

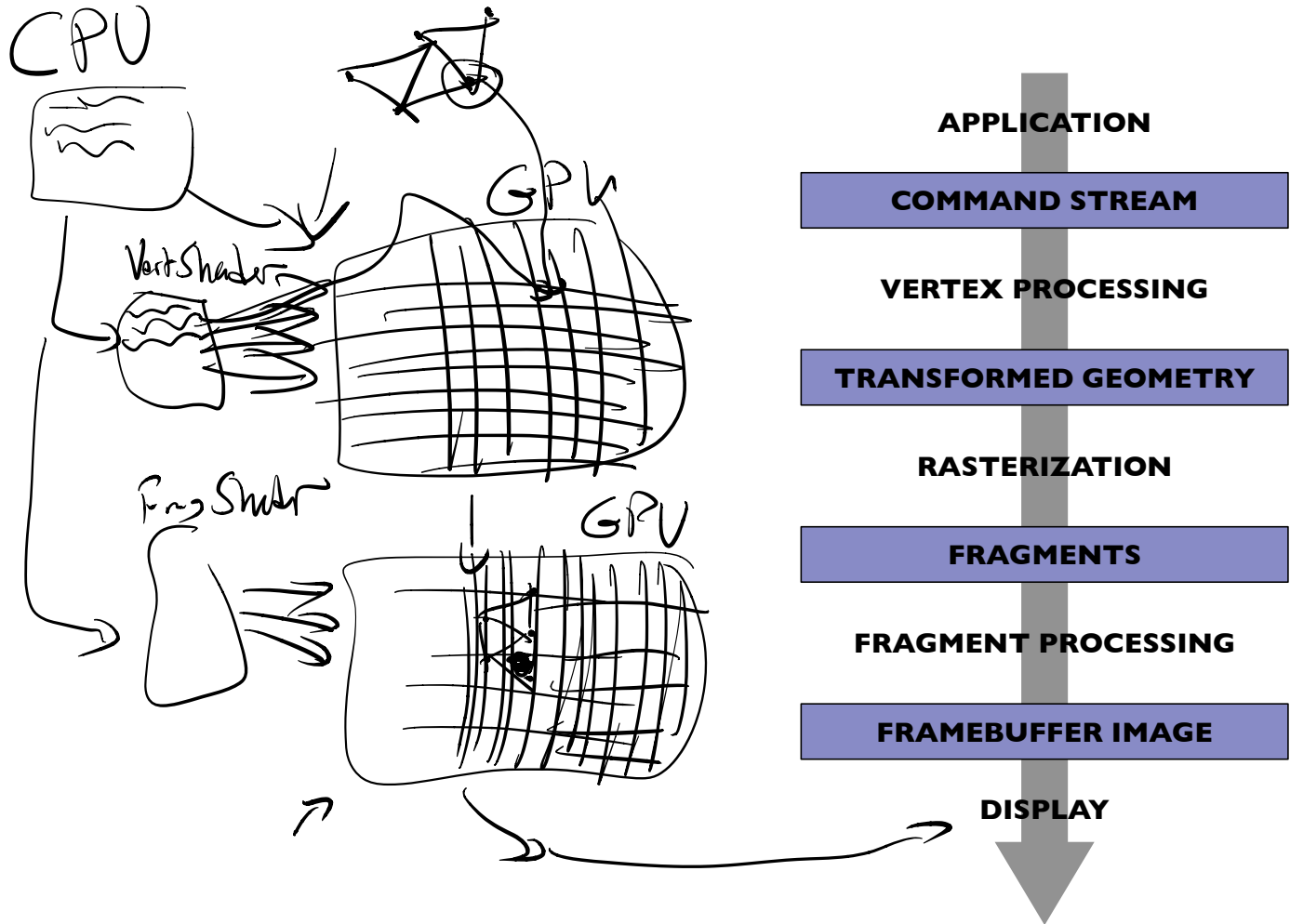
Computer Graphics

Lecture 24:
Shading in the Graphics Pipeline
Clipping

Announcements

- GL lab solution github repo posted on the course webpage - one commit per task
- Midterm exam released after class.
- Upload 3 files to Canvas by Monday 10pm:
 - obj: solution to problem 1
 - .txt: solution template for remaining problems
 - .pdf showing your work for problems 2-6
- Project proposal feedback by Tuesday.

Graphics Pipeline: Overview



Rendering Realistic Images

- We have a pipeline that gives us access to the compute power of shaders and does a bunch of nice things for us.
- We know how to get data in and out
- How do we realistic-looking images using shading models like Lambertian and Blinn-Phong?

