LEC23: Implementing Phong Reflection Model-part1

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

- Modeling ambient and diffuse terms for Phong reflection model
- Program code

Polygon Mesh Model Shading

- 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 (smooth shading)
- We can also fill the triangle with a single shade (flat shading)

Flat Shading

- Triangles have a single Face normal
  - Each face normal generates the same shade for each face
- Want different normals at each vertex

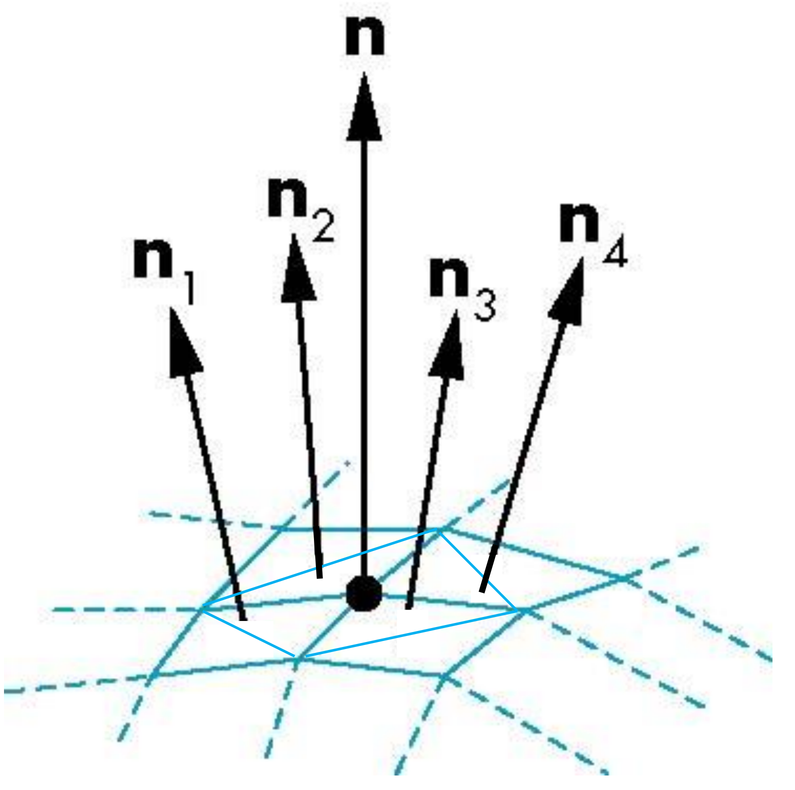
Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note silhouette edge

Vertex Normals for Mesh Shading

- It is not general that we know the normal at each vertex analytically
- For polygonal models, vertex normal can be computed as the average of the face normals of the 1-ring neighbors of a vertex

$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\|\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4\|$$



Phong Reflection Model

- Light source Terms
  - ambient
  - diffuse
  - specular
- Material reflection terms
  - ambient
  - diffuse
  - specular

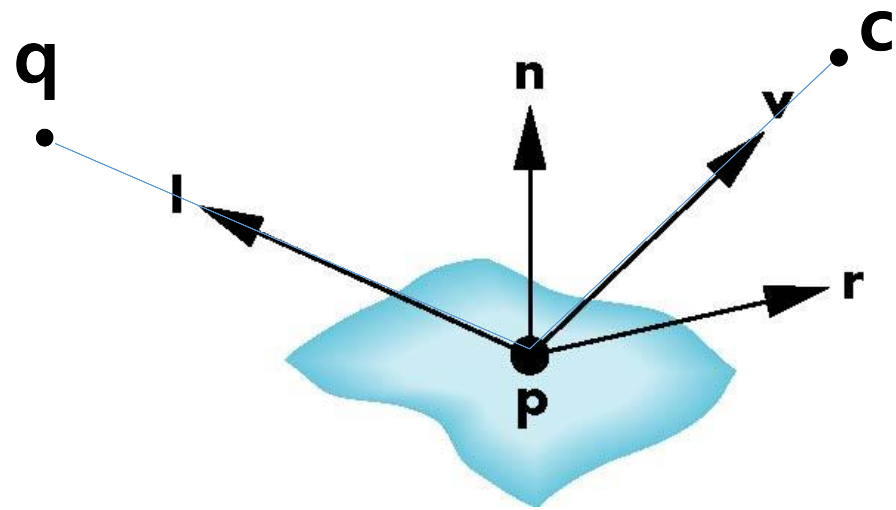
Light Source Terms

- In the Phong Model, we add the results from each light source
- Each light source has separate diffuse, specular, and ambient terms to allow for maximum flexibility even though this form does not have a physical justification
- Separate red, green and blue components
- Hence, 9 coefficients for each point source
  - red - L_{ar}, L_{dr}, L_{sr}
  - green - L_{ag}, L_{dg}, L_{sg}
  - blue - L_{ab}, L_{db}, L_{sb}

Material Reflection Terms

- Material properties match light source properties
  - Nine reflection coefficients
    - red - k_{ar}, k_{dr}, k_{sr}
    - green - k_{ag}, k_{dg}, k_{sg}
    - blue - k_{ab}, k_{db}, k_{sb}
  - Shininess coefficient α

