LEC25: Implementing Phong Reflection Model-part3

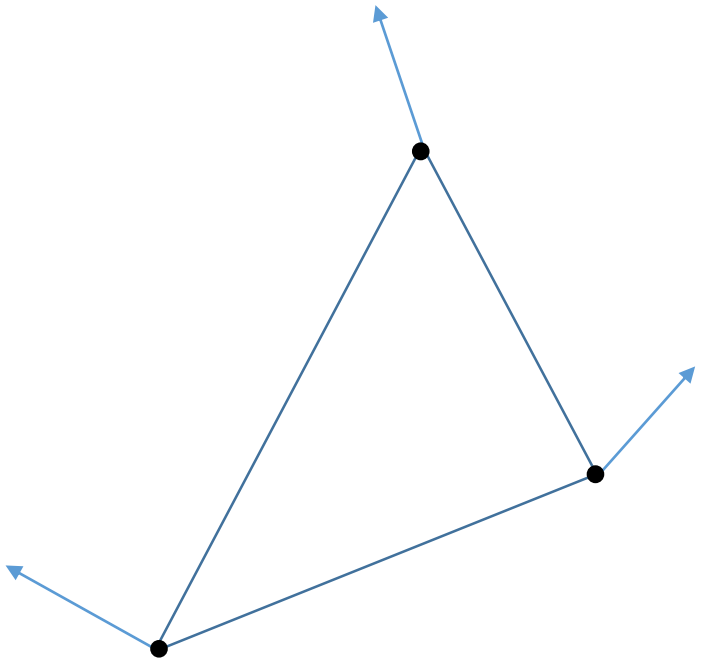
Ku-Jin Kim
School of Computer Science & Engineering
Kyungpook National University

Contents

- Gouraud Shading & Phong Shading
- Implementing Phong Shading

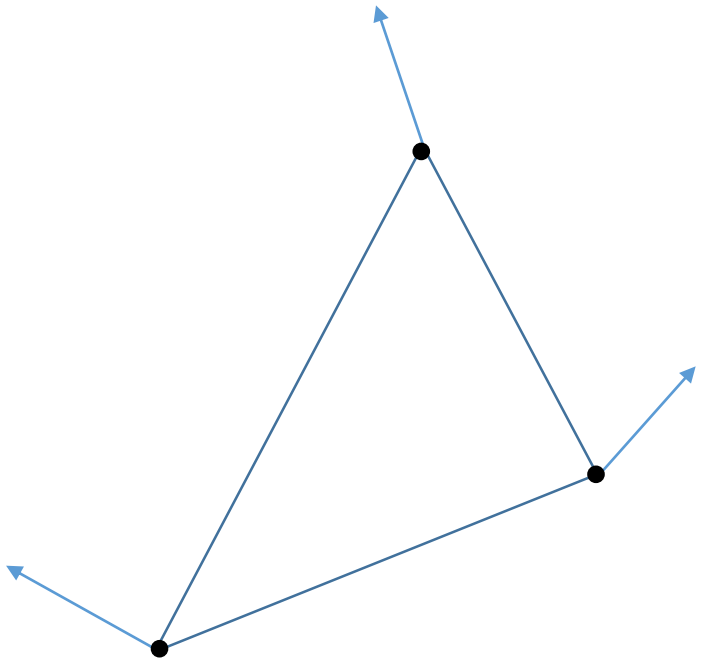
Gouraud and Phong Shading (1)

- Gouraud Shading
  - Given vertex normal at each vertex, apply modified Phong model at each vertex
  - Interpolate vertex shades across each polygon
  - per-vertex shading



Gouraud and Phong Shading (2)

- Phong shading
  - Given vertex normals, interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply modified Phong model at each fragment
  - per-fragment shading



Gouraud and Phong Shading (3)

- 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
  - A few years ago, it was not available in real time systems
  - It can be done using fragment shaders
- Both need data structures to represent meshes so we can obtain vertex normals

Phong Reflection Model & Phong Shading

- Phong shading is not the same with Phong Reflection model !!!
- Both of Phong shading and Gouraud shading use modified Phong reflection model for computing each vertex shade

Program Code for Phong shading with modified Phong model (1)

```
static char* vsSource = "#version 140 \n\
in vec4 aPosition; \n\
in vec4 aNormal; \n\
out vec4 vPosition; \n\
out vec4 vNormal; \n\
uniform mat4 uscale; \n\
uniform mat4 utranslate; \n\
uniform mat4 urotate; \n\
uniform mat4 uView; \n\
void main(void) { \n\
vPosition = uView * urotate * uscale * utranslate * aPosition; \n\
mat4 mNormal = transpose(inverse(uView* urotate * uscale * utranslate)); \n\
vNormal = mNormal * aNormal; \n\
gl_Position = vPosition; \n\
}";
```