Rendering Realistic Images

- We have a pipeline that gives us access to the compute power of shaders and does a bunch of nice things for us.
- We know how to get data in and out
- How do we realistic-looking images using shading models like Lambertian and Blinn-Phong?

but first, a rant about terminology

Phong shading Lambertian shading in the fragment shader

- Shade (v.): determine color of a pixel
basically all of computer graphics...
- Shader (n.): a program that runs on GPU
vertex shader, fragment shader
- Shading model (**reflection** or **illumination model**):
light interaction model that determines a pixel's color
Lambertian reflection, Blinn-Phong reflection
- Shading algorithm (**interpolation technique**):
when, and in which shader, is the reflection model
computed, and using what normals?
flat shading, Gouraud shading, Phong shading

Flat shading (interpolation)

- Shade using the real normal of the triangle
 - same result as ray tracing a bunch of triangles without normal interpolation
- Leads to constant shading and faceted appearance
 - truest view of the mesh geometry

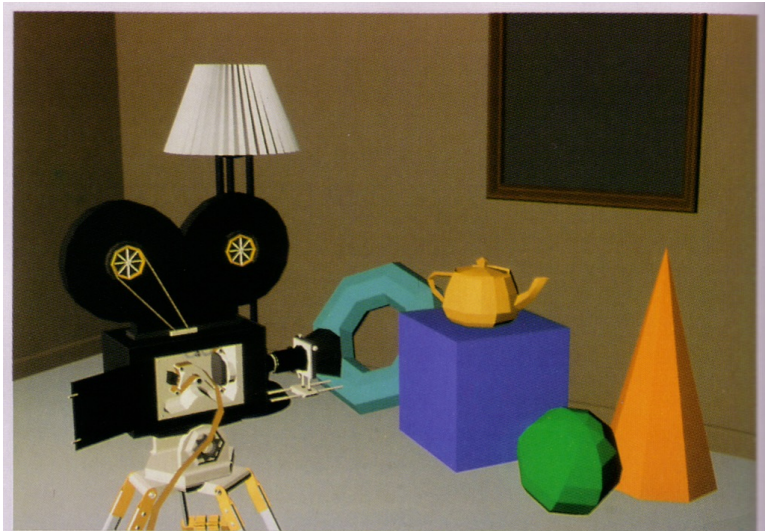
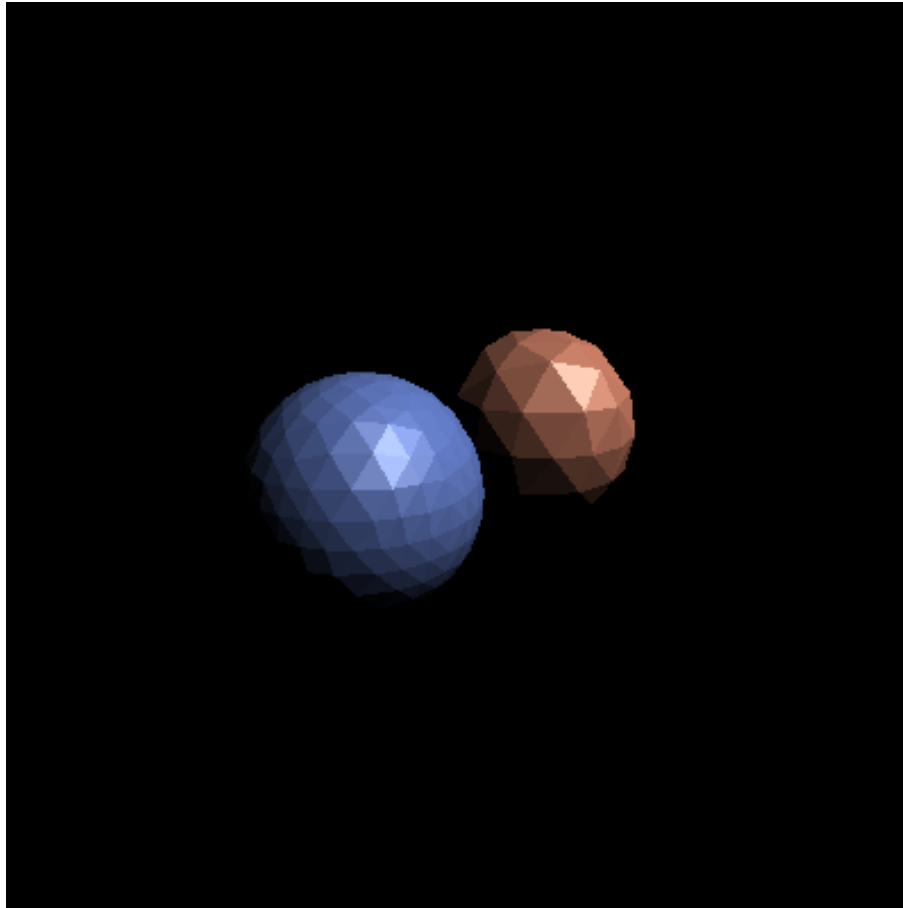