Phong Reflection Model Vectors



- Input:
  - Point: **p**
  - Camera position: **c**
  - Light source position: **q**

- Derives four vectors at **p**
  - normal vector : **n**
  - light vector : $\mathbf{l} = (\mathbf{q}-\mathbf{p}) / \|\mathbf{q}-\mathbf{p}\|$
  - view vector : $\mathbf{v} = (\mathbf{c}-\mathbf{p}) / \|\mathbf{c}-\mathbf{p}\|$
  - reflection vector : $\mathbf{r} = 2(\mathbf{l} \cdot \mathbf{n})\, \mathbf{n} - \mathbf{l}$

  - l, v, n, r are used after normalization
    - $\| \mathbf{l} \| = \| \mathbf{v} \| = \| \mathbf{n} \| = \| \mathbf{r} \| = 1$

Modeling Ambient Reflection

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the object
- Represented by $k_a \, L_a$

reflection coefficient　　　　intensity of ambient light

Program Example (1)

```
static char* vsSource = "#version 130 ₩n₩
in vec4 aPosition; ₩n₩
in vec4 aNormal; ₩n₩
out vec4 vColor; ₩n₩
uniform mat4 uscale; ₩n₩
uniform mat4 utranslate; ₩n₩
uniform vec4 light_ambient; ₩n₩
uniform vec4 material_ambient; ₩n₩
void main(void) { ₩n₩
        vec4 ambient = light_ambient * material_ambient; ₩n₩
        vColor = ambient; ₩n₩
        gl_Position = uscale * utranslate * aPosition; ₩n₩
}";

static char* fsSource = "#version 130 ₩n₩
in vec4 vColor; ₩n₩
void main(void) { ₩n₩
        gl_FragColor = vColor; ₩n₩
}";
```

Program Example (2)

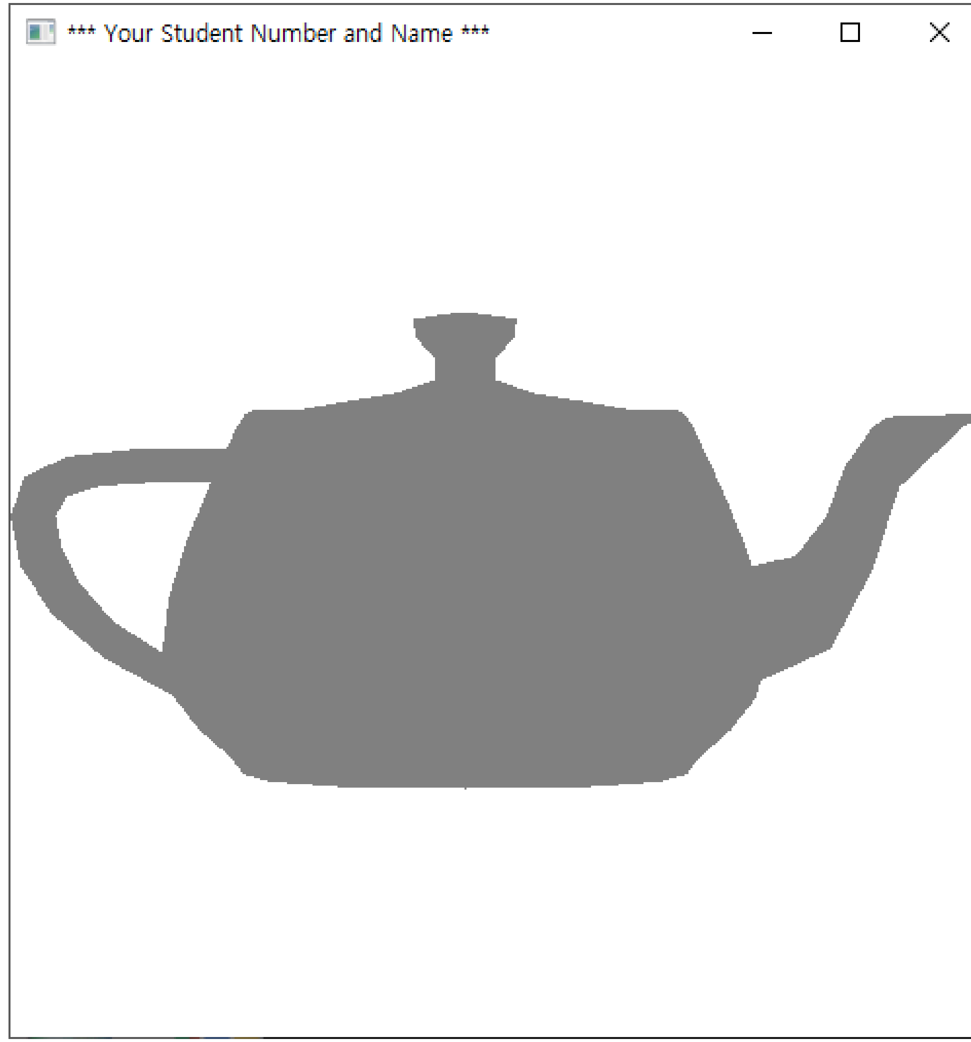
```
void setLightAndMaterial(void) {
        GLuint loc;
        GLfloat light_amb[4] = { 0.5, 0.5, 0.5, 1.0 };
        GLfloat mat_amb[4] = { 1.0, 1.0, 1.0, 1.0 };

        loc = glGetUniformLocation(prog, "light_ambient");
        glUniform4fv(loc, 1, light_amb);

        loc = glGetUniformLocation(prog, "material_ambient");
        glUniform4fv(loc, 1, mat_amb);
}
```