Program Code for Phong shading with modified Phong model (2)

```
static char* fsSource = "#version 120 \n\
in vec4 vPosition; \n\
in vec4 vNormal; \n\
uniform vec4 light_position; \n\
uniform vec4 light_ambient; \n\
uniform vec4 light_diffuse; \n\
uniform vec4 light_specular; \n\
uniform vec4 light_att; \n\
uniform vec4 material_ambient; \n\
uniform vec4 material_diffuse; \n\
uniform vec4 material_specular; \n\
uniform float material_shineness; \n\
```

Program Code for Phong shading with modified Phong model (3)

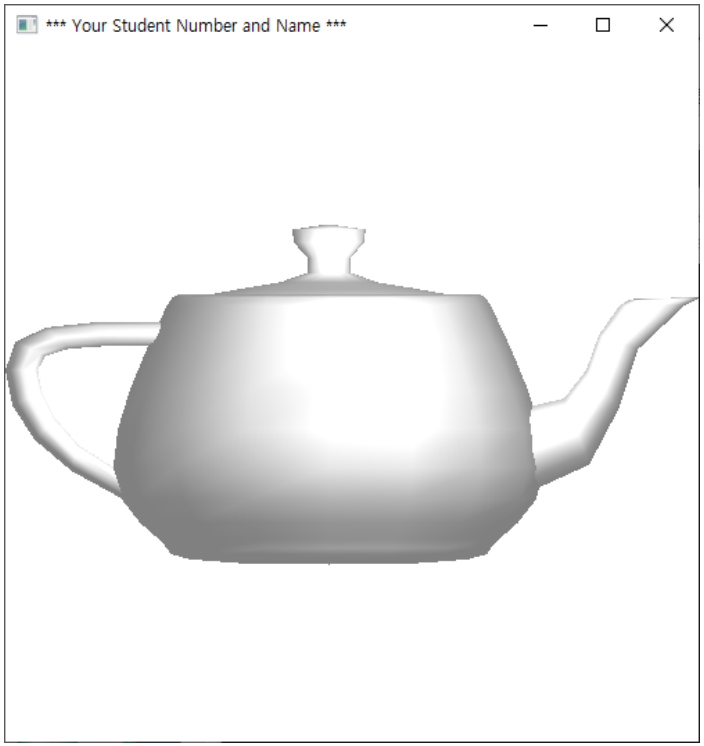
```
void main(void) { \n\
vec3 N = normalize(vNormal.xyz); \n\
vec3 L = normalize(light_position.xyz - vPosition.xyz); \n\
vec3 V = normalize(vec3(0.0, 0.0, 0.0) - vPosition.xyz); \n\
vec3 H = normalize (L + V); \n\
vec4 ambient = light_ambient * material_ambient; \n\
float d = length(light_position.xyz - vPosition.xyz); \n\
float denom = light_att.x + light_att.y * d + light_att.z * d * d; \n\
vec4 diffuse = max(dot(L, N), 0.0)*light_diffuse*material_diffuse / denom;\n\
vec4 specular = pow(max(dot(N, H), 0.0), material_shineness) * light_specular *
material_specular / denom; \n\
vec4 vColor = ambient + diffuse + specular; \n\
gl_FragColor = vColor; \n\
}";
```