Plate II.29 *Shutterbug*. Individually shaded polygons with diffuse reflection (Sections 14.4.2 and 16.2.3). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Pipeline for flat shading

- Vertex stage (input: position / vtx; color and normal / tri)
 - transform position and normal (object to eye space)
 - compute shaded color per triangle using normal
 - transform position (eye to screen space)
- Rasterizer
 - interpolated parameters: z' (screen z)
 - pass through color
- Fragment stage (output: color, z')
 - write to color planes only if interpolated $z' <$ current z'

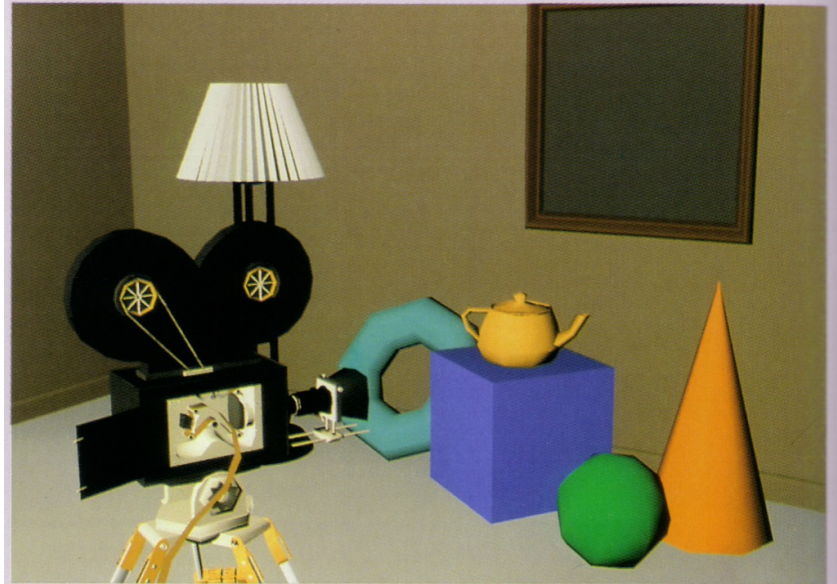
Result of flat-shading pipeline



Gouraud shading

- Often we're trying to draw smooth surfaces, so facets are an artifact
 - compute colors at vertices using vertex normals
 - interpolate colors across triangles
 - “Gouraud shading”
 - “Smooth shading”

Plate II.30 *Shutterbug*. Gouraud shaded polygons with diffuse reflection (Sections 14.4.3 and 16.2.4). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)



Pipeline for Gouraud shading

- Vertex stage (input: position, color, and normal / vtx)

- transform position and normal (object to eye space)
- compute shaded color per vertex
- transform position (eye to screen space)