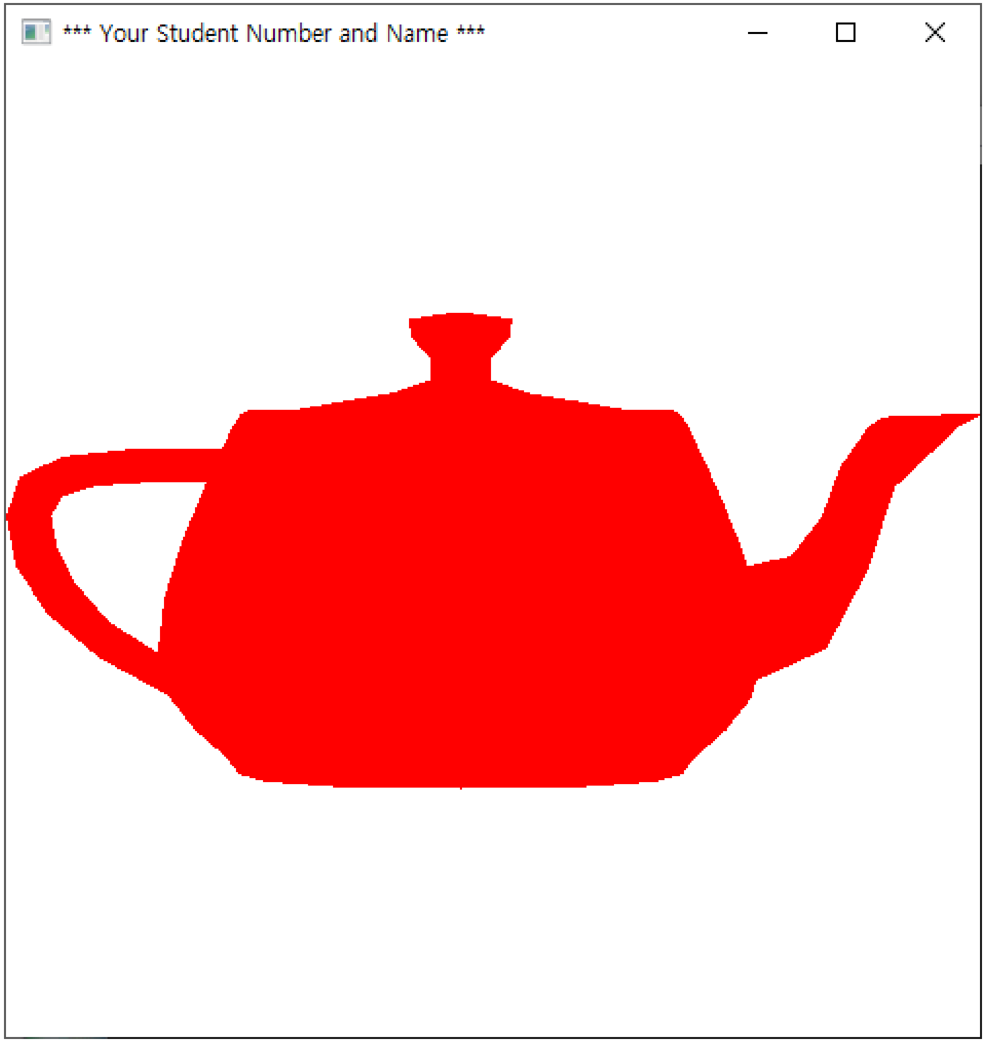
Ambient Reflection (1)

GLfloat light_amb[4] =
{ 0.5, 0.5, 0.5, 1.0 };
GLfloat mat_amb[4] =
{ 1.0, 1.0, 1.0, 1.0 };