Gouraud Shading vs. Phong Shading

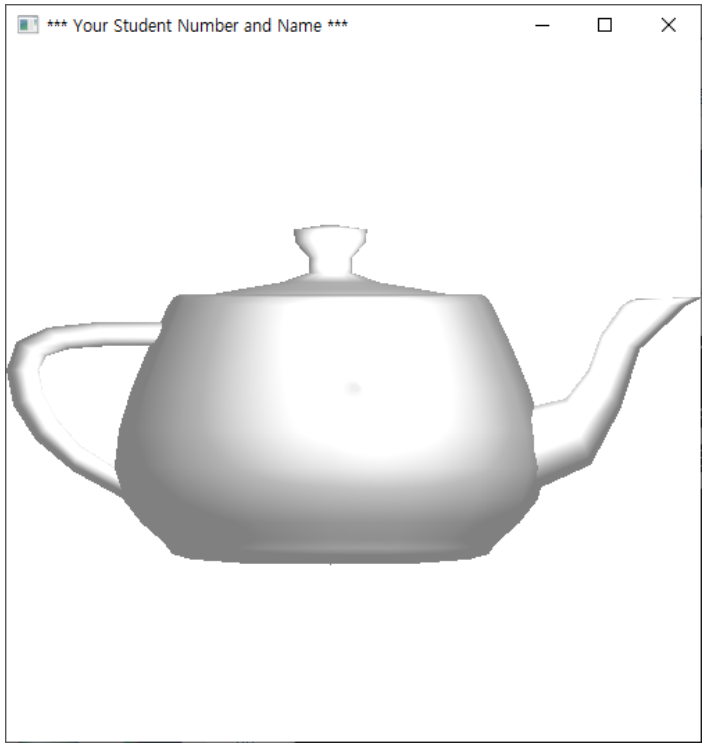
```
GLfloat light_pos[4] = { 0.5, 1.0, -2.0, 1.0 };
GLfloat light_amb[4] = { 0.5, 0.5, 0.5, 1.0 };
GLfloat light_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_spe[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] = { 0.0, 1.0, 0.0, 1.0 };
```

```
GLfloat mat_amb[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_spe[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shi = 1;
```

Gouraud Shading
(per-vertex shading)



Phong Shading
(per-fragment shading)

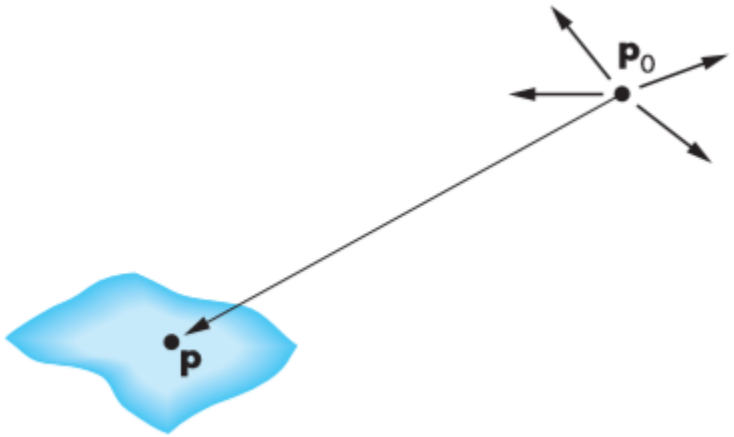


Computing Phong Reflection Model

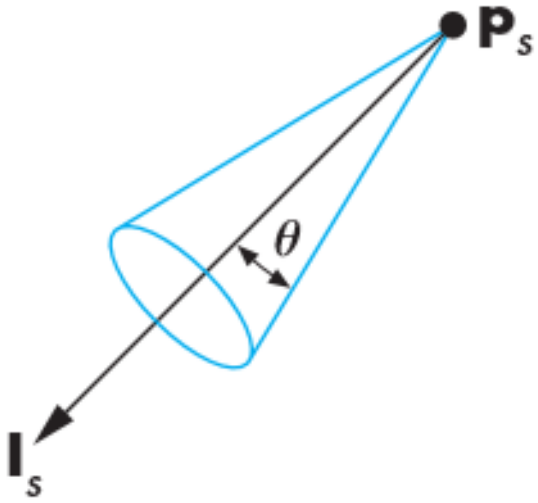
- R, G, B intensity we can see at **p** from light source i
  - $I_{ir} = I_{ira} + I_{ird} + I_{irs}$
  - $I_{ig} = I_{iga} + I_{igd} + I_{igs}$
  - $I_{ib} = I_{iba} + I_{ibd} + I_{ibs}$
- Total intensity at **p**
  - $I_r = \sum_i (I_{ir}) + I_{ar}$ (I_{ar}, I_{ag}, I_{ab} : global ambient term)
  - $I_g = \sum_i (I_{ig}) + I_{ag}$
  - $I_b = \sum_i (I_{ib}) + I_{ab}$

