Shading in OpenGL

Review: Phong Model

- A simple model that can be computed rapidly
- Has three components
    - Diffuse
    - Specular
    - Ambient

Ambient+Diffuse+Specular Reflections

- Single light source

$$I = k_a I_a + k_d I_l (N \cdot L) + k_s I_l (R \cdot V)^n$$

- Multiple light source

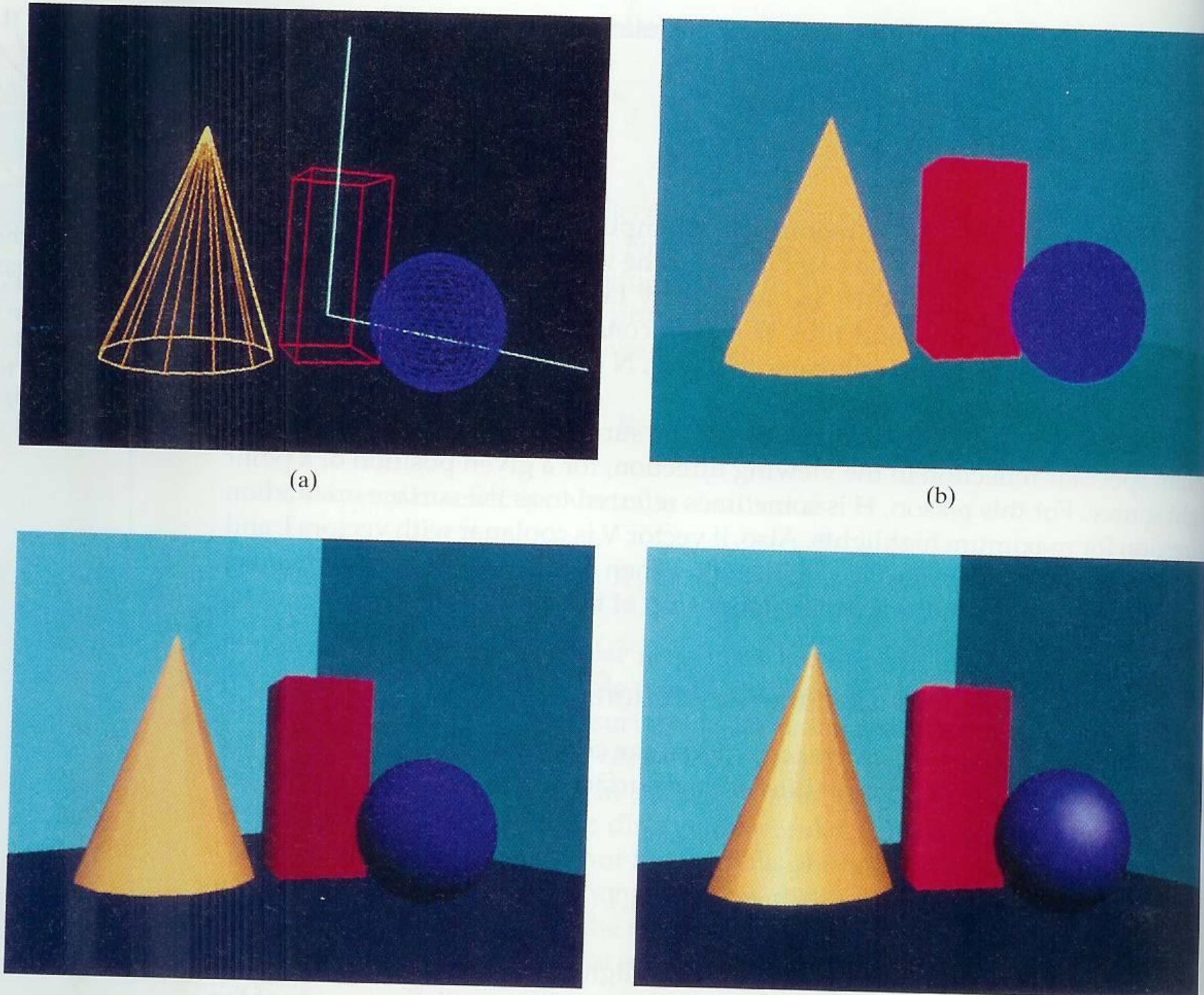
$$I = k_a I_a + \sum_l k_d I_l (N \cdot L) + k_s I_l (R \cdot V)^n$$