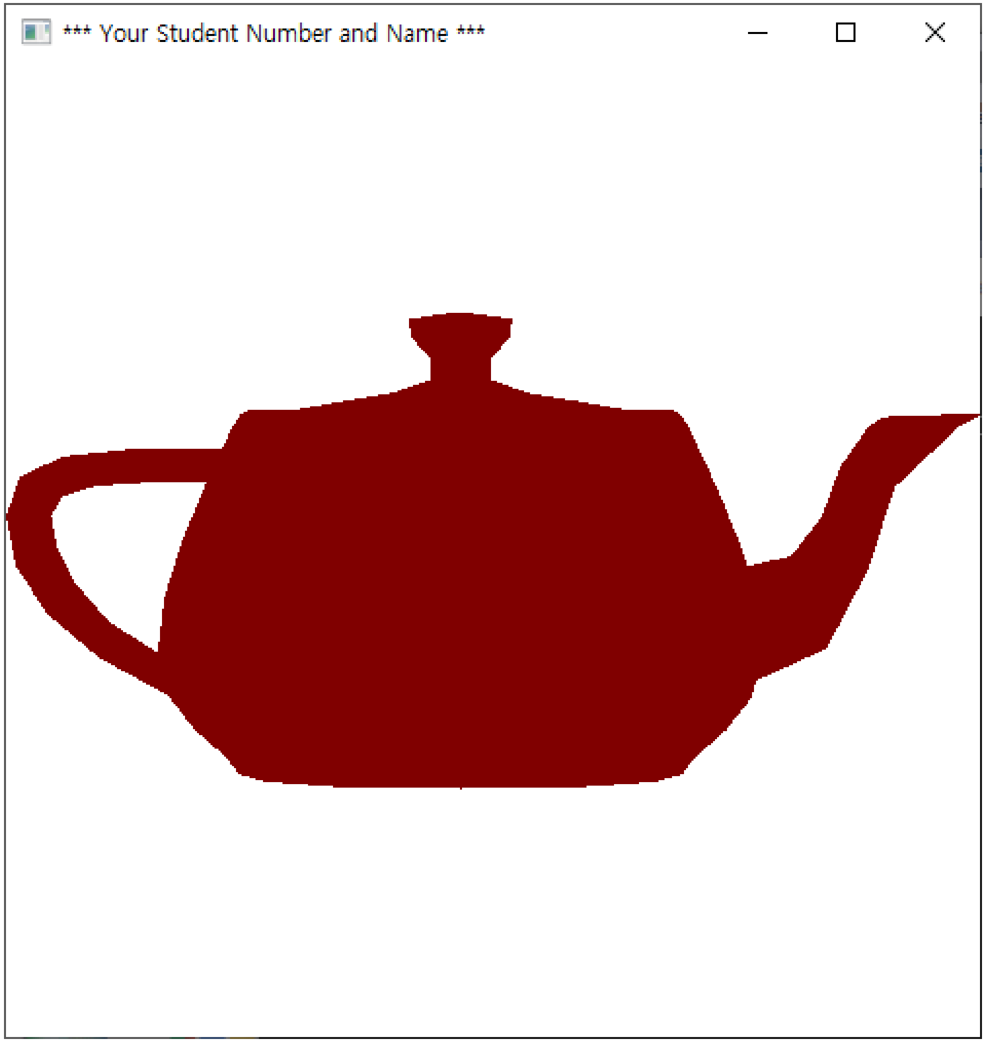
Ambient Reflection (2)

GLfloat light_amb[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_amb[4] = { 1.0, 0.0, 0.0, 1.0 };



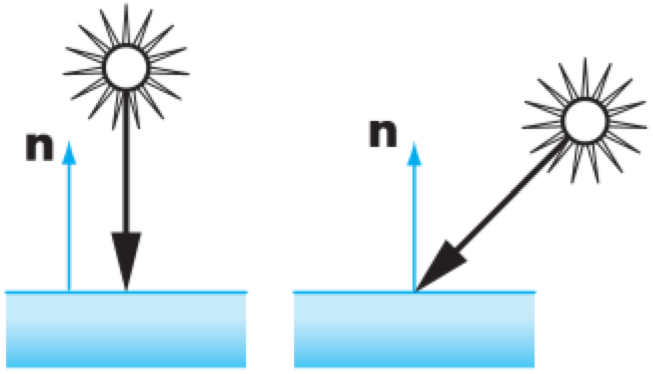
Ambient Reflection (3)

GLfloat light_amb[4] =
{ 0.5, 0.5, 0.5, 1.0 };
GLfloat mat_amb[4] =
{ 1.0, 0.0, 0.0, 1.0 };



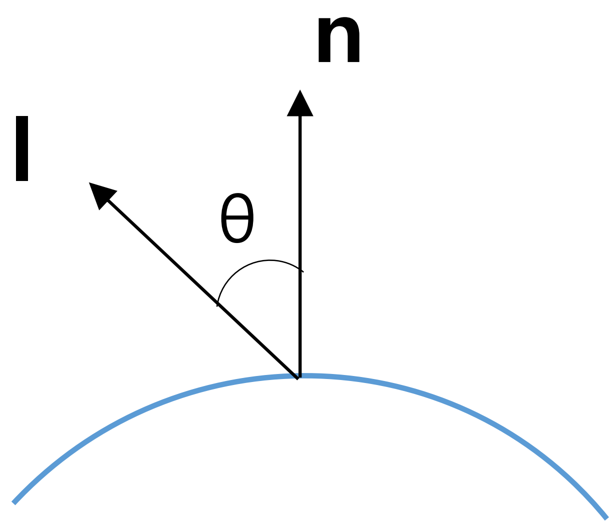
Modeling Diffuse Reflection (1)

- reflected light $\sim\cos\theta_i$
- $\cos\theta_i = \mathbf{l}\cdot\mathbf{n}$ if **l** and **n** vectors are normalized