Revisit Simple Light Source

Point light source



Spot light source



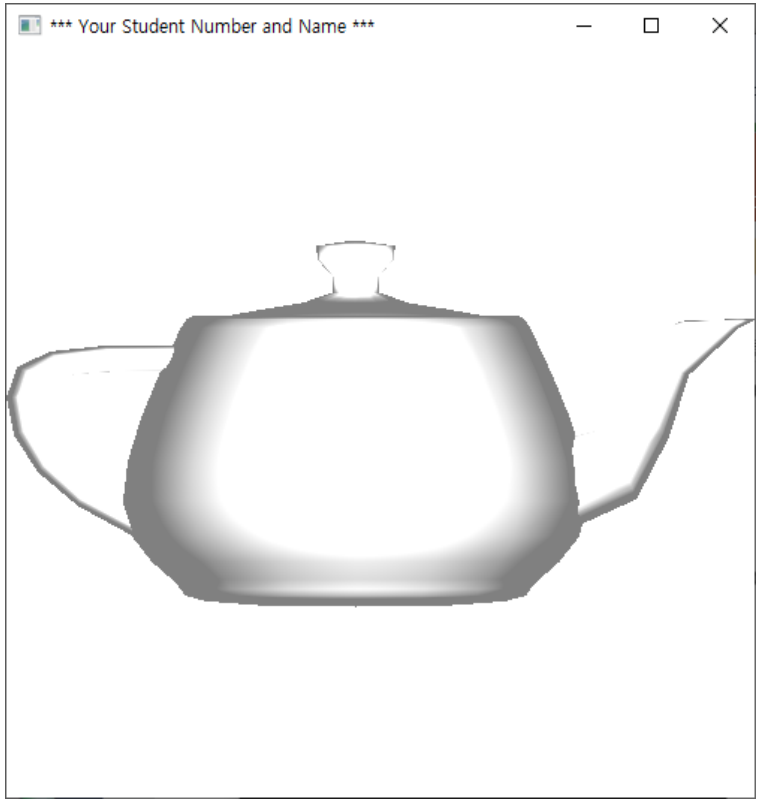
Ambient light source

Point Light Source

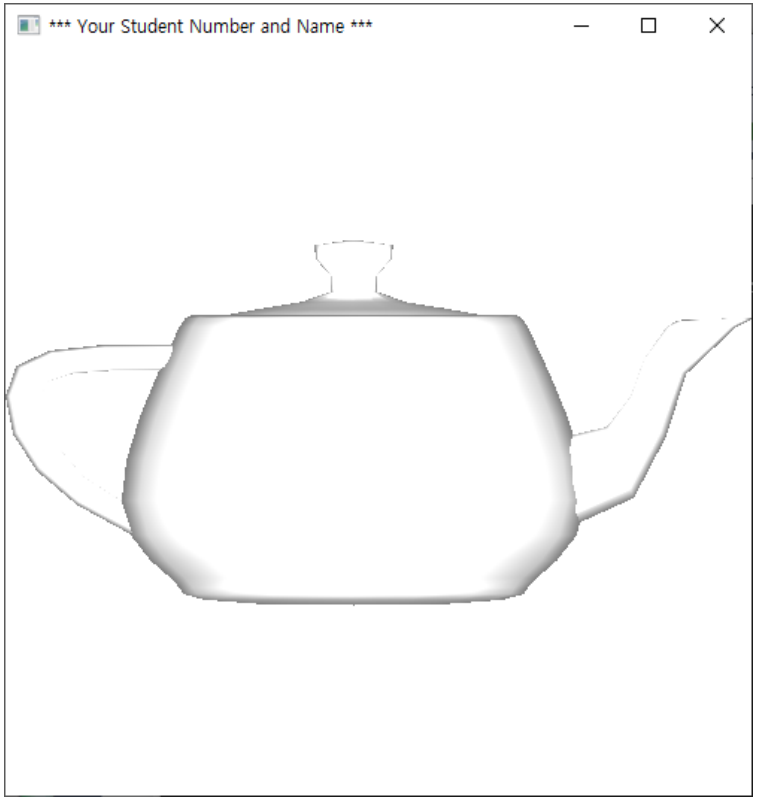
- The light source colors are specified in RGBA
- The light position is given in homogeneous coordinates
  - If w =1.0, we are specifying a finite location
    - vec3 L = normalize(light_position.xyz - vPosition.xyz);
  - If w =0.0, we are specifying a parallel light source with the given direction vector
    - vec3 L = normalize(light_direction.xyz_);
    - attenuation term can not be used

Program Execution Result

Point Light
GLfloat light_pos[4] = { 0.0, 0.0, -1.0, 1.0 };
GLfloat light_att[4] = { 1.0, 0.0, 0.0, 1.0 };

