Shading and Shadows

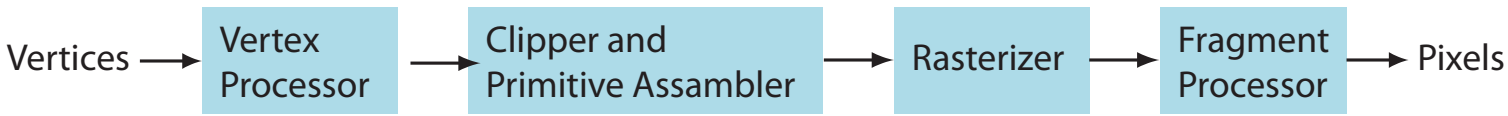
Computer Graphics and Visualization

Stefan Johansson
Department of Computing Science
Fall 2016

Polygonal Shading

- Or surface rendering!
- Previous lecture: How to compute the illumination model **at a point** on a surface.
  - The calculations are costly, including several normalizations of vectors

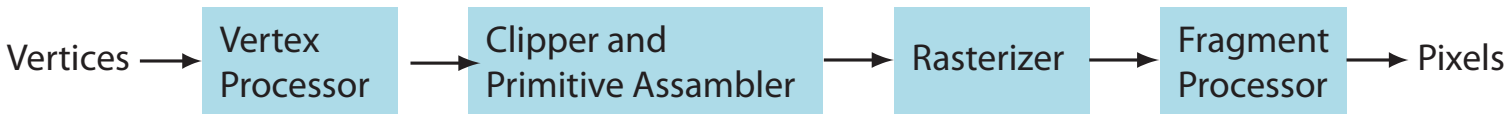
The graphic pipeline:



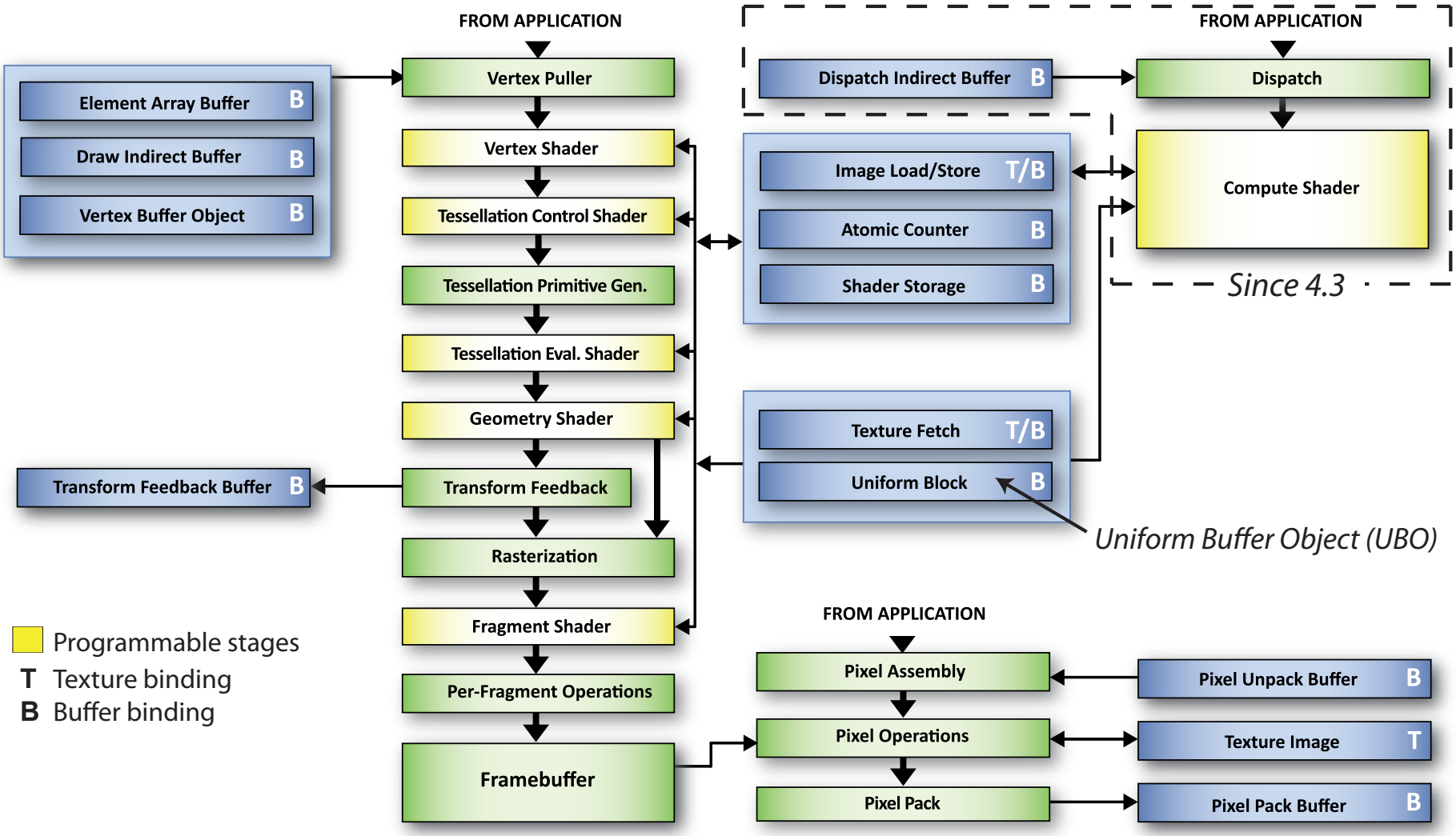
Polygonal Shading

- Or surface rendering!
- Previous lecture: How to compute the illumination model **at a point** on a surface.
  - The calculations are costly, including several normalizations of vectors
- Compute it only for a few points and interpolate
  - At which point should we apply the model?
  - Where (in the graphical pipeline) and how often it is applied have a noticeable effect on the end result
- Three major shading techniques: *Flat*, *Gouraud*, and *Phong*

The graphic pipeline:



OpenGL Pipeline (4.3+)

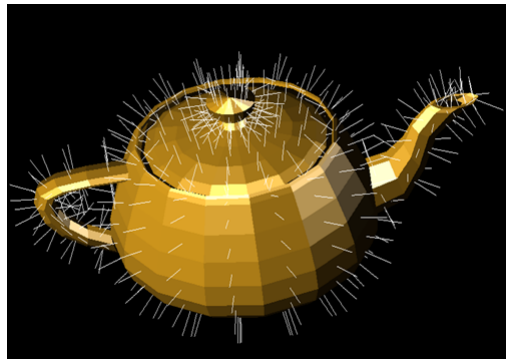


https://www.opengl.org/sdk/docs/reference_card/opengl45-reference-card.pdf

Flat Shading

- Applying the illumination calculation once per face (primitive)
- Introducing normals on the surface

$$n = (v2 - v0) \times (v1 - v0)$$



- Enable flat shading in the vertex shader by

```
out flat vec3 flatNormal;
```

- Illumination is usually calculated at the centroid of the face

$$I = \frac{\sum_{i=1}^{n_{\text{vertices}}} I_i}{n_{\text{vertices}}}$$

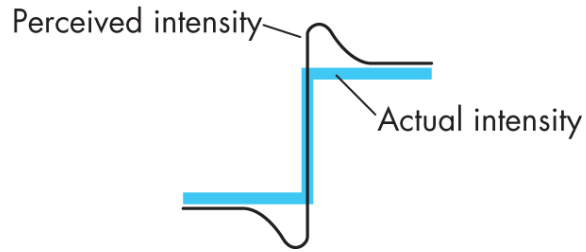
Flat Shading

Issues

- For point light sources, the direction light varies over the face
- For specular reflections the direction to the eye varies over the face
- Mach bands