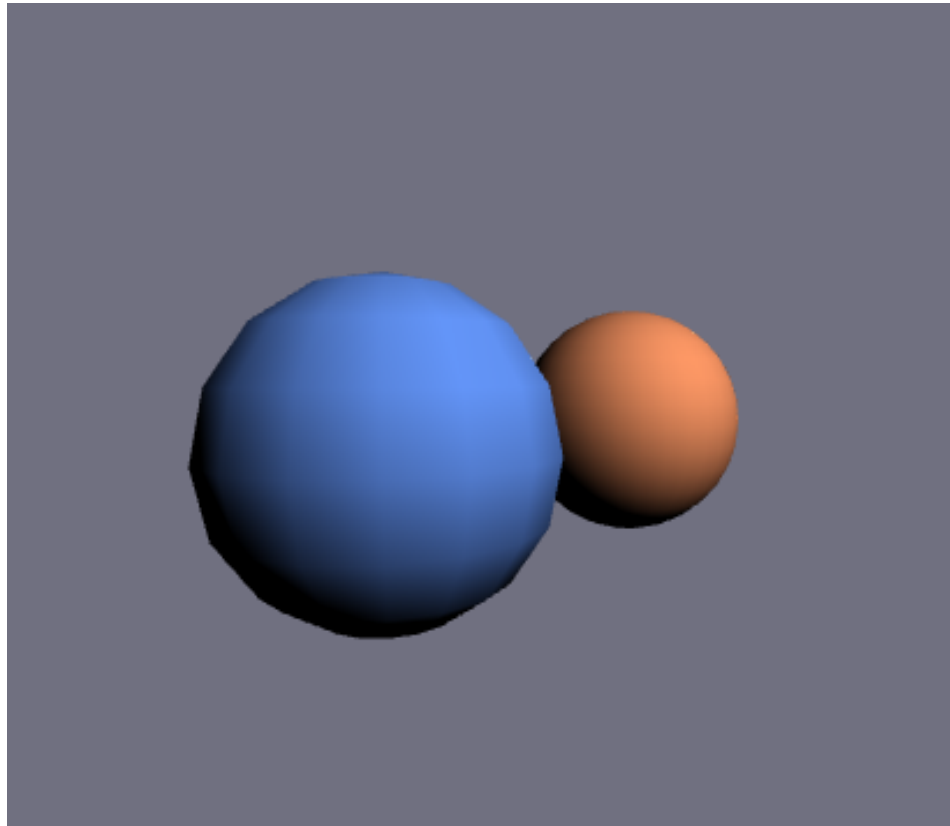
- Rasterizer

- interpolated parameters: z' (screen z); r, g, b color

- Fragment stage (output: color, z')

- write to color planes only if interpolated $z' < \text{current } z'$

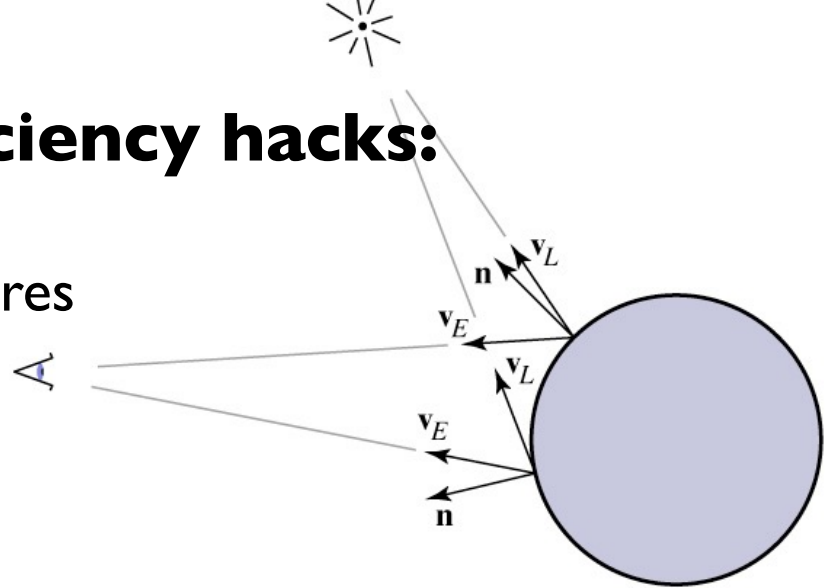
Result of Gouraud shading pipeline



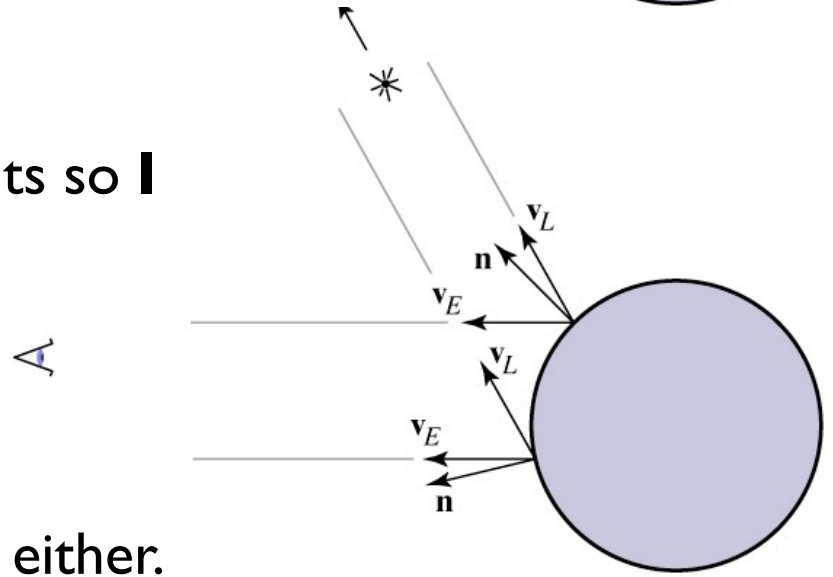
Demo

Some possible efficiency hacks:

- Blinn-Phong model requires knowing
 - normal
 - light direction
 - view direction



- Hack: use directional lights so \mathbf{l} doesn't change



- Hack: pretend viewer is infinitely distant so view direction doesn't change either.