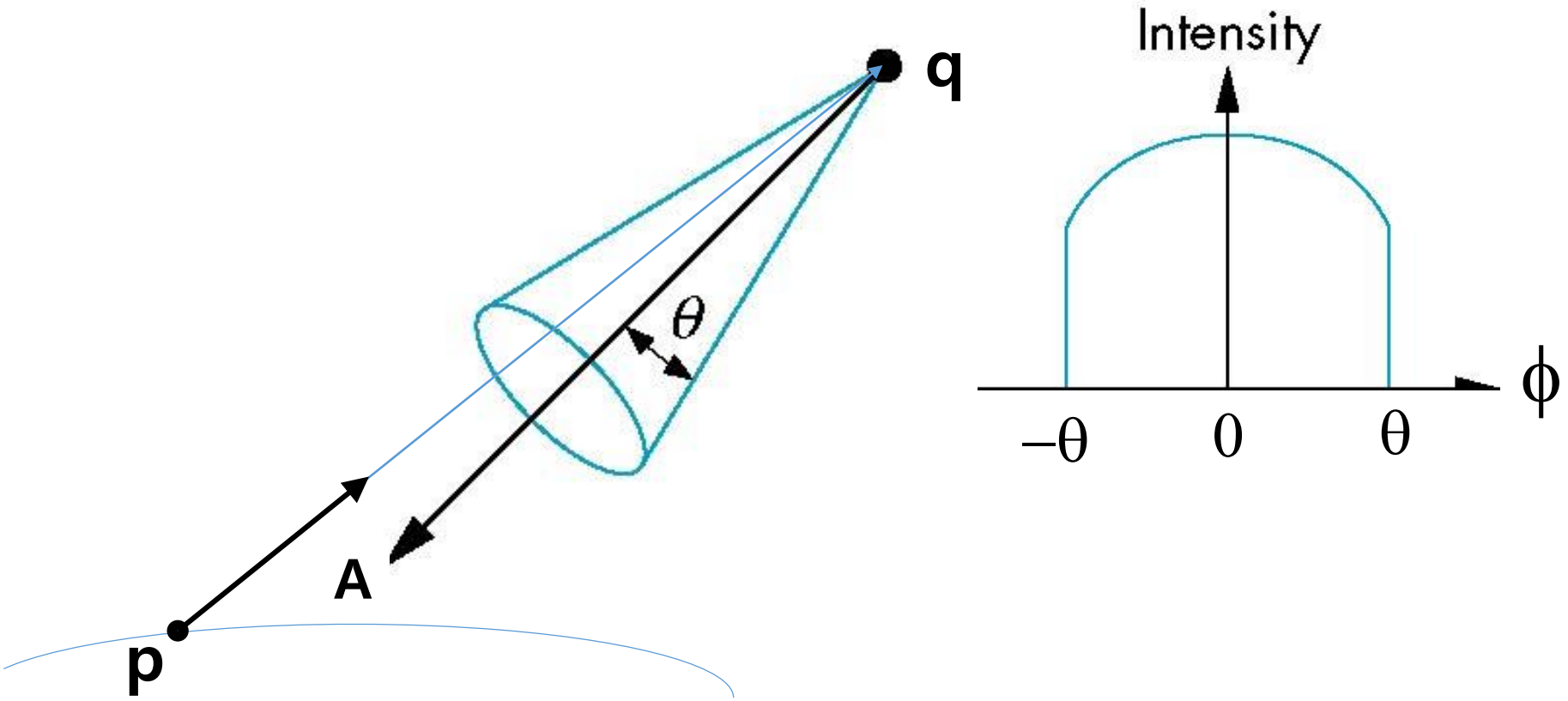


Parallel Light
GLfloat light_pos[4] = { 0.0, 0.0, 1.0, 0.0 };



Spotlights

- Representation:
  - Vertex **q,** cone axis **A,** cutoff angle θ
  - Attenuation proportional to $\cos^{\alpha}\phi$



Moving Light Sources

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix
- Depending on where we place the position (direction) setting function, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently

Transparency

- Material properties are specified as RGBA values
- The A value can be used to make the surface translucent
- The default is that all surfaces are opaque regardless of A
- A can be used for blending

glPolygonMode

- void glPolygonMode(GLenum face, GLenum mode);
  - face: specifies the polygons that mode applies to
    - GL_FRONT_AND_BACK
  - mode: specifies how polygons will be rasterized
    - GL_FILL, GL_POINT, GL_LINE

HW #24 Watch SIGGRAPH Trailer

- Watch following videos
  - SIGGRAPH 2020 Technical Papers Trailer
    - https://www.youtube.com/watch?v=jYdMKdRUq_8
  - SIGGRAPH Asia 2020 Technical Papers Trailer
    - https://www.youtube.com/watch?v=Q45KT0lGd7A
  - and other videos you can find by keyword SIGGRAPH.