- Emission and attenuation

$$I = I_{emit} + k_a I_a$$
$$+ \sum_l f_{l,rad\_atten} f_{l,ang\_atten} \left(k_d I_l (N \cdot L) + k_s I_l (R \cdot V)^n \right)$$

Parameter Choosing Tips

- For a RGB color description, each intensity and reflectance specification is a three-element vector
- The sum of reflectance coefficients is usually smaller than one $k_a + k_d + k_s \leq 1$
- Try *n* in the range [0, 100]
- Use a small k_a (~0.1)
- Example
  - Metal: n=90, k_a=0.1, k_d=0.2, k_s=0.5



(a) (b) (c) (d)

Implementation with GLSL

OpenGL shading

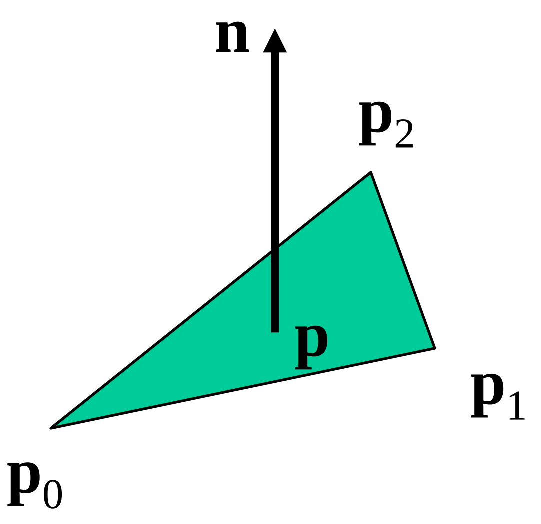
- Need
  - Normals
  - material properties
  - Lights
- State-based shading functions have been deprecated (glNormal, glMaterial, glLight)
- send attributes or uniforms to shaders

Normal for Triangle

plane $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$

normalize $\mathbf{n} \leftarrow \mathbf{n}/ |\mathbf{n}|$



Note that right-hand rule determines outward face

Polygon Rendering Methods

- Curved surfaces are often approximated by polygonal surfaces

- So, polygonal (piecewise planar) surfaces often need to be rendered as if they are smooth

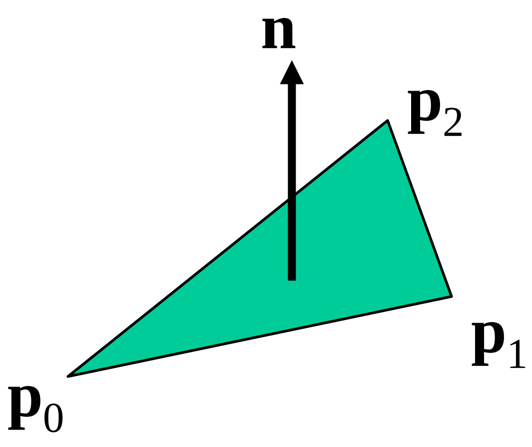
Flat shading

Smooth shading

Flat Shading

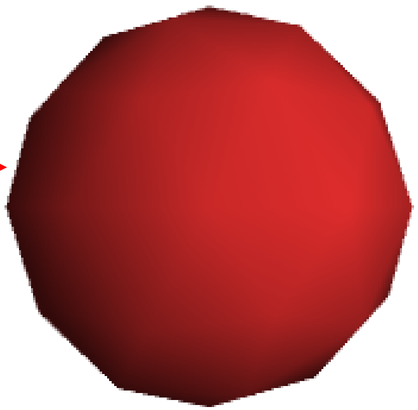
- We set a single normal for each triangle