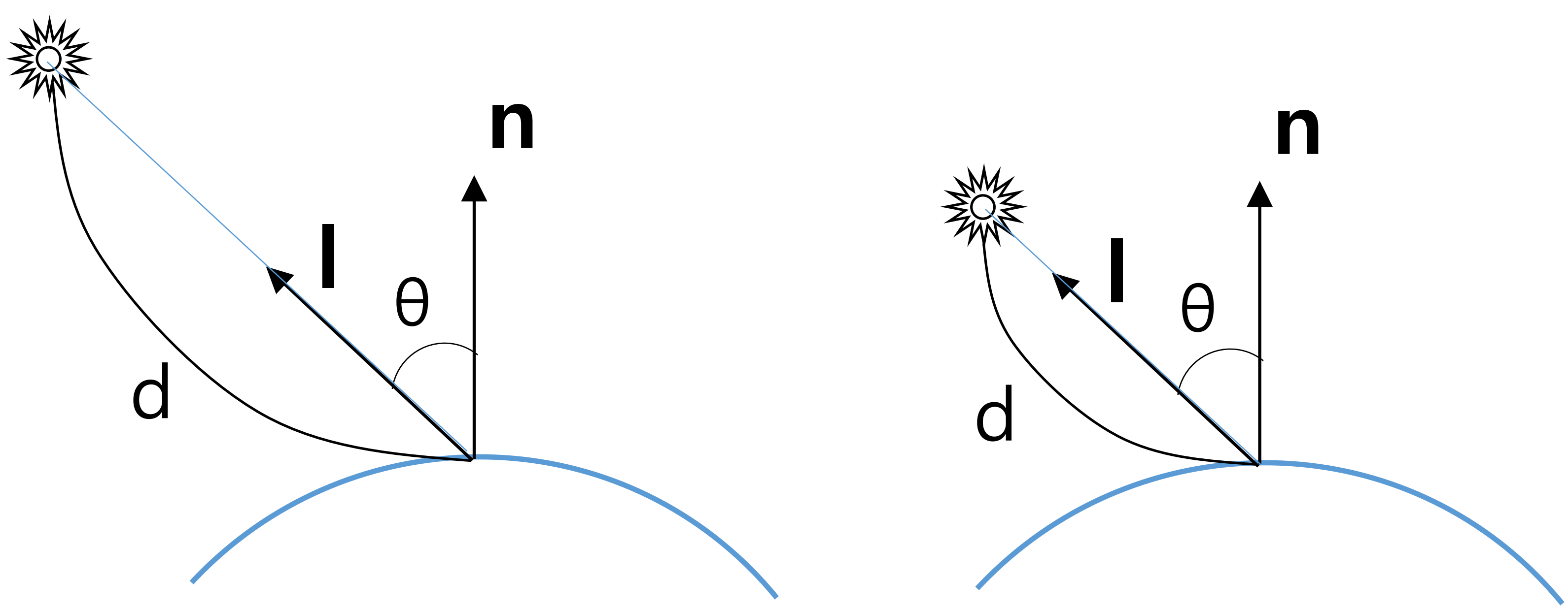
At noon

In the afternoon



Modeling Diffuse Reflection (2)

- Attenuation Terms
  - The light from a point source that reaches a surface is inversely proportional to the distance between them
  - We can add a factor of the form $1/(a + bd + cd^2)$ to the diffuse and specular terms
  - The constant and linear terms soften the effect of the point source



Modeling Diffuse Reflection (3)

- no consideration of distance term
  - $I_d = k_d L_d (l \cdot n)$
- considering distance term
  - d: distance from the light source to the point
  - $I_d = k_d L_d (l \cdot n) / (a + bd + cd^2)$

Modeling Diffuse Reflection (4)

- Normal transformation

```
mat4 mNormal = transpose(inverse( uModel ));
vec4 vNormal = mNormal * aNormal;
```

Program Example (1)

```
static char* vsSource = "#version 140 ₩n₩
in vec4 aPosition; ₩n₩
in vec4 aNormal; ₩n₩
out vec4 vColor; ₩n₩
uniform mat4 uscale; ₩n₩
uniform mat4 utranslate; ₩n₩
uniform vec4 light_position;₩n₩
uniform vec4 light_ambient; ₩n₩
uniform vec4 light_diffuse; ₩n₩
uniform vec4 light_att; ₩n₩
uniform vec4 material_ambient; ₩n₩
uniform vec4 material_diffuse; ₩n₩
void main(void) { ₩n₩
        vec4 vPosition = uscale * utranslate * aPosition; ₩n₩
        vec4 ambient = light_ambient * material_ambient; ₩n₩
        mat4 mNormal = transpose(inverse(uscale*utranslate)); ₩n₩
        vec4 vNormal = mNormal * aNormal; ₩n₩
        vec3 N = normalize(vNormal.xyz); ₩n₩
        vec3 L = normalize(light_position.xyz - vPosition.xyz); ₩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₩
        vColor = ambient + diffuse; ₩n₩
        gl_Position = vPosition; ₩n₩
}";
```

Program Example (2)