Non-diffuse Gouraud shading

- Can apply Gouraud shading to any illumination model
 - it's just an interpolation method
- Results are not so good with fast-varying models like specular ones
 - problems with any highlights smaller than a triangle
 - (demo)

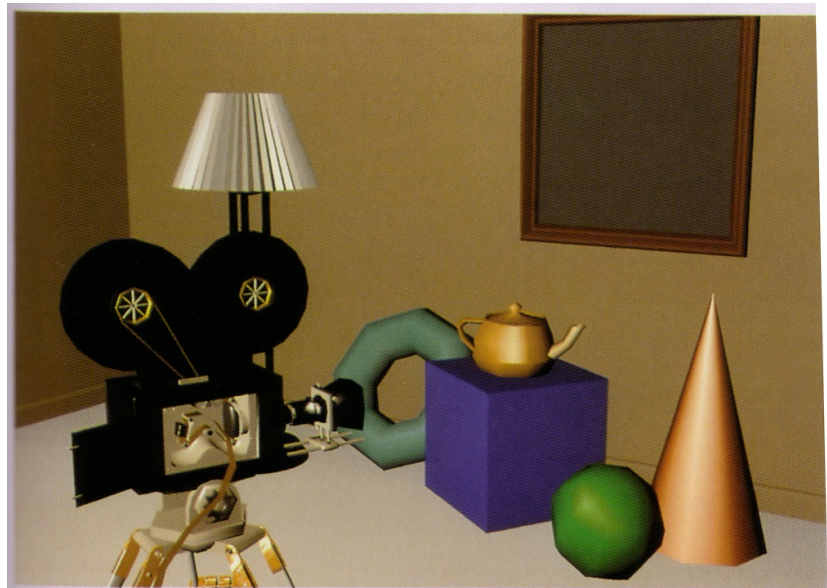
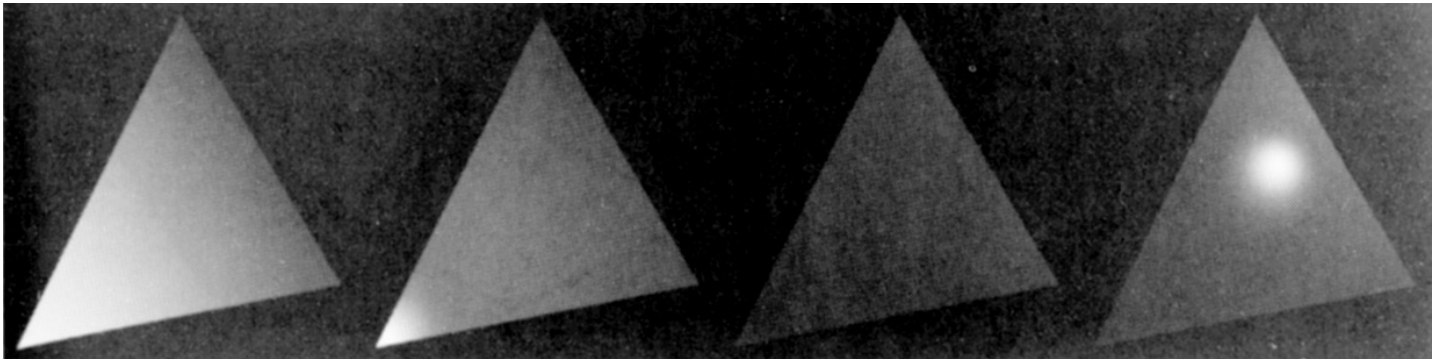


Plate II.31 *Shutterbug*. Gouraud shaded polygons with specular reflection (Sections 14.4.4 and 16.2.5). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Per-pixel (Phong*) shading

- Get higher quality by interpolating the normal
 - just as easy as interpolating the color
 - but now we are evaluating the illumination model per pixel rather than per vertex (and normalizing the normal first)
 - in pipeline, this means we are moving illumination from the vertex processing stage to the fragment processing stage



Per-pixel (Phong) shading

- Bottom line: produces much better highlights



Shutterbug. Gouraud shaded polygons with specular reflection (Sections 14.4.4 and 16.2.5). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Plate II.32 *Shutterbug*. Phong shaded polygons with specular reflection (Sections 14.4.4 and 16.2.5). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