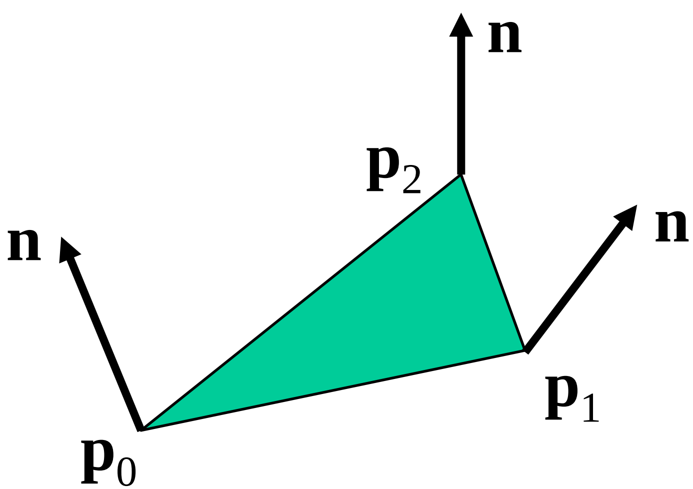
- Because three vertices of a triangle has the same normal, shades computed by the Phong model can be almost same



Smooth Shading

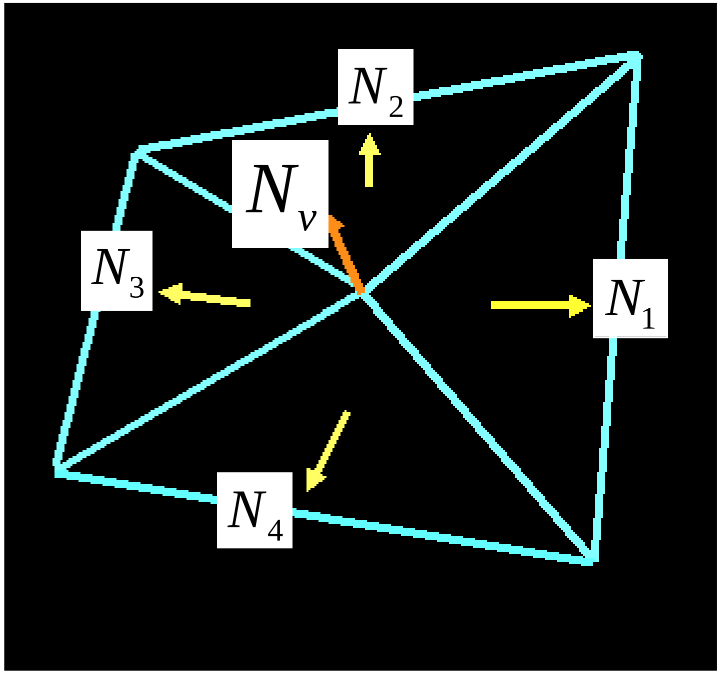
- We set a new normal at each vertex as if the polygon is the approximation of a smooth surface
- For a shpere model, it is easy
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Note *silhouette edge*





Vertex Normal Vector

- Normal vectors at vertices
  - Averaging the normal vectors for each polygon sharing that vertex