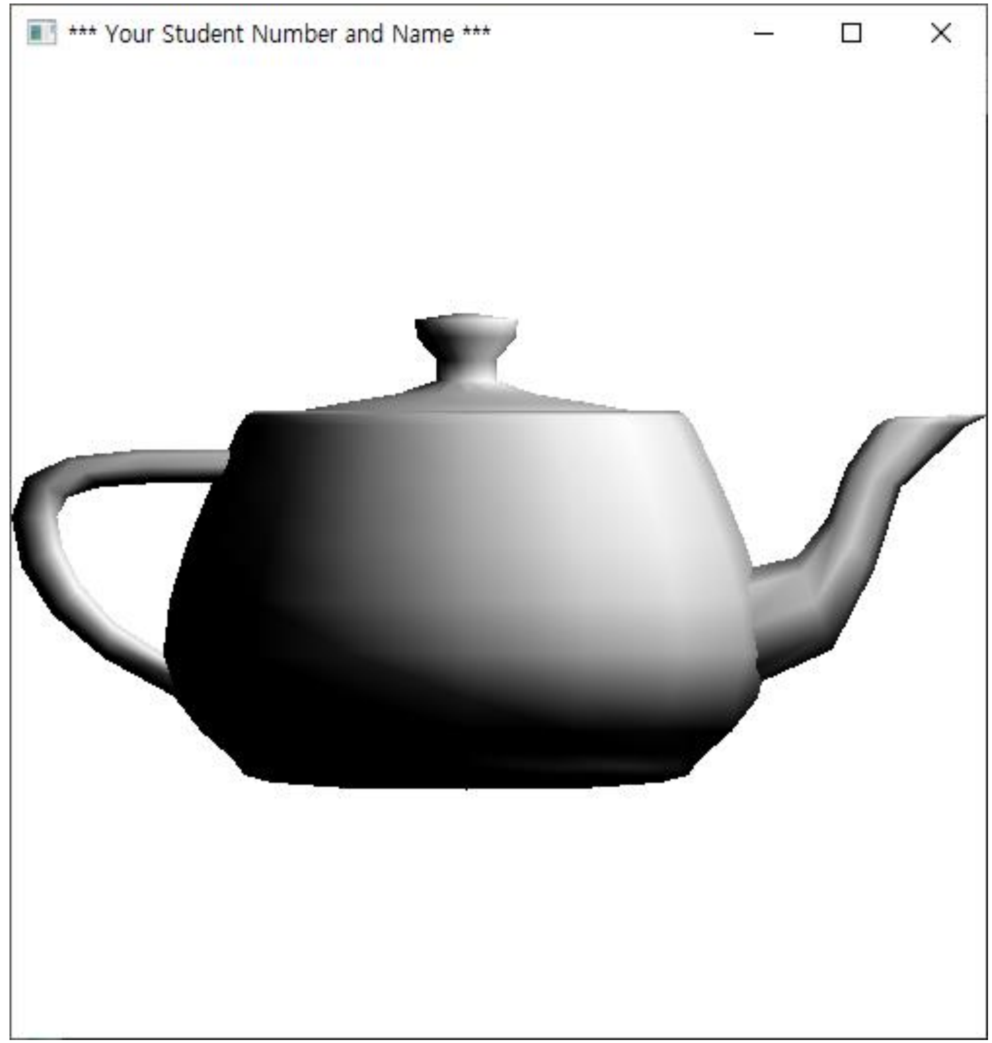
```
void setLightAndMaterial(void) {
        GLfloat light_pos[4] = { 2.0, 2.0, -2.0, 1.0 };
        GLfloat light_amb[4] = { 0.3, 0.3, 0.3, 1.0 };
        GLfloat light_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat light_att[4] = { 1.0, 0.0, 0.0, 1.0 };
        GLfloat mat_amb[4] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat mat_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
        GLuint loc;

        loc = glGetUniformLocation(prog, "light_position");
        glUniform4fv(loc, 1, light_pos);
        loc = glGetUniformLocation(prog, "light_ambient");
        glUniform4fv(loc, 1, light_amb);
        loc = glGetUniformLocation(prog, "light_diffuse");
        glUniform4fv(loc, 1, light_dif);
        loc = glGetUniformLocation(prog, "light_att");
        glUniform4fv(loc, 1, light_att);

        loc = glGetUniformLocation(prog, "material_ambient");
        glUniform4fv(loc, 1, mat_amb);
        loc = glGetUniformLocation(prog, "material_diffuse");
        glUniform4fv(loc, 1, mat_dif);
}
```