One normal for each face is obviously not enough

Mach Bands

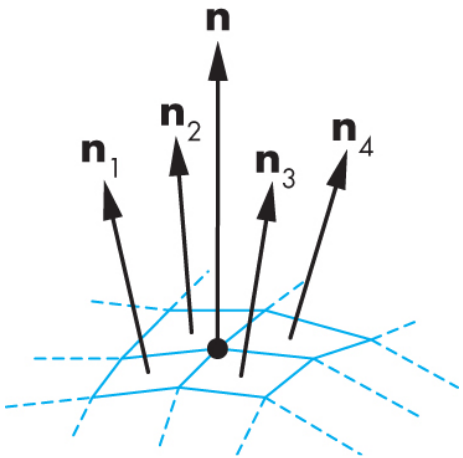


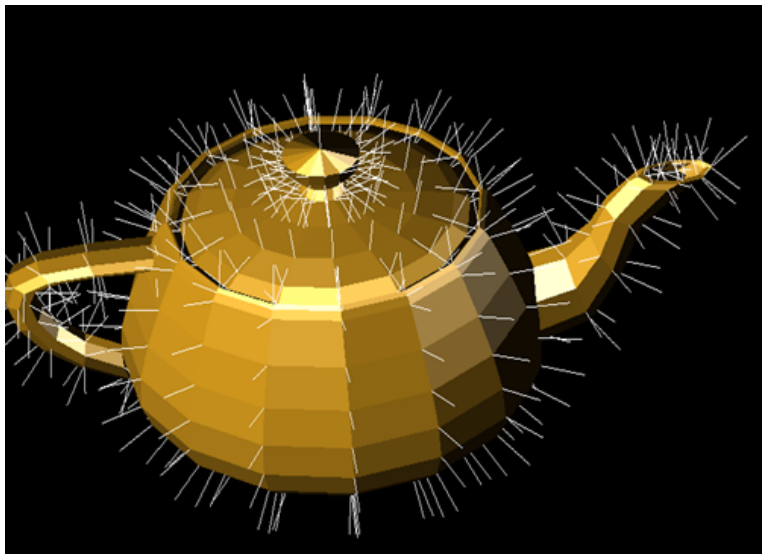
Gouraud Shading

- Vertex normals instead of face normals
  - Normalized average of all adjacent face normals

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$

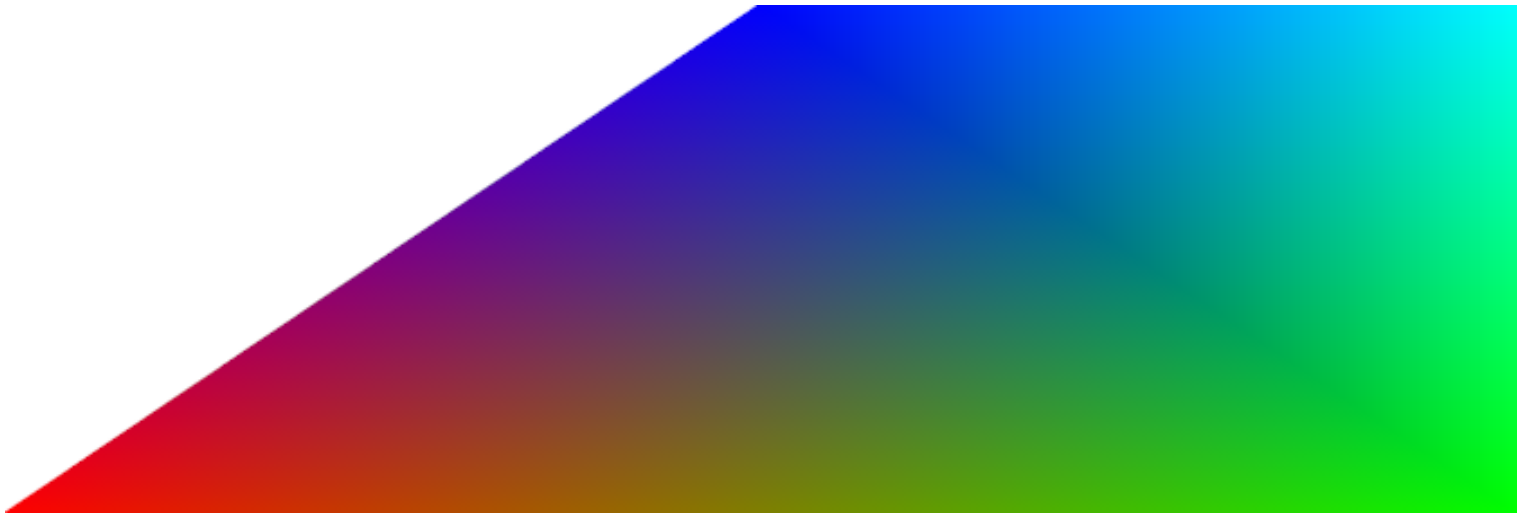
- Must provide vertices as well as normals to the vertex shader
- Compute color at each vertex (in vertex shader)





Gouraud Shading

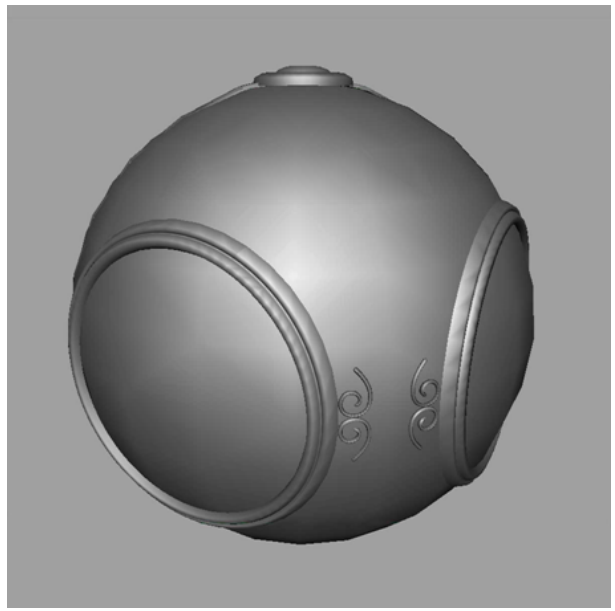
- Interpolates the vertex colors along the edges and scan lines
- Out parameters in the vertex shader are interpolated by default