$$N_v = \frac{(N_1 + N_2 + N_3 + N_4)}{\left\| N_1 + N_2 + N_3 + N_4 \right\|}$$

Applying Phong Model in two different ways

- Applying Phong model *at each vertex*
  ➔ Gouroud Shading

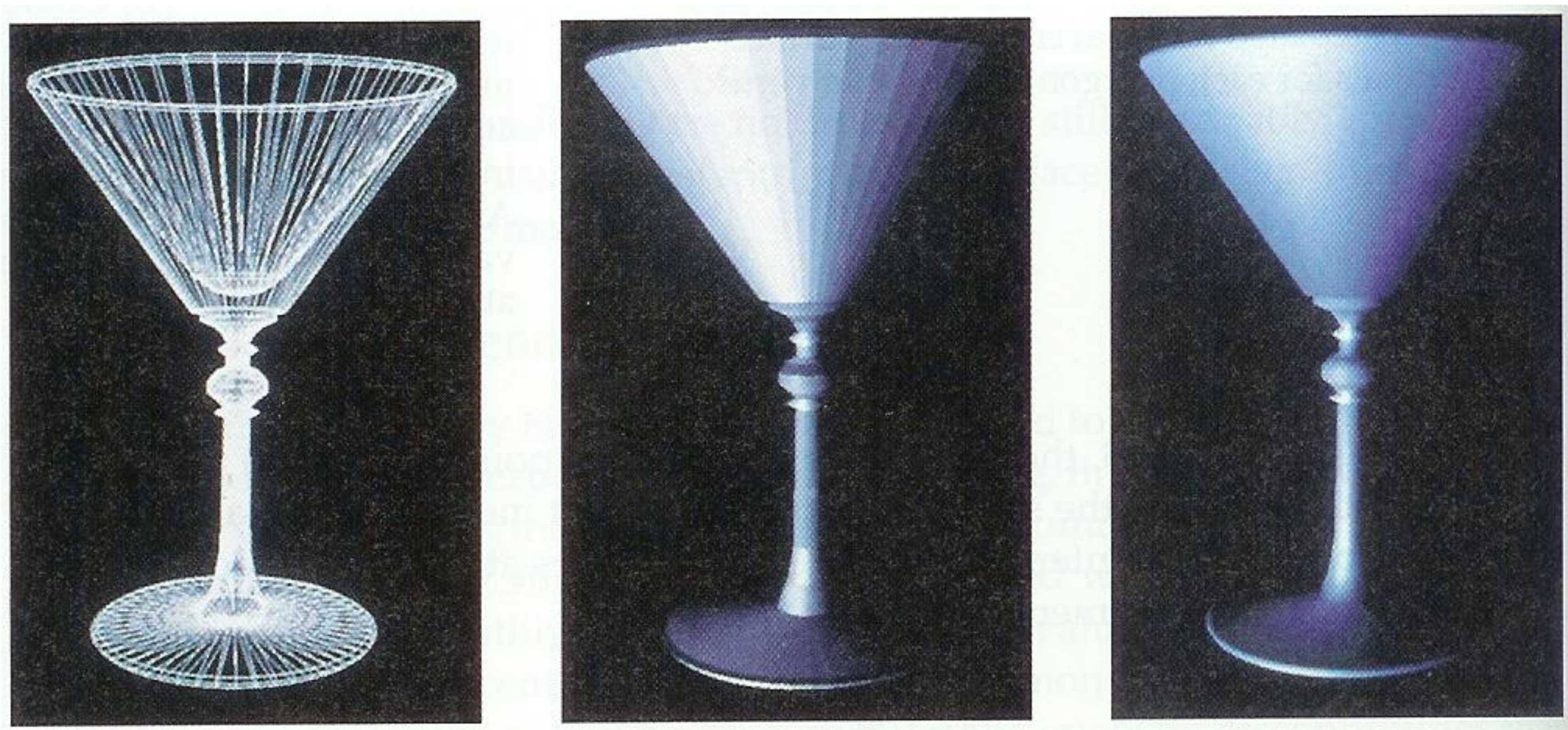
- Applying Phong model *at each fragment*
  ➔ Phong Shading

Intensity-Interpolation Surface Rendering

- **Gouraud shading**
  - Rendering a curved surface that is approximated with a polygon mesh
  - Interpolate intensities at polygon vertices

- Procedure
  1. Determine the average unit normal vector at each vertex
  2. Apply an illumination model at each polygon vertex to obtain the light intensity at that position
  3. Linearly interpolate the vertex intensities over the projected area of the polygon