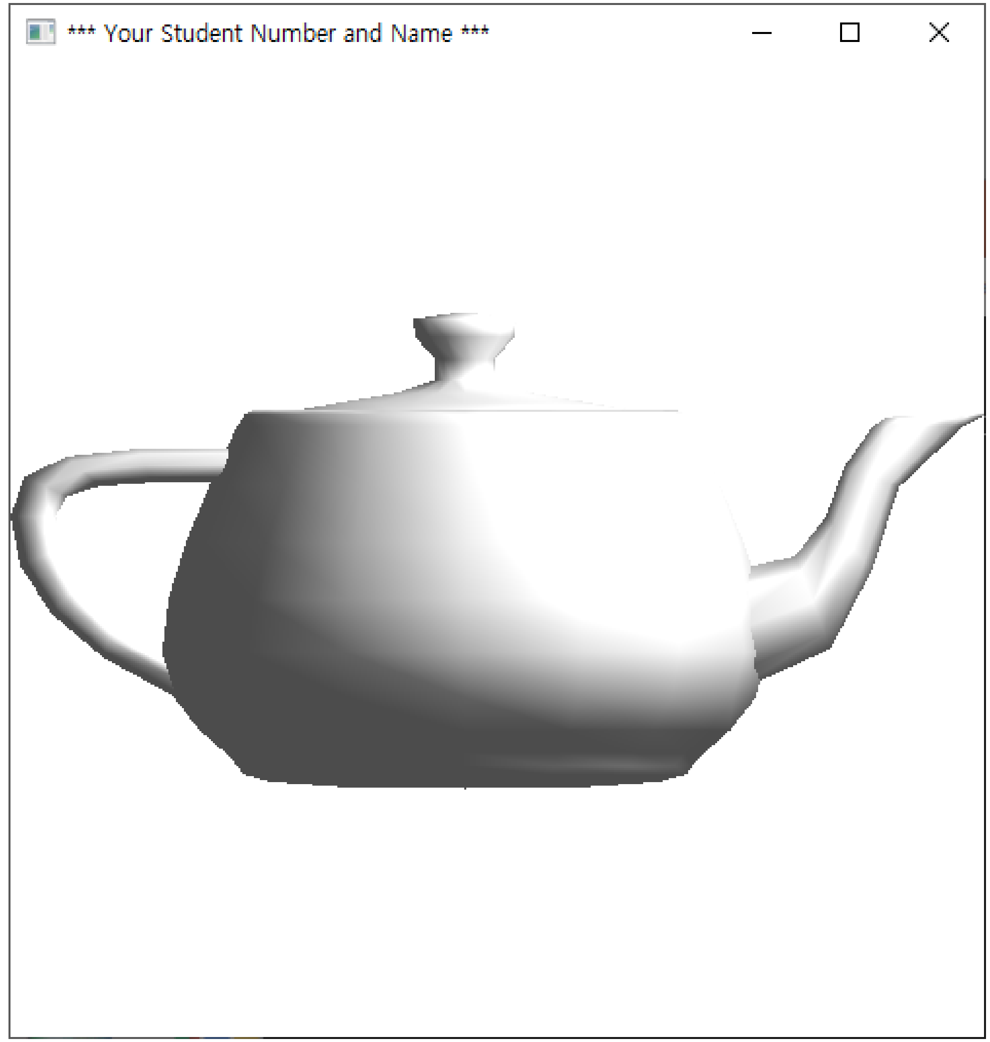
Diffuse Example

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



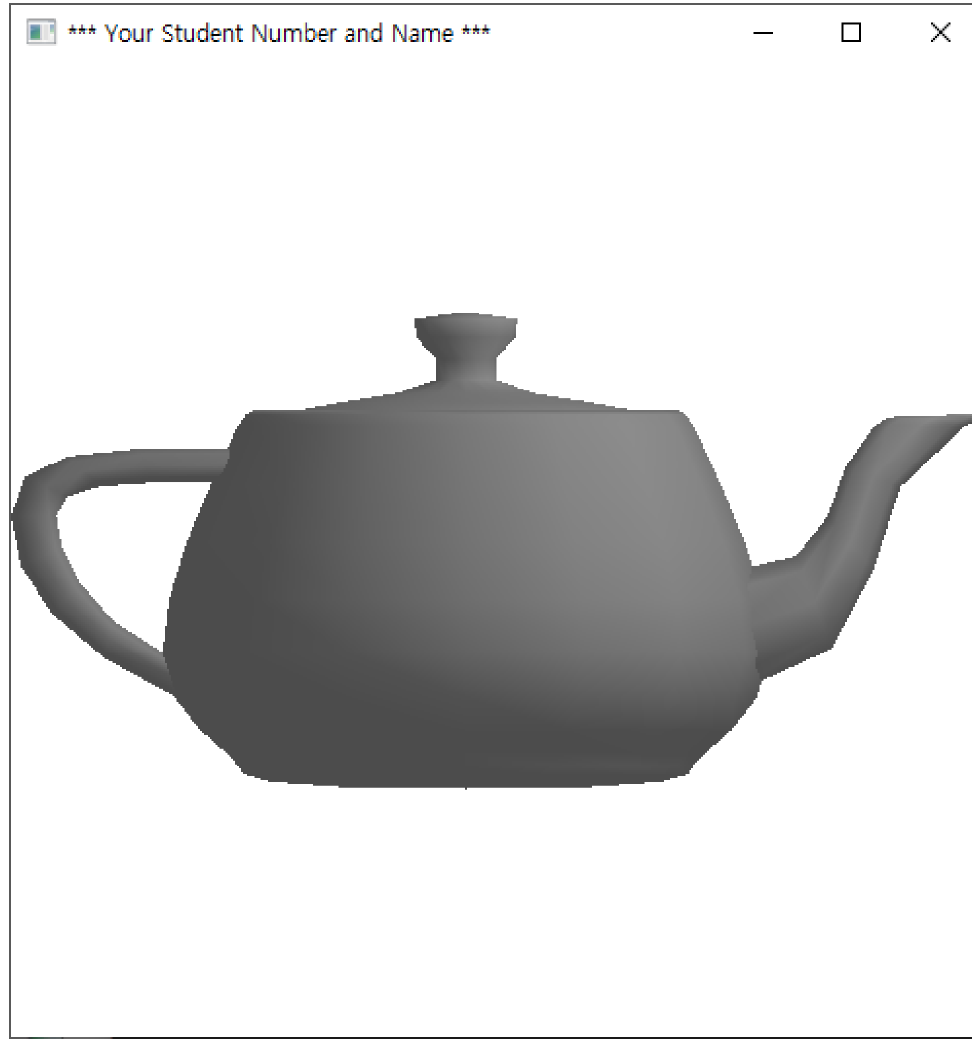
Diffuse + Ambient Exmaple (1)

