LEC24: Implementing Phong Reflection Model-part2

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

- Modeling specular term for Phong reflection model
- Program code
- Modified Phong model

Modeling Specular Reflection

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased
- α : shininess coefficient

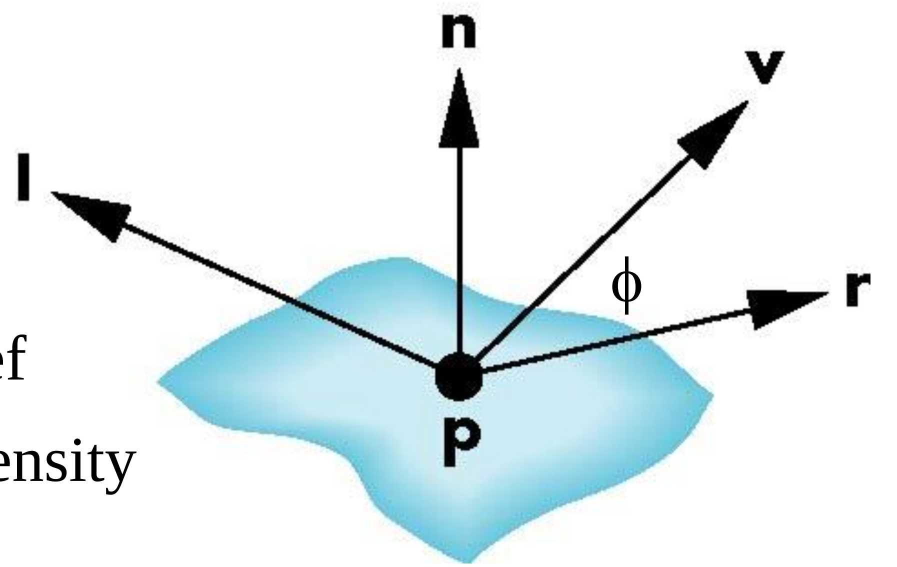
$$\mathbf{I} \sim k_s \, L_s \cos^{\alpha}\phi$$

reflected intensity

material reflection coefficient

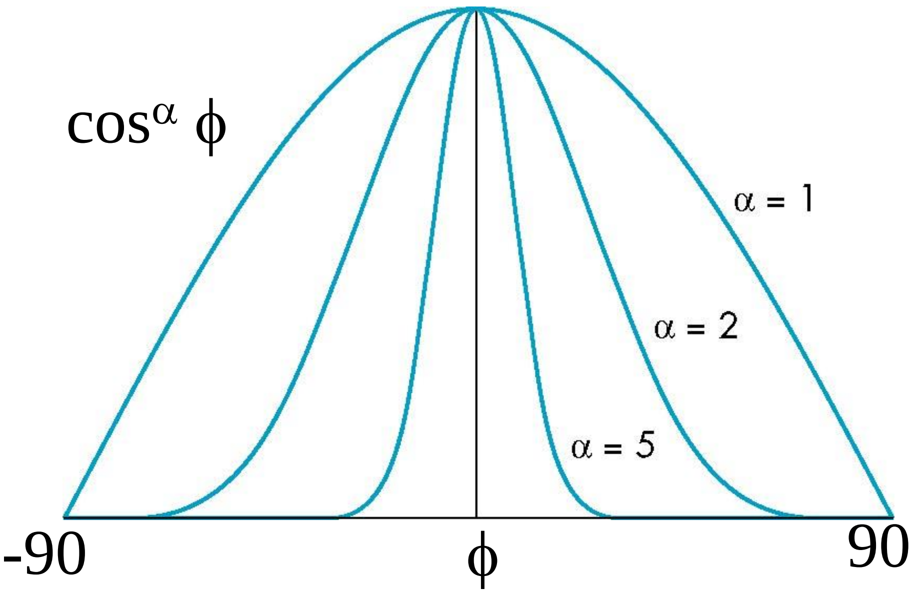
incoming light intensity

shininess coef



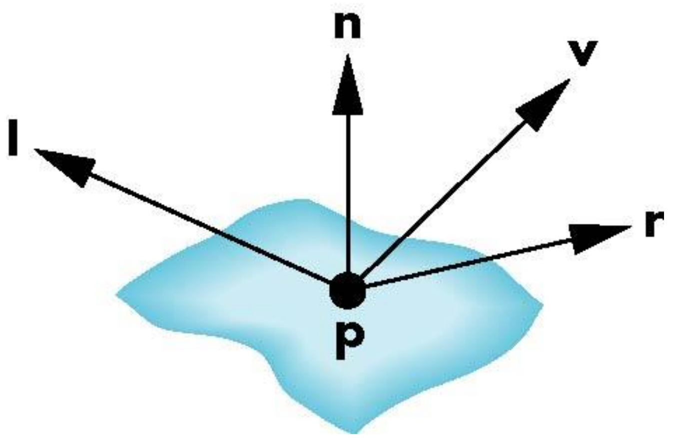
The Shininess Coefficient

- Values of α between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic



Adding up the Components

- For each light source and each color component, the Phong model can be written as
  - $I = k_a L_a + k_d L_d\ \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^{\alpha}$
  - $I = k_a L_a + (k_d L_d\ \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^{\alpha})/(a + bd + cd^2)$
- For each color component, we add contributions from all sources



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 mat4 urotate; ₩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 Example (2)

```
void main(void) { ₩n₩
vec4 vPosition = urotate * uscale * utranslate * aPosition; ₩n₩
mat4 mNormal = transpose(inverse(urotate * uscale * utranslate)); // normal transformation ₩n₩
vec4 vNormal = mNormal * aNormal; // normal vector in view frame ₩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 R = normalize(2 * dot(L, N) * N - L); ₩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(R, V), 0.0), material_shineness) * light_specular * material_specular / denom; ₩n₩
vColor = ambient + diffuse + specular; ₩n₩
gl_Position = vPosition; ₩n₩
}";
```

Program Example (3)

```
void setLightAndMaterial(void) {
        GLfloat light_pos[4] = { 1.0, 0.5, -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 = 100;
        GLuint loc;
        // light
        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);
```

Program Example (4)

```
    loc = glGetUniformLocation(prog, "light_specular");
    glUniform4fv(loc, 1, light_spe);
    loc = glGetUniformLocation(prog, "light_att");
    glUniform4fv(loc, 1, light_att);
    // material
    loc = glGetUniformLocation(prog, "material_ambient");
    glUniform4fv(loc, 1, mat_amb);
    loc = glGetUniformLocation(prog, "material_diffuse");
    glUniform4fv(loc, 1, mat_dif);
    loc = glGetUniformLocation(prog, "material_specular");
    glUniform4fv(loc, 1, mat_spe);
    loc = glGetUniformLocation(prog, "material_shininess");
    glUniform1f(loc, mat_shi);
}
```

Ambient + Diffuse + Specular Example (1)