Applying Phong Model at Vertices

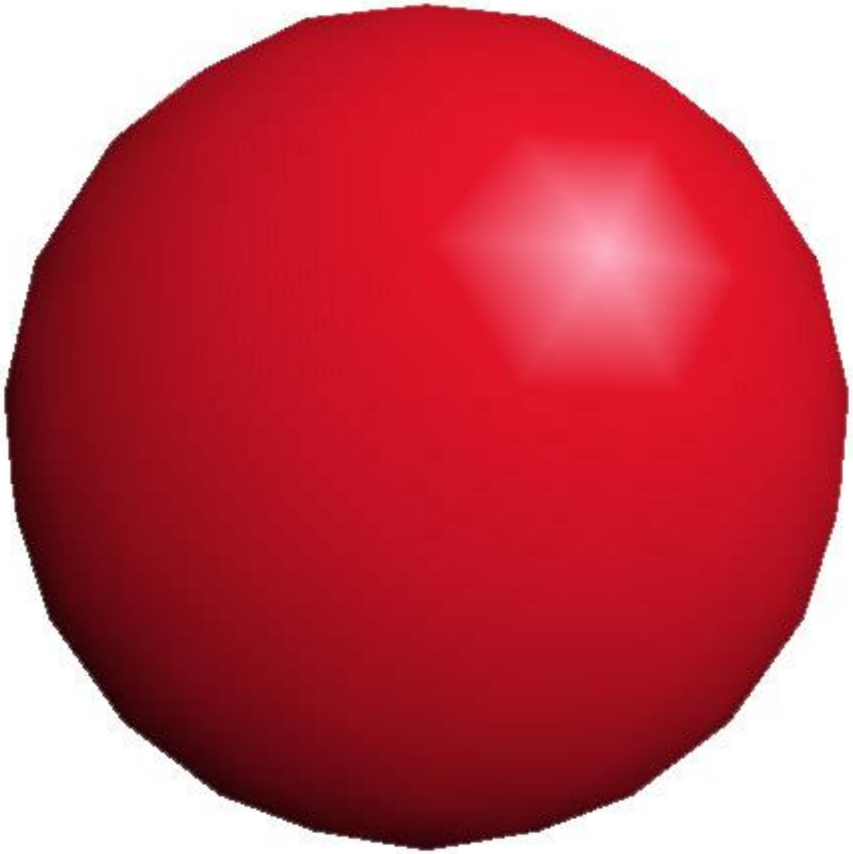


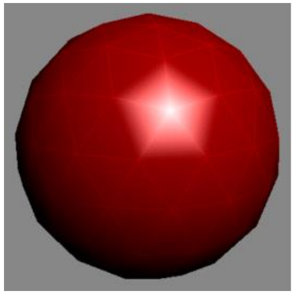
Applying Phong Model at Vertices

- In per vertex shading, shading calculations are done for each vertex
  - Vertex colors become vertex shades and can be sent to the vertex shader as a vertex attribute
  - Alternately, we can send the parameters to the vertex shader and have it compute the shade
- By default, vertex shades are interpolated across an object if passed to the fragment shader as a varying variable (smooth shading)
- We can also use uniform variables to shade with a single shade (flat shading)

Gouraud Shading Problems

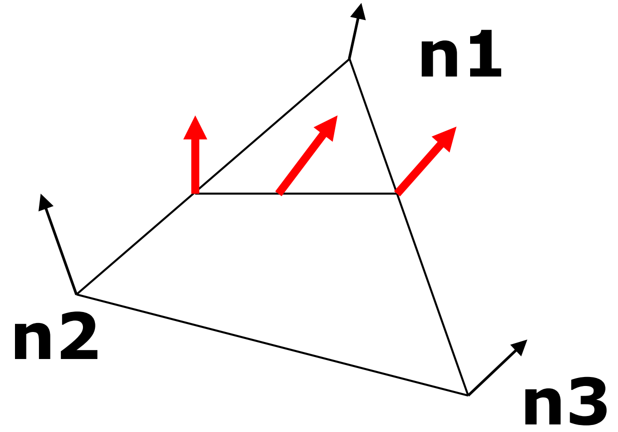
- Lighting in the polygon interior is inaccurate





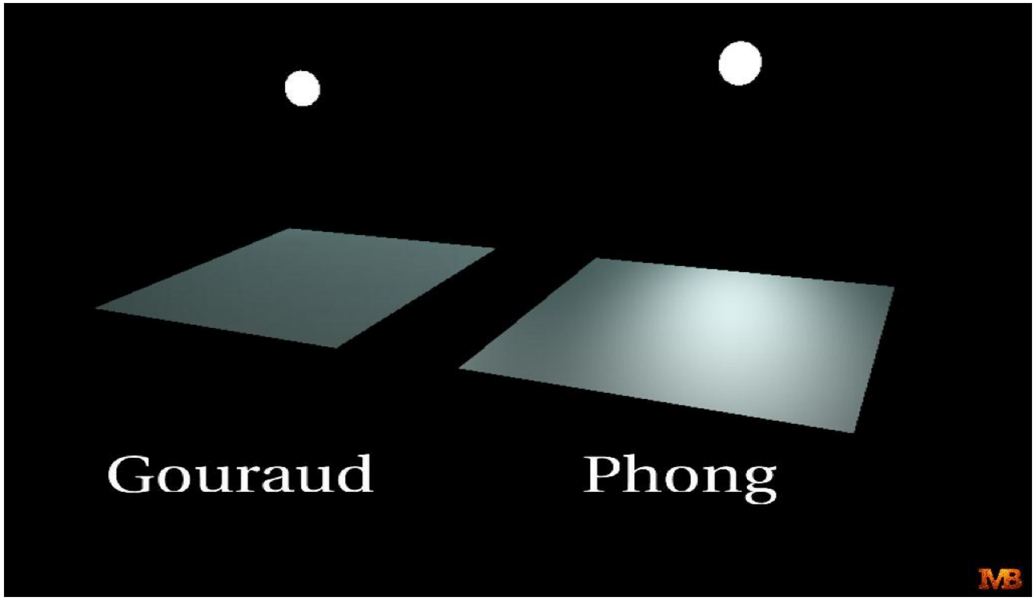
Normal-Vector Interpolation Surface Rendering