GLfloat light_pos[4] =
{ 2.0, 2.0, -2.0, 1.0 };
GLfloat light_amb[4] =
{ 0.3, 0.3, 0.3, 1.0 };
GLfloat light_dif[4] =
{ 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] =
{ 1.0, 0.0, 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 };



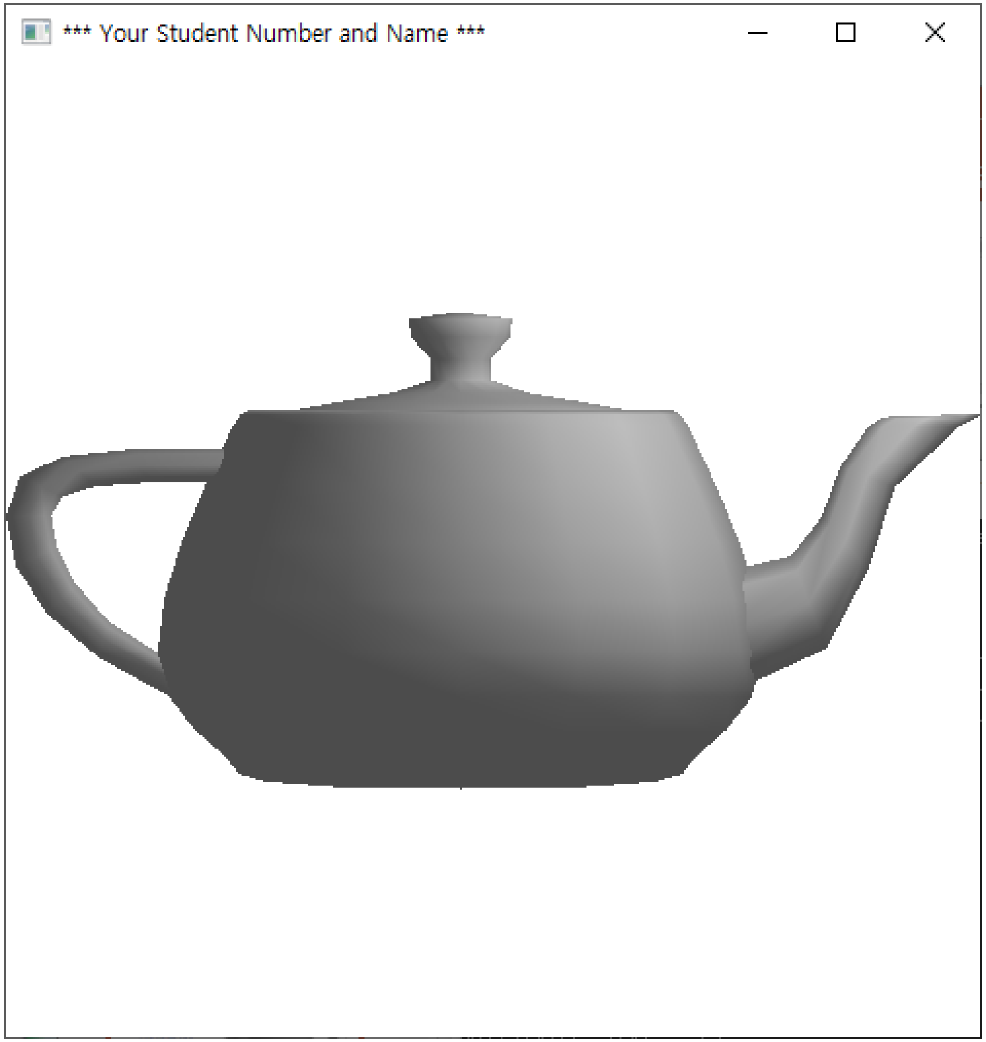
Diffuse + Ambient Exmaple (2)

GLfloat light_pos[4] =
{ 2.0, 2.0, -2.0, 1.0 };
GLfloat light_amb[4] =
{ 0.3, 0.3, 0.3, 1.0 };
GLfloat light_dif[4] =
{ 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] =
{ 1.0, 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 };



Diffuse + Ambient Exmaple (3)

```
GLfloat light_pos[4] =
{ 1.0, 1.0, -1.0, 1.0 };
GLfloat light_amb[4] =
{ 0.3, 0.3, 0.3, 1.0 };
GLfloat light_dif[4] =
{ 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] =
{ 1.0, 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 };
```



HW#22 Implement ambient & diffuse reflection (1) (20points)

- Due date: Next Friday 6:00pm
- Implement ambient & diffuse reflection by using (& modifying if necessary) the codes presented in this class.
- You can use a model either
  - by reading the file 'teapot.obj'
  - or by fill vertices, normals, indices arrays by given '3d_object.obj', if you have difficulties for file read
- The entire model must be shown in the window.
- Adjust the coefficients for shading and light source position to clearly show the shading effect.
- If the shading result doesn't look natural, or wrong computation is found, there will be score deduction.
- Submit .c file

HW#22 Implement ambient & diffuse reflection (2)

- Possible running examples