Gouraud Shading and artifacts

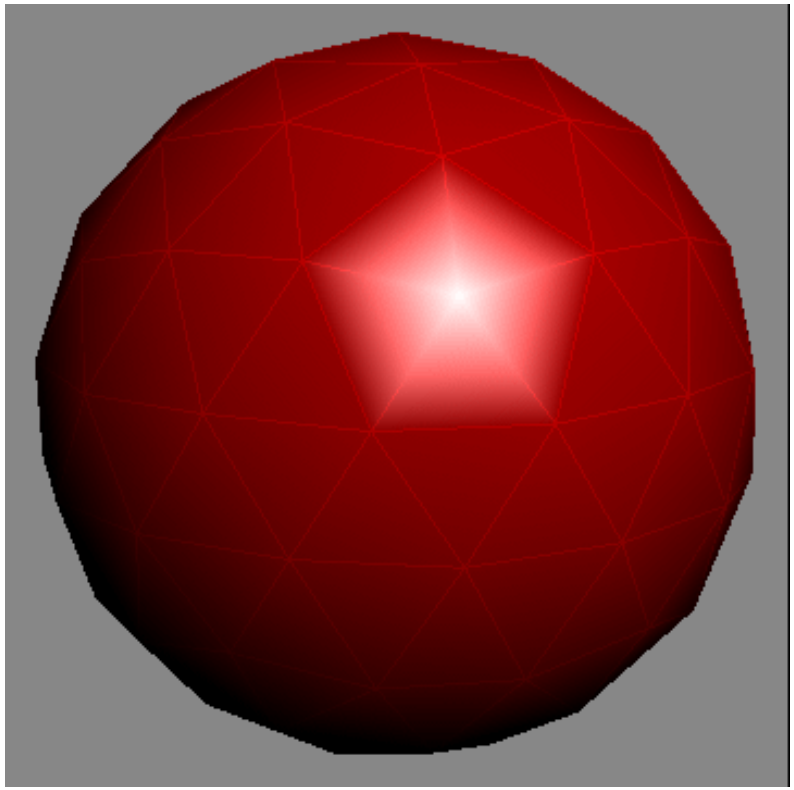
Still artifacts present, highlights sometime show anomalies

- Specular highlight inside a primitive disappears
- Spreads highlight over the primitive
- Mach banding



Gouraud Shading and artifacts

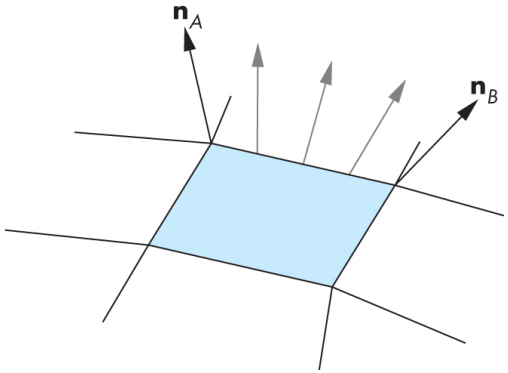
Specular highlight centered on a vertex is enhanced