- **Phong shading**
  - Interpolate normal vectors at polygon vertices
- Procedure
  1. Determine the average unit normal vector at each vertex
  2. Linearly interpolate the vertex normals over the projected area of the polygon
  3. Apply an illumination model at positions along scan lines to calculate pixel intensities



Gouraud versus Phong Shading

- Gouraud shading is faster than Phong shading
  - OLD OpenGL supports Gouraud shading
- Phong shading is more accurate.
  - Can be implemented using Fragment shader



Gouraud and Phong Shading

- Gouraud Shading
  - Find average normal at each vertex (vertex normals)
  - **Apply Phong model at each vertex**
  - Interpolate vertex shades across each polygon

- Phong shading
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply **Phong model at each fragment**

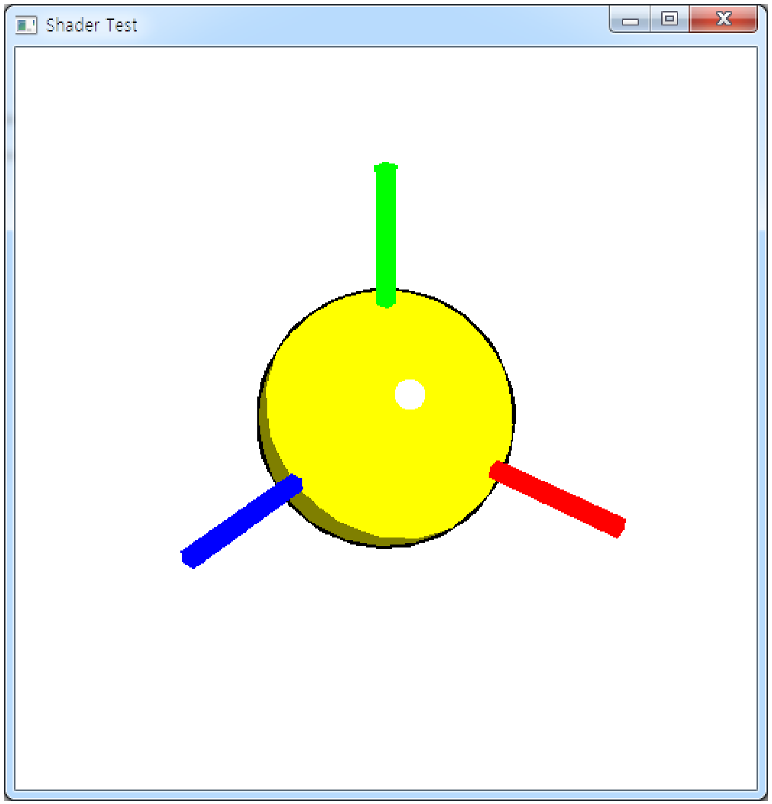
Comparison

- If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges

- Phong shading requires much more work than Gouraud shading
  - Until recently not available in real time systems
  - Now can be done using fragment shaders

- Both need data structures to represent meshes so we can obtain vertex normals

Simple Non-Photorealistic Rendering



Observation:

- Only Two Colors for diffuse
- One Color for Highlights

At Silhouette:

- Black line