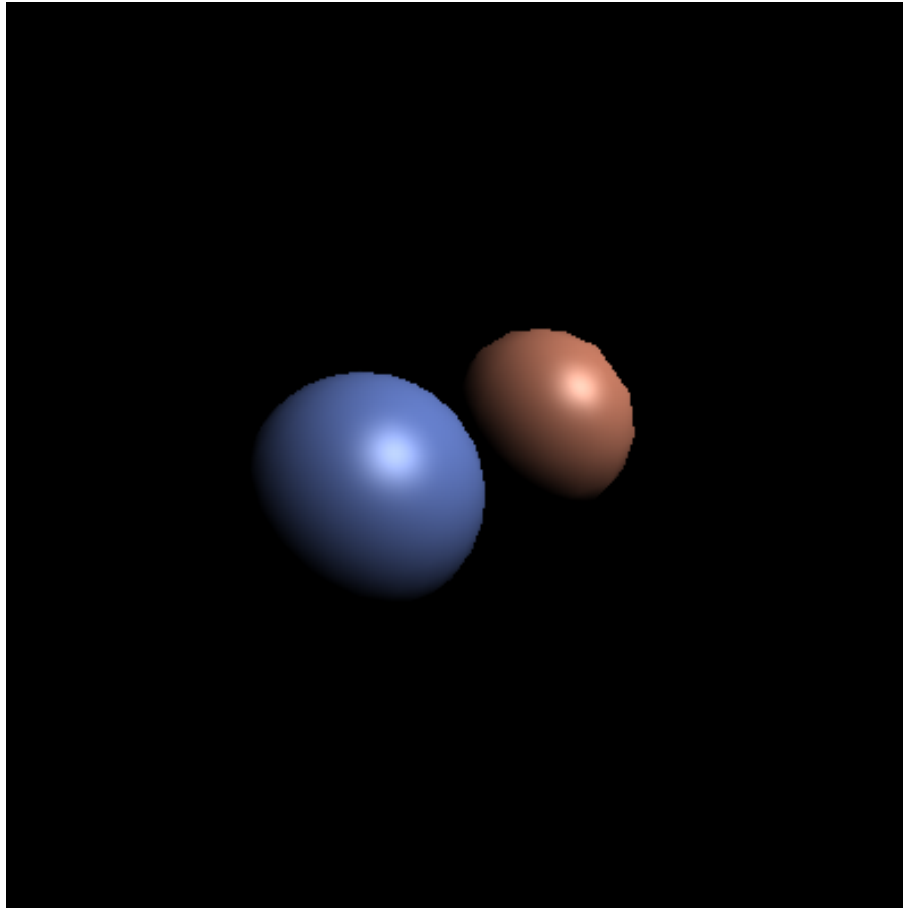


[Foley et al.]

Pipeline for per-pixel shading

- Vertex stage (input: position, color, and normal / vtx)
 - transform position and normal (object to eye space)
 - transform position (eye to screen space)
 - pass through color
- Rasterizer
 - interpolated parameters: z' (screen z); r, g, b color; x, y, z normal
- Fragment stage (output: color, z')
 - compute shading using interpolated color and normal
 - write to color planes only if interpolated $z' <$ current z'

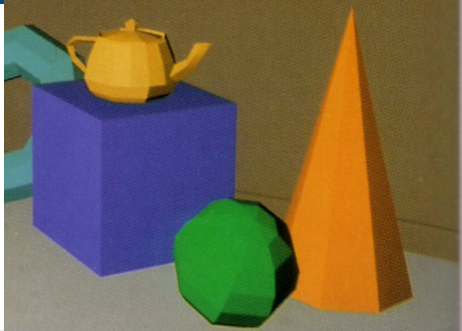
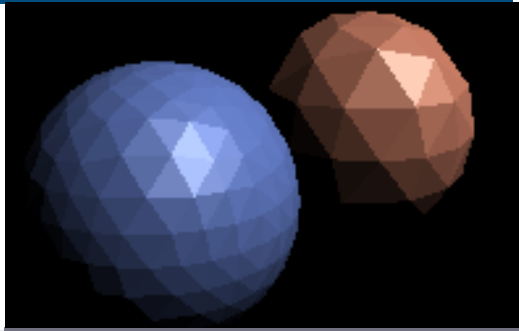
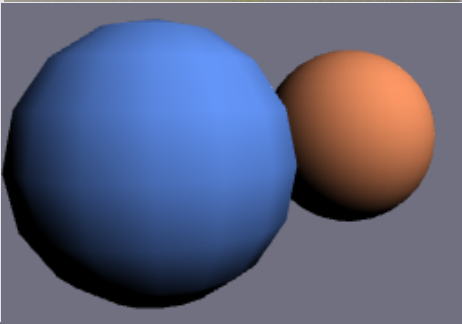
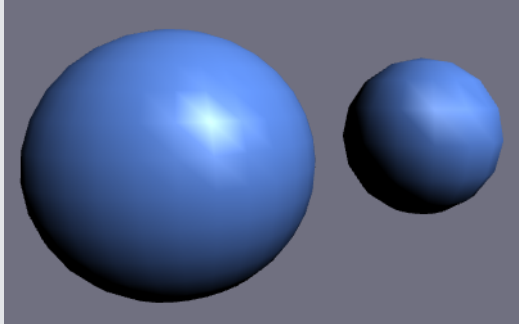
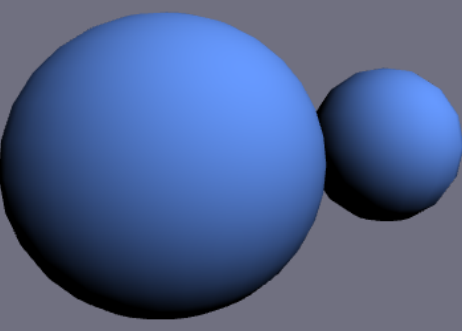
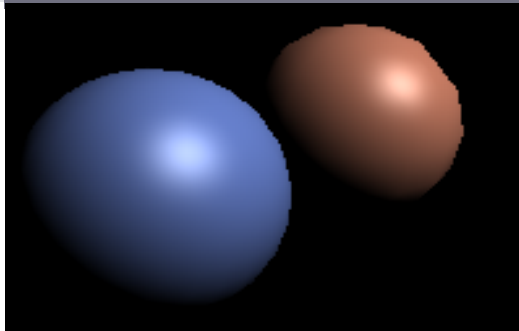
Result of per-pixel shading pipeline



