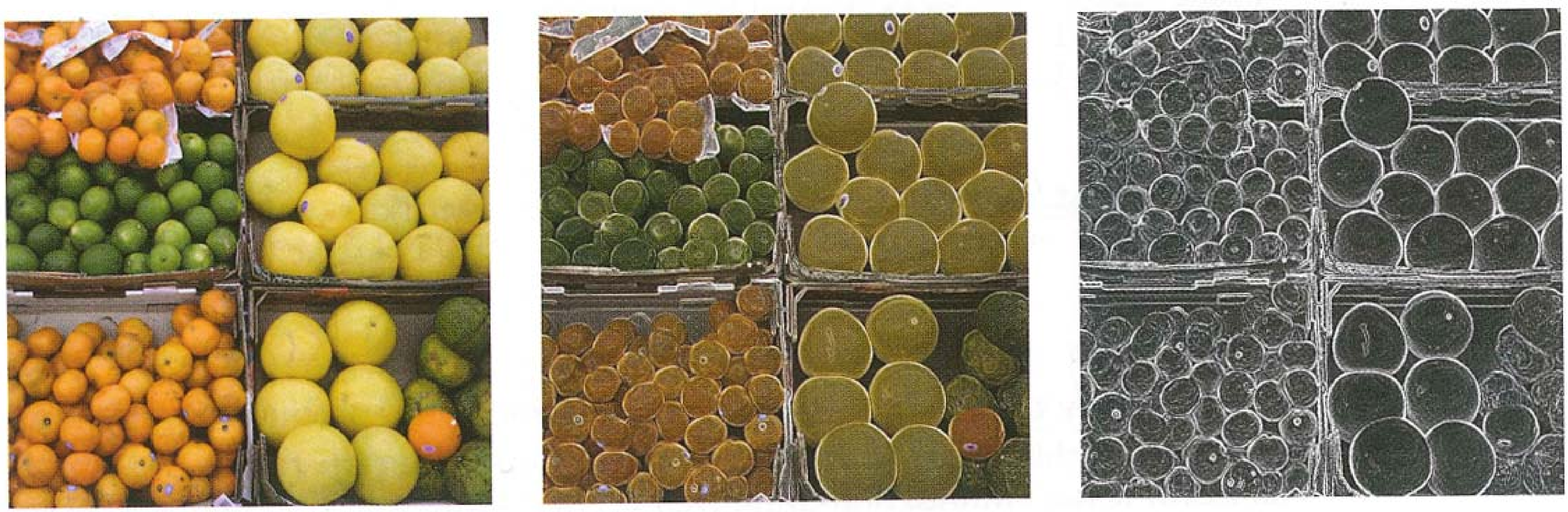
- Another example

(R, G, B)

(1.-R, 1.-G, 1.-B)

Edge Detection

- Edge Detection
  - Compare adjacent pixels
    - If difference is “large”, this is an edge
    - If difference is “small”, not an edge
- Comparison can be done in color or luminance