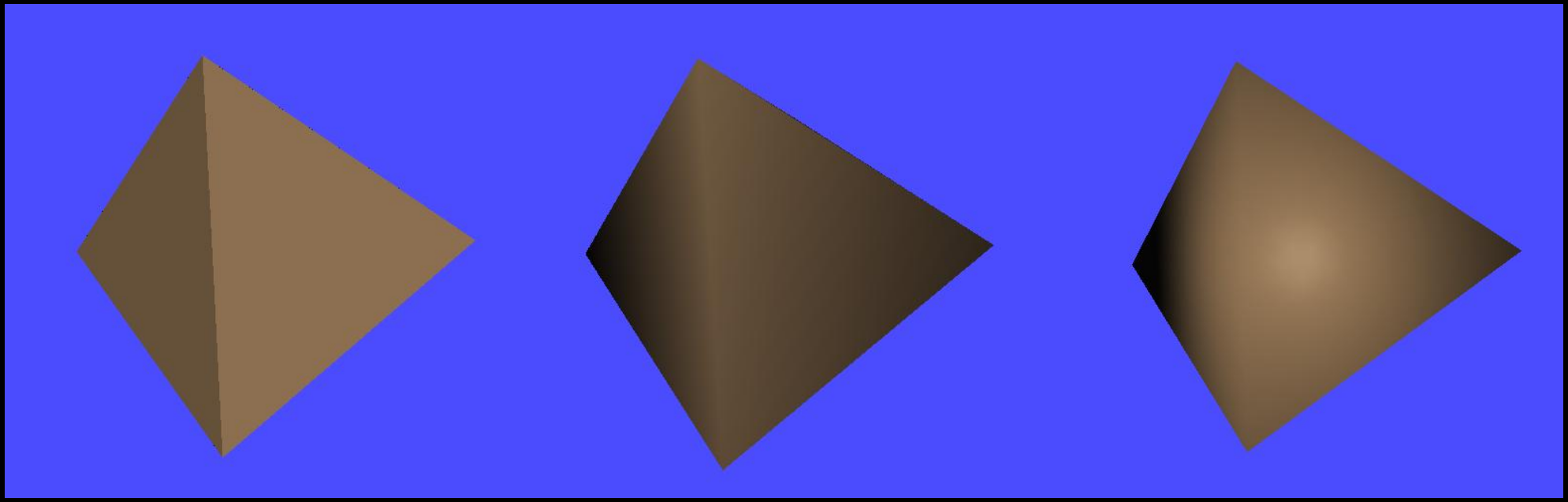
Phong Shading

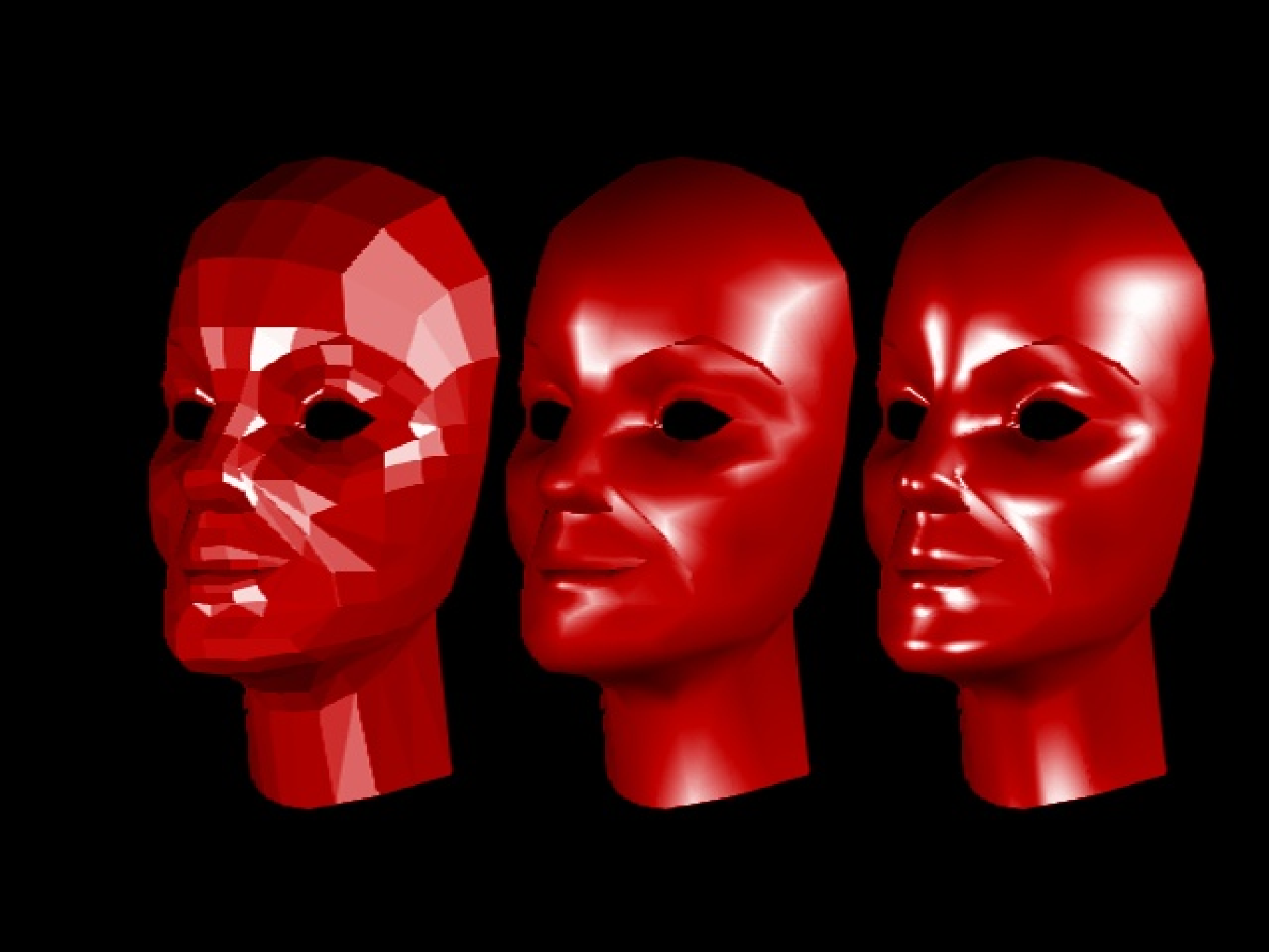
- Not the same as Phong's lighting model! Shading $\neq$ Lighting
- Lighting model applied to every point on the primitives surface in the fragment shader
- Requires a normal per vertex (as Gouraud)

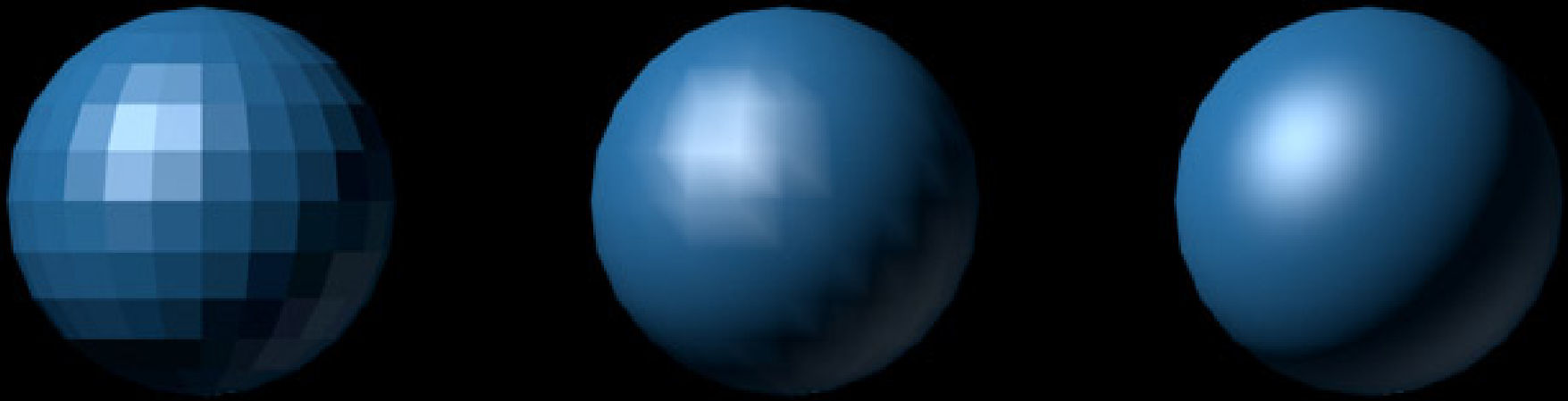
- **Interpolates the normals** over each point on the surface (Gouraud interpolates colors!)
- Drawbacks
  - Computational demanding



Comparison







Shaders example – Gouraud

Vertex Shader

```
in vec4 vPosition;
in vec3 vNormal;
out vec4 color; // interpolated
uniform mat4 P;
uniform mat4 V;
uniform mat4 M;
// Light and material properties goes here...

void main()
{
    // Compute the illumination...

    color = ambient + diffuse + specular;
    color.a = 1.0;
    gl_Position =  P*V*M*vPosition;
}
```

Shaders example – Gouraud

Fragment Shader

```
in vec4 color;
out vec4 fColor;

void main()
{
    fColor = color;
}
```

Shaders example – Phong

Vertex Shader – Part 1

```
in vec4 vPosition;
in vec3 vNormal;

// Outputs that are interpolated per fragment
out vec3 fN; // Normal vector
out vec3 fE; // Direction of the viewer
out vec3 fL; // Direction of the light

uniform mat4 P;
uniform mat4 V;
uniform mat4 M;
uniform vec4 LightPosition;
```

Shaders example – Phong

Vertex Shader – Part 2

```
void main()
{
   fN = vNormal;
   fE = vPosition.xyz;
   fL = LightPosition.xyz;
   if( LightPosition.w != 0.0 ) {
      fL = LightPosition.xyz - vPosition.xyz;
   }
   gl_Position =  P*V*M*vPosition;
}
```

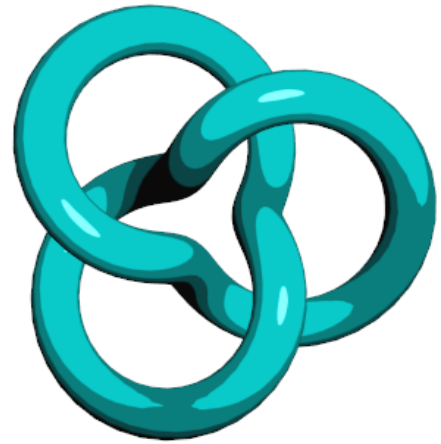
Shaders example – Phong

Fragment Shader

```
in vec3 fN;
in vec3 fL;
in vec3 fE;

// Light and material properties goes here...

out vec4 color;

void main()
{
  // Compute the illumination...

  color = ambient + diffuse + specular;
  color.a = 1.0; // Alpha value
}
```

Toon or Cel Shading

- Non-photorealistic shading
- Cartoon-like shading
- Illumination quantized to small number of discrete colors
- Usually together with contour outline