(demo)

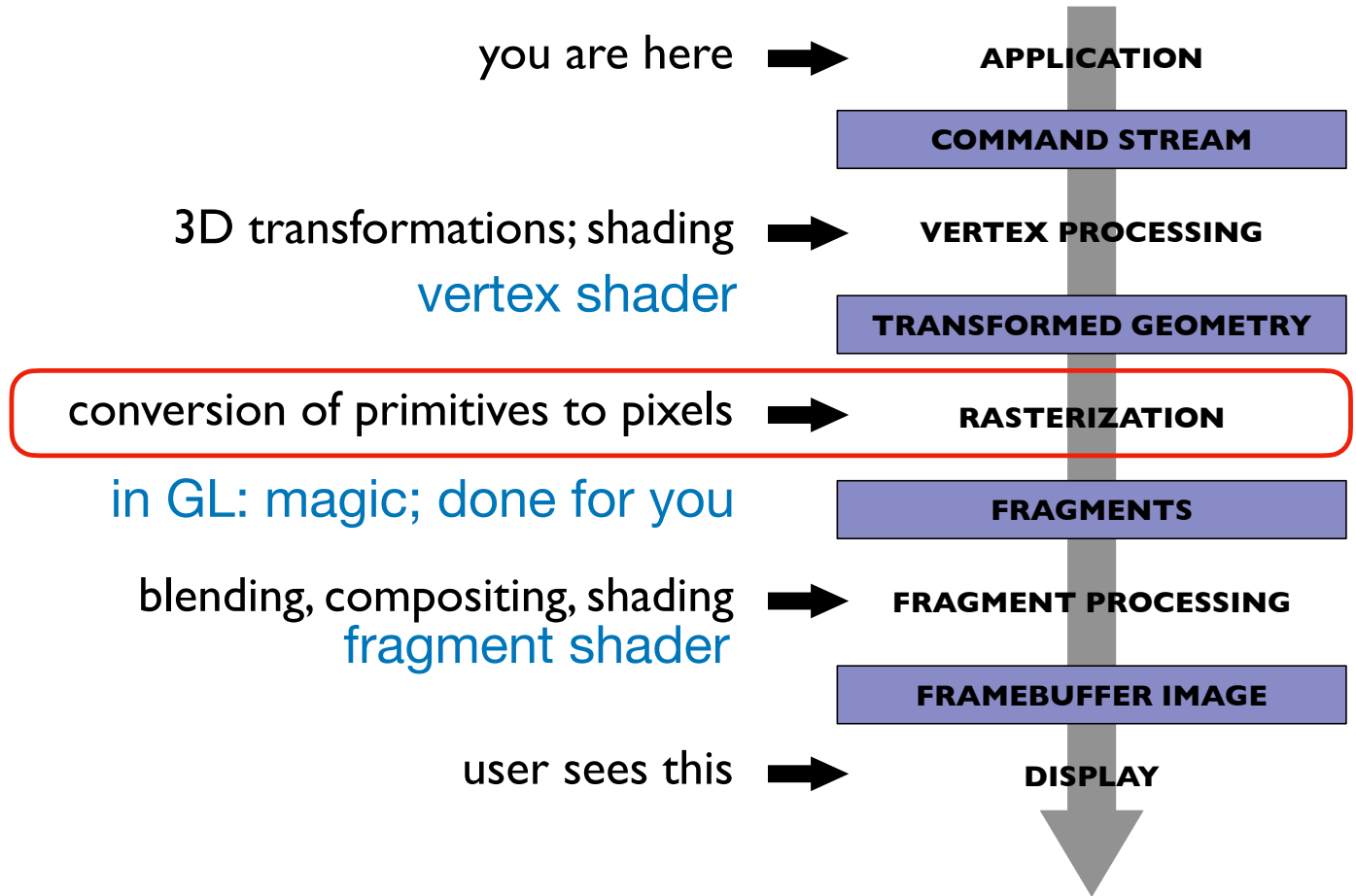
Summary: Shading and Interpolation Techniques

reflection

		Lambertian	Blinn-phong
interpolation	Flat		
	Gouraud		
	Phong		

Questions?