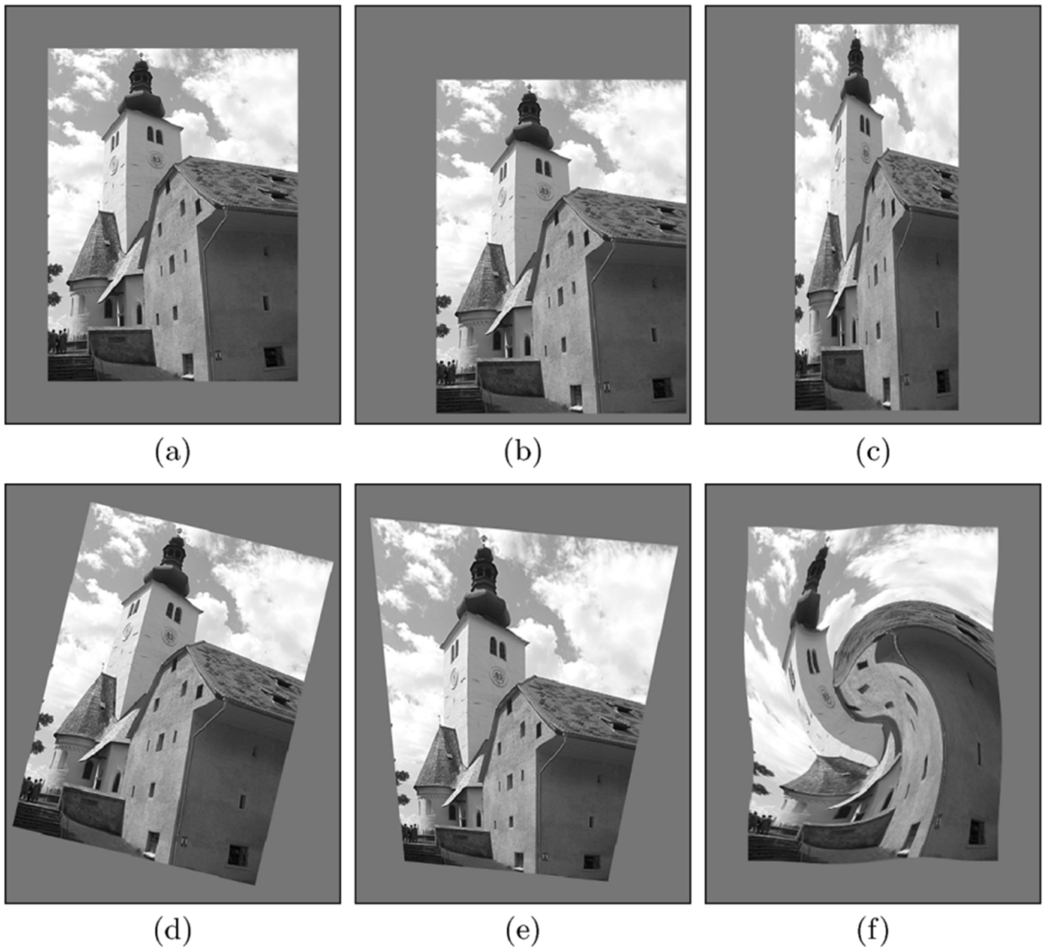
Embossing

- Embossing is similar to edge detection
- Replace pixel color with grayscale proportional to contrast with neighboring pixel
- Add highlights depending on angle of change