Contour outline

Two examples of methods

1. Render the back faces of a slightly enlarged model in the outline color
2. First normal and depth textures are generated from the rendered image. Then an edge detection filer (e.g., Sobel filter) is applied on the generated textures

Phong VS Toon Shading

Deferred Shading

- The rendering phase is done in two passes: *Geometry* and *Lighting* passes
- Lighting is postponed to the end of the rendering pipeline
- Used in games

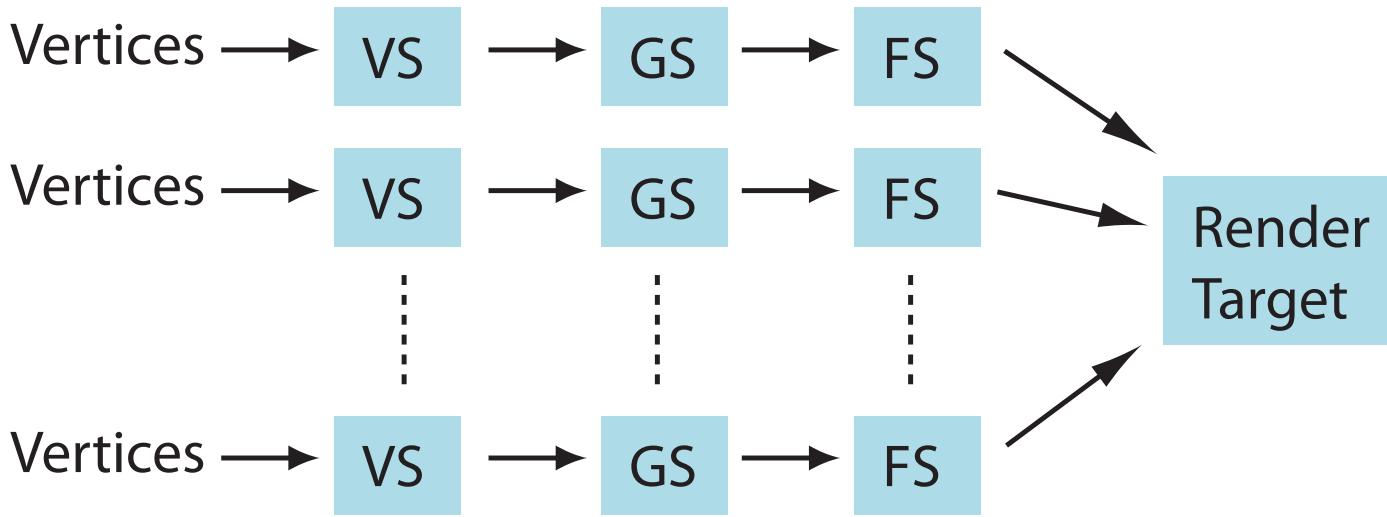
Deferred Shading

- The rendering phase is done in two passes: *Geometry* and *Lighting* passes
- Lighting is postponed to the end of the rendering pipeline
- Used in games

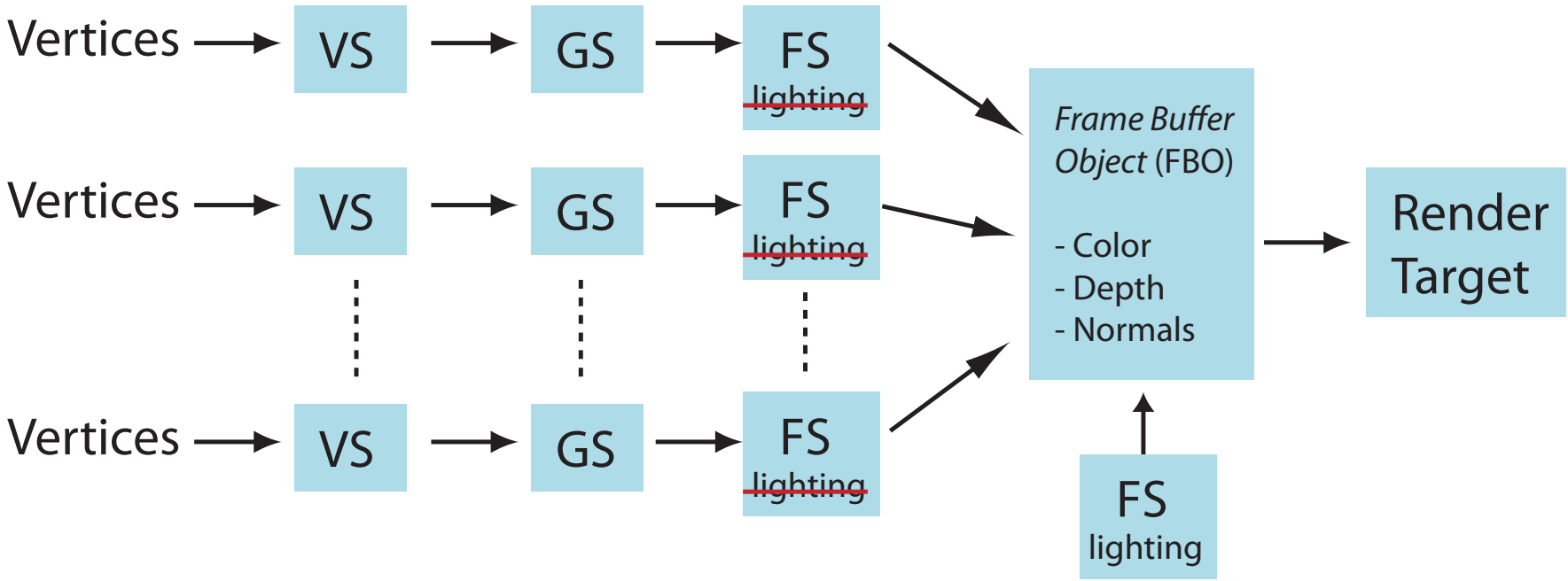
Advantages

- Only shade pixels that are visible
- Lighting is applied at pixel level rather than mesh level
- Can handle more and complex light sources resulting in natural lighting
- Reduced CPU usage

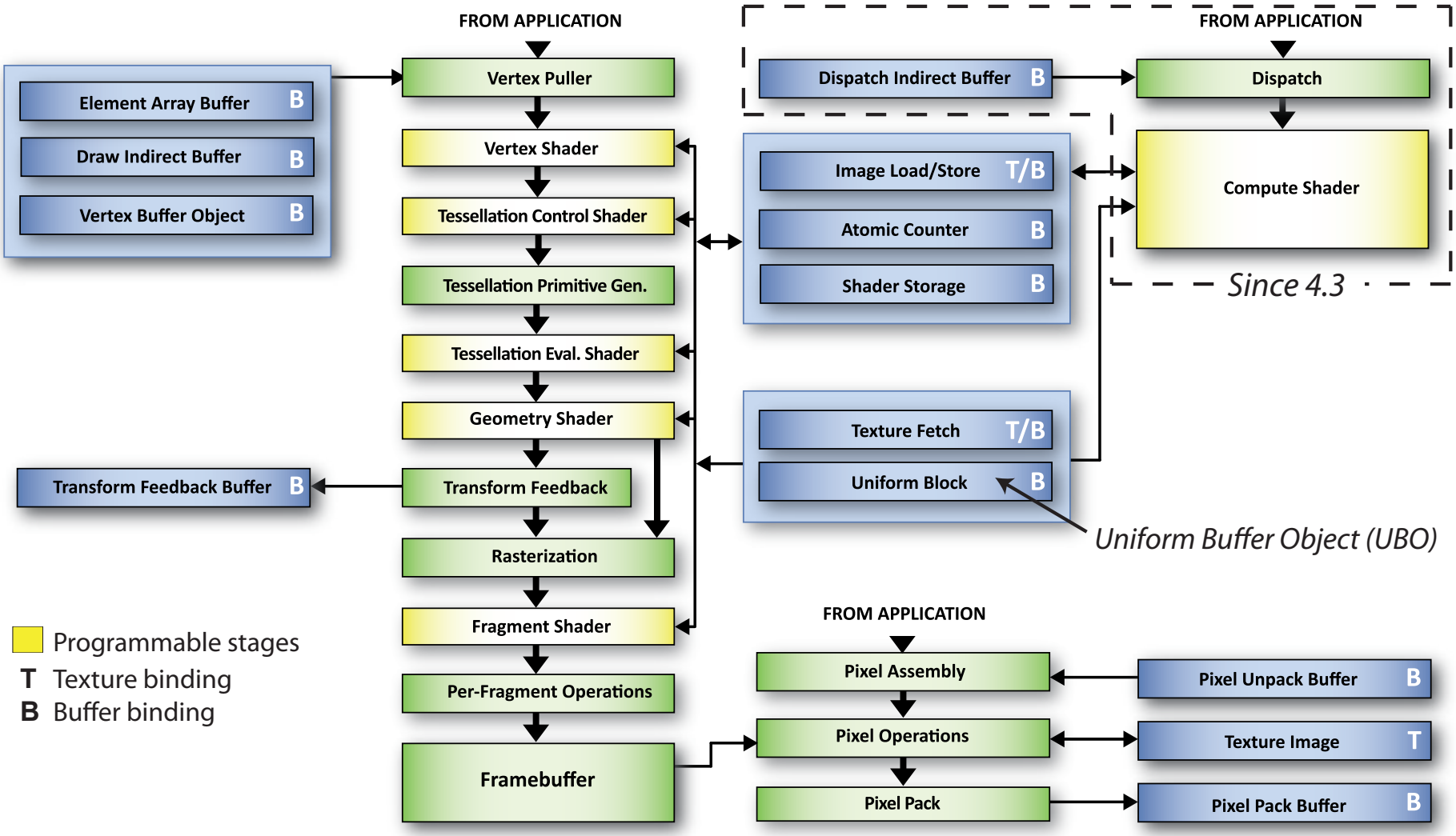
Forward rendering pipeline



Deferred rendering pipeline



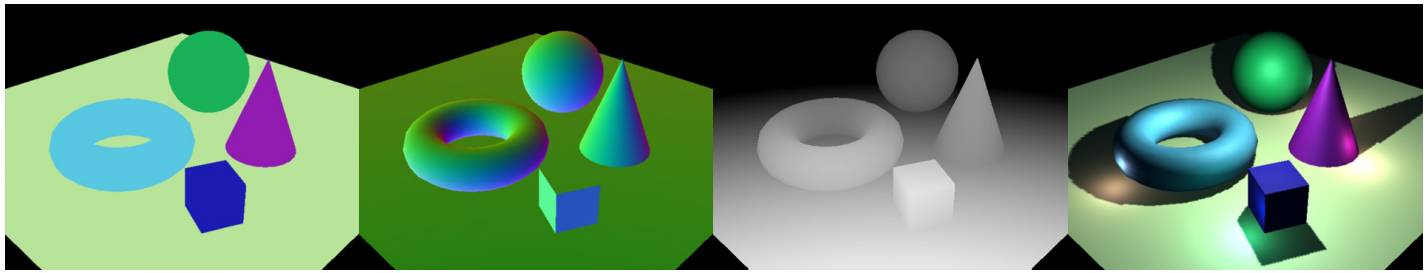
OpenGL Pipeline (revisited)



https://www.opengl.org/sdk/docs/reference_card/opengl45-reference-card.pdf

Deferred Shading

- First pass: Render (at least) three textures using *Multiple Render Targets* (MRT) (*Frame Buffer Objects* (FBO) in OpenGL)
  - Color texture with only diffuse
  - Color texture with normals in world space
  - Position (depth) texture
  - (and other attributes like textures and specular map)
- Stored in the *G-Buffer* and read in the lighting pass



Color Normals Depth Final result

Deferred Shading

Cons

- Hard to deal with transparencies
  - Must be rendered separately; one deferred pass and one forward pass
- Multiple materials require large amount of data and high memory bandwidth
  - Compression
- New anti-aliasing techniques are required
  - Edge detection
  - Multisample anti-aliasing (MSAA)
- High overhead; second pass and texture sampling
- The graphics card must support Multiple Render Targets

Forward VS Deferred Shading

S.T.A.L.K.E.R., GSC Game World (2004)

Deferred shading in Uncharted 4

Uncharted 4, Naughty Dog (2016)

Deferred shading in Uncharted 4

Uncharted 4, Naughty Dog (2016)

Shadows – An overview

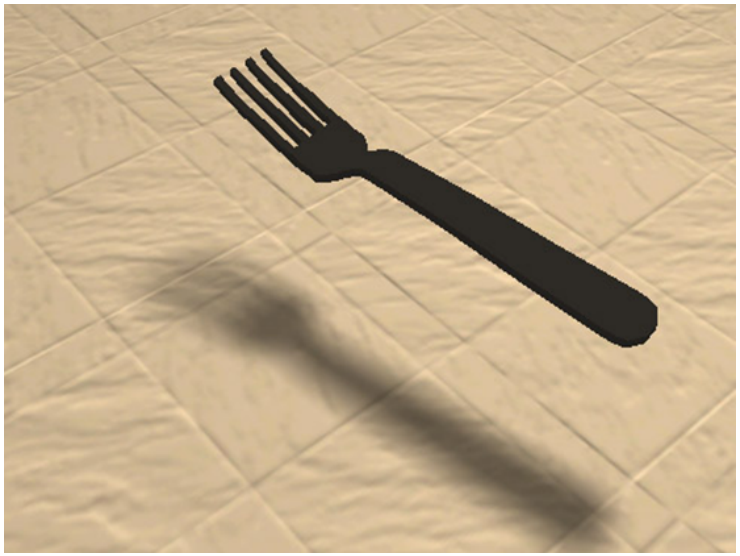
Dynamical (real-time):

- Shadow mapping
- Shadow volumes

An active research area

Static:

- Ray-Traced Shadows
- Area Shadows
- Lightmaps



Shadows – An overview

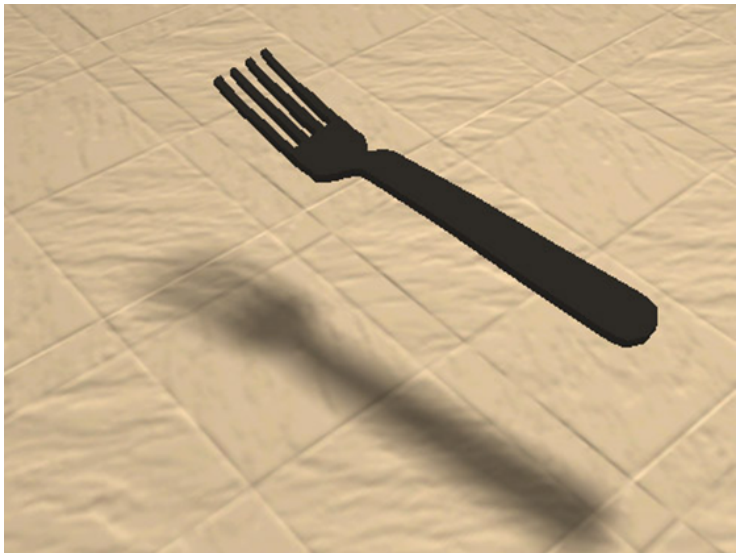
Dynamical (real-time):

- **Shadow mapping**
- Shadow volumes

An active research area

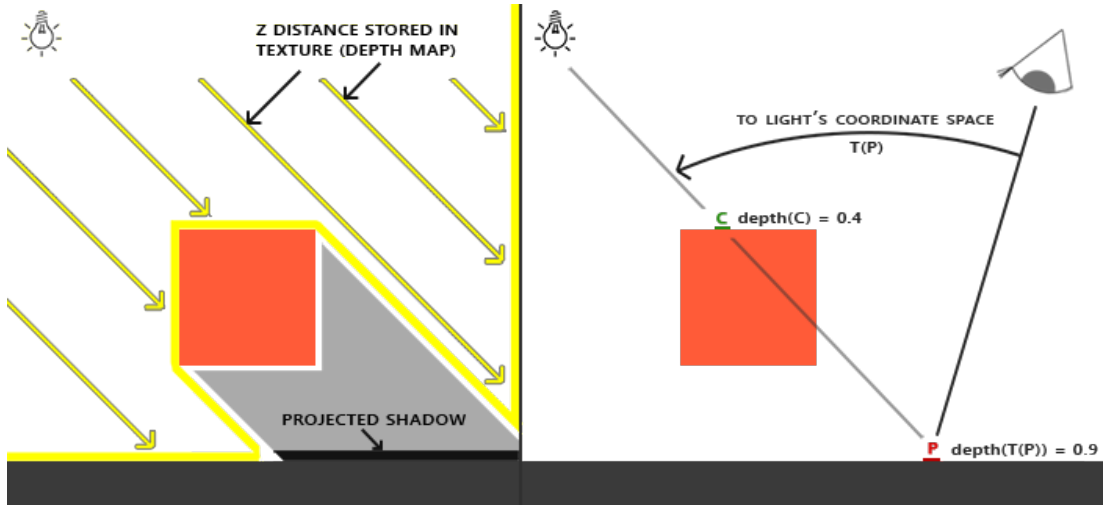
Static:

- Ray-Traced Shadows
- Area Shadows
- Lightmaps



Shadow mapping

- For realtime graphics (one of the major methods)
- Is a texture (called *depth map*) based approach
- Needs two passes:
  1. Render the scene from the light's point of view, generating the depth map
  2. Render as usual, but check for each fragment if it is in shadow or not



Shadow mapping – The process

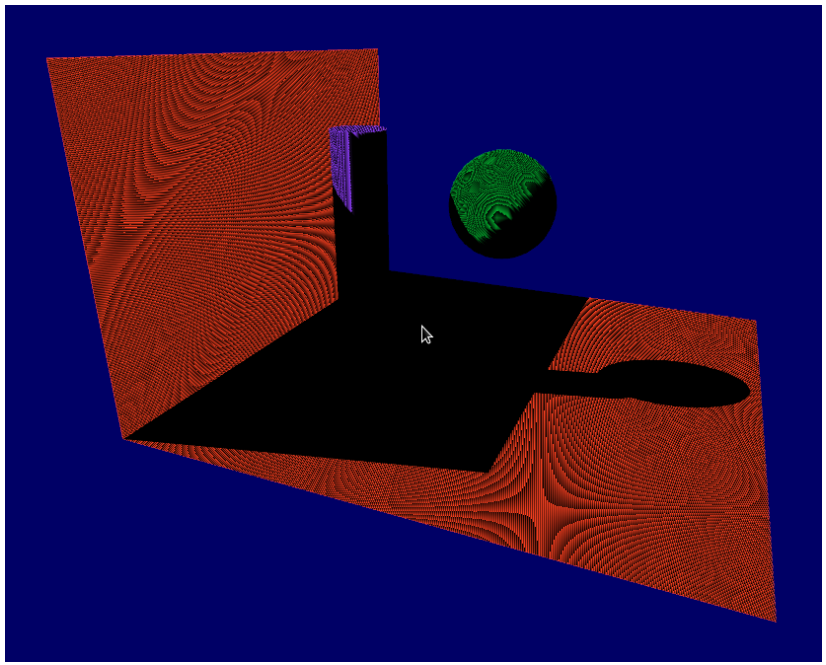
Pass one:

1. For each light in the scene
   1.1 Compute the scenes depth (in world coordinates) from the point of the light

Pass two:

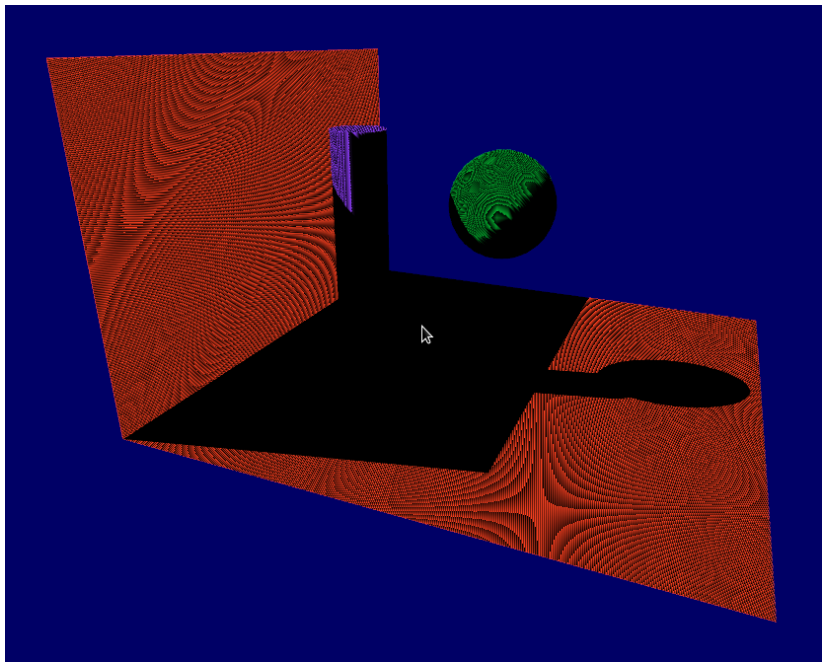
1. If current fragment is further from light than the shadow map
   1.1 Fragment is in shadow
2. otherwise
   2.1 Fragment is *not* in shadow

Shadow mapping – The result

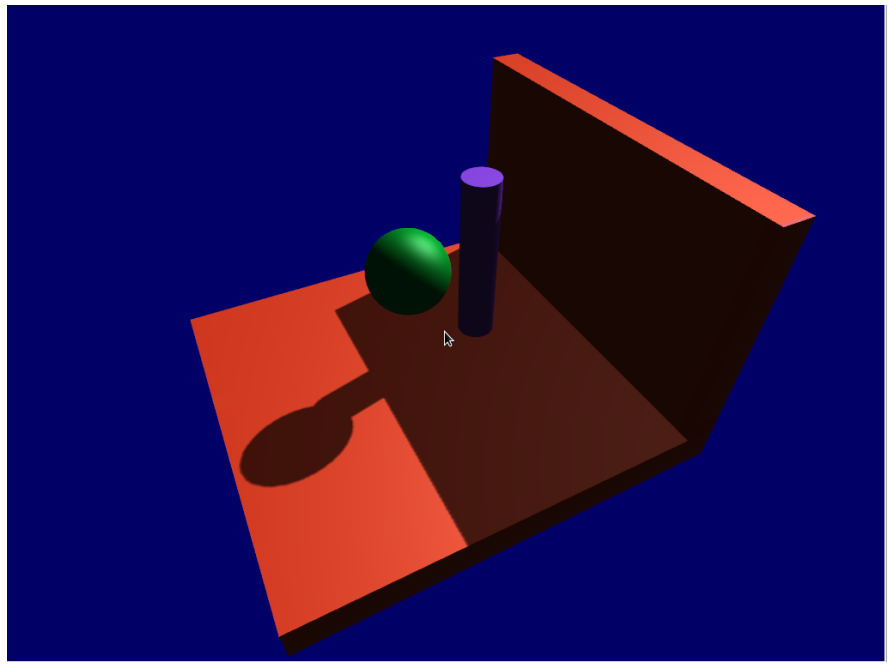


Out of the box

Shadow mapping – The result



Out of the box

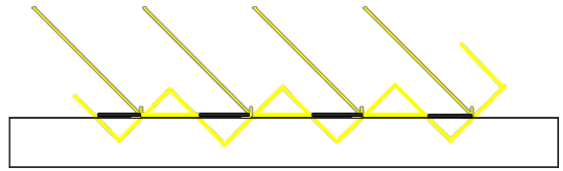


After some tweaking

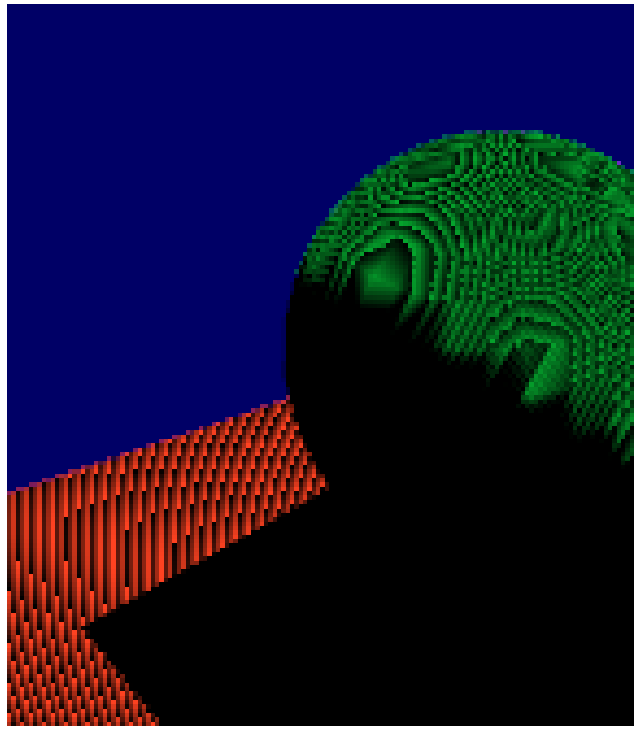
Shadow mapping – Problem: Acne

Shadow acne

– shown as a moiré pattern



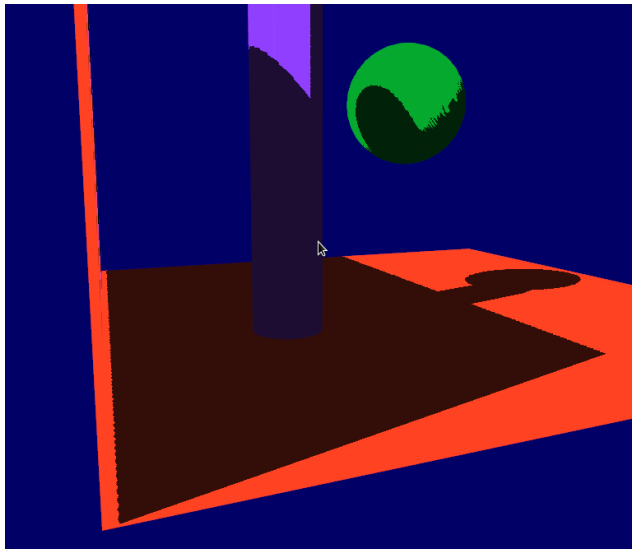
Usually solved by adding a slope based bias (offset) to the depth test



Shadow mapping - Problem: Peter Panning

Using a bias enhance the so called *Peter Panning* effect

- The shadow is detached from the object; Looks like the object is “flying”



Simple solution: Avoid thin geometries and use front face culling when computing the depth map

Shadow mapping – Problem: Resolution (Undersampling)

- Shadow far from the object has low resolution
- Low resolution depth map gives low resolution shadow
- High resolution depth map is memory demanding

Shadow mapping – Problem: Resolution (Undersampling)

- Shadow far from the object has low resolution
- Low resolution depth map gives low resolution shadow
- High resolution depth map is memory demanding

One solution is *Cascaded Shadow Maps*

- Render several depth maps
- The depth maps closer to the camera cover a small area with high resolution. Depth maps further away cover a large area with low resolution

Other techniques are *Trapezoidal Shadow Maps* and *Sample Distribution Shadow Maps*

Shadow mapping - Problem: Aliasing

- Hard shadows
- Shadows should get softer the further away from the object (blocker) they are
- Micro details produce no shadows (like antennas)

Shadow mapping - Problem: Aliasing

Solved by smoothing the shadow's edges depending on the distance to the blocker, e.g., by

- Percentage closer filtering
  (old technique (-87) but still widely used)
- Percentage closer soft shadows (PCSS)
- Variance shadow maps
- Exponential shadow maps
- Exponential variance shadow maps
- Moment soft shadow maps
- Hybrid frustum traced shadows

Shadow volumes

- The (only?) alternative to shadow maps
- Create a shadow geometry (volume)
- All objects inside the shadow volume are considered to be in shadow
- Major breakthrough with the Doom 3 engine
- Pros: Much more accurate than shadow maps
- Cons: Computational demanding