Graphics Pipeline: Overview



Rasterization: Overview

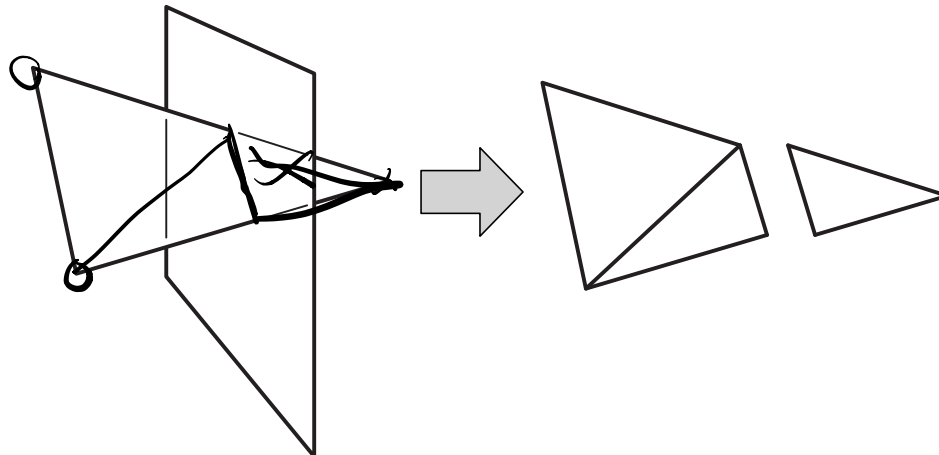
- 7(!?) weeks ago: rasterizing triangles
- Today: **clipping**
- Next week: rasterizing lines

Clipping

- Rasterizer tends to assume triangles are on screen
 - particularly problematic to have triangles crossing the plane $z = 0$
- After projection
 - clip against the planes $x, y, z = 1, -1$ (6 planes)
 - primitive operation: clip triangle against axis-aligned plane

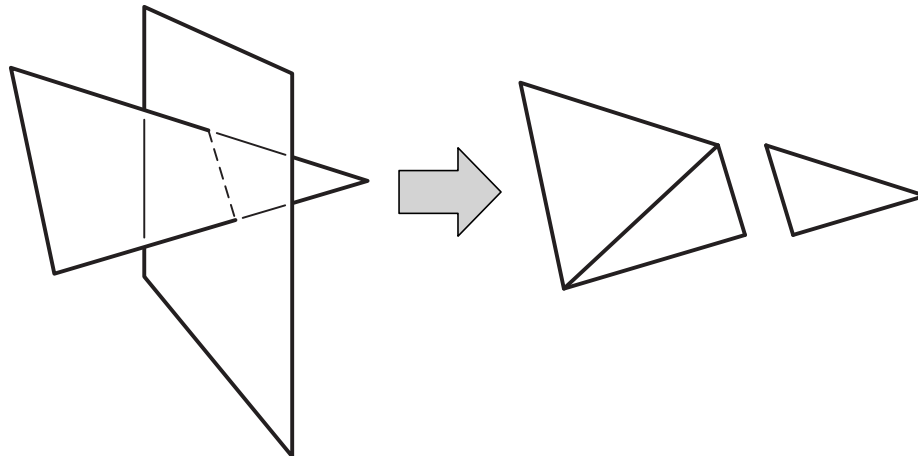
Clipping a triangle against a plane

- 4 cases, based on sidedness of vertices
 - all in (keep)
 - all out (discard)
 - one in, two out (one clipped triangle)
 - two in, one out (two clipped triangles)



Exercise: Write pseudocode to do this.

- 4 cases, based on sidedness of vertices
 - all in (keep)
 - all out (discard)
 - one in, two out (one clipped triangle)
 - two in, one out (two clipped triangles)



```
Which-case(a, b, c) {  
  v-out = []; v-in = [];  
  for v in (a, b, c)  
    if v[i] > 1  
      push!(v-out, v)  
    else  
      push!(v-in, v)
```

```
  if length(v-out) == 3:  
    return Nothing // case 2
```

```
  if length(v-in) == 3:  
    return (a, b, c) // case 1
```

To be continued...