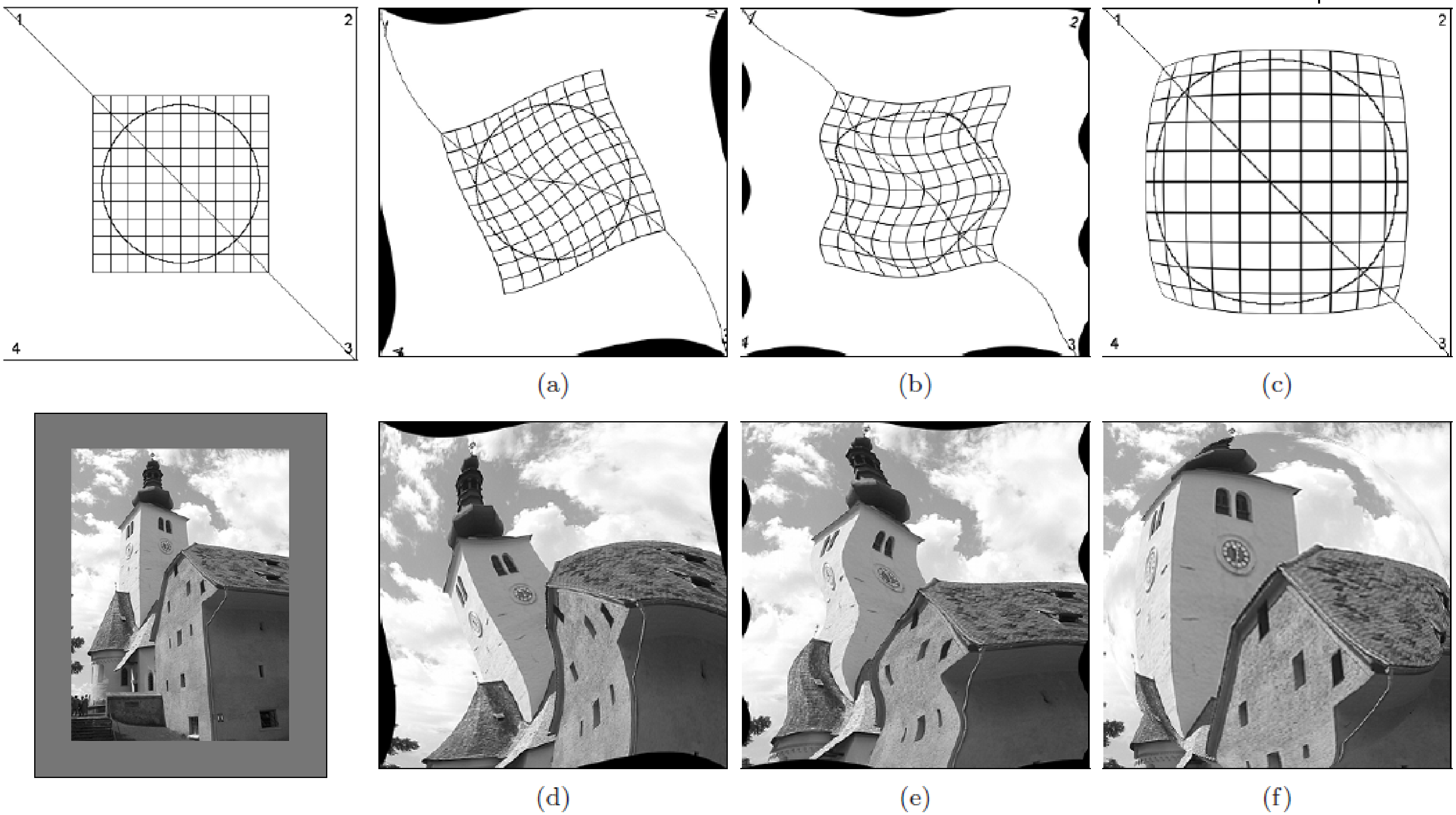
Toon Rendering for Non-Photorealistic Effects

Geometric Operations

- **Examples:** translating, rotating, scaling an image



Non-Linear Image Warps



Original Twirl Ripple Spherical

References

- Mike Bailey and Steve Cunningham, Graphics Shaders (second edition)
- Wilhelm Burger and Mark Burge, Digital Image Processing: An Algorithmic Introduction using Java, Springer Verlag Publishers
- OpenGL 4.0 Shading Language Cookbook, David Wolff
- Real Time Rendering (3rd edition), Akenine-Moller, Haines and Hoffman
- Suman Nadella, CS 563 slides, Spring 2005

Computer Graphics
CS 4731 – Final Review

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Exam Overview

- Thursday, October 16, 2014 in-class
- Midterm covered up to lecture 13 (Viewing & Camera Control)
- Final covers lecture 14 till today's class (lecture 26)
- Can bring:
  - 1 page cheat-sheet, hand-written (not typed)
  - Calculator
- Will test:
  - Theoretical concepts
  - Mathematics
  - Algorithms
  - Programming
  - OpenGL/GLSL knowledge (program structure and commands)

Topics

- Projection
- Lighting, shading and materials
- Shadows and fog
- Texturing & Environment mapping
- Image manipulation
- Clipping (2D and 3D clipping) and viewport transformation
- Hidden surface removal
- Rasterization (line drawing, polygon filling, antialiasing)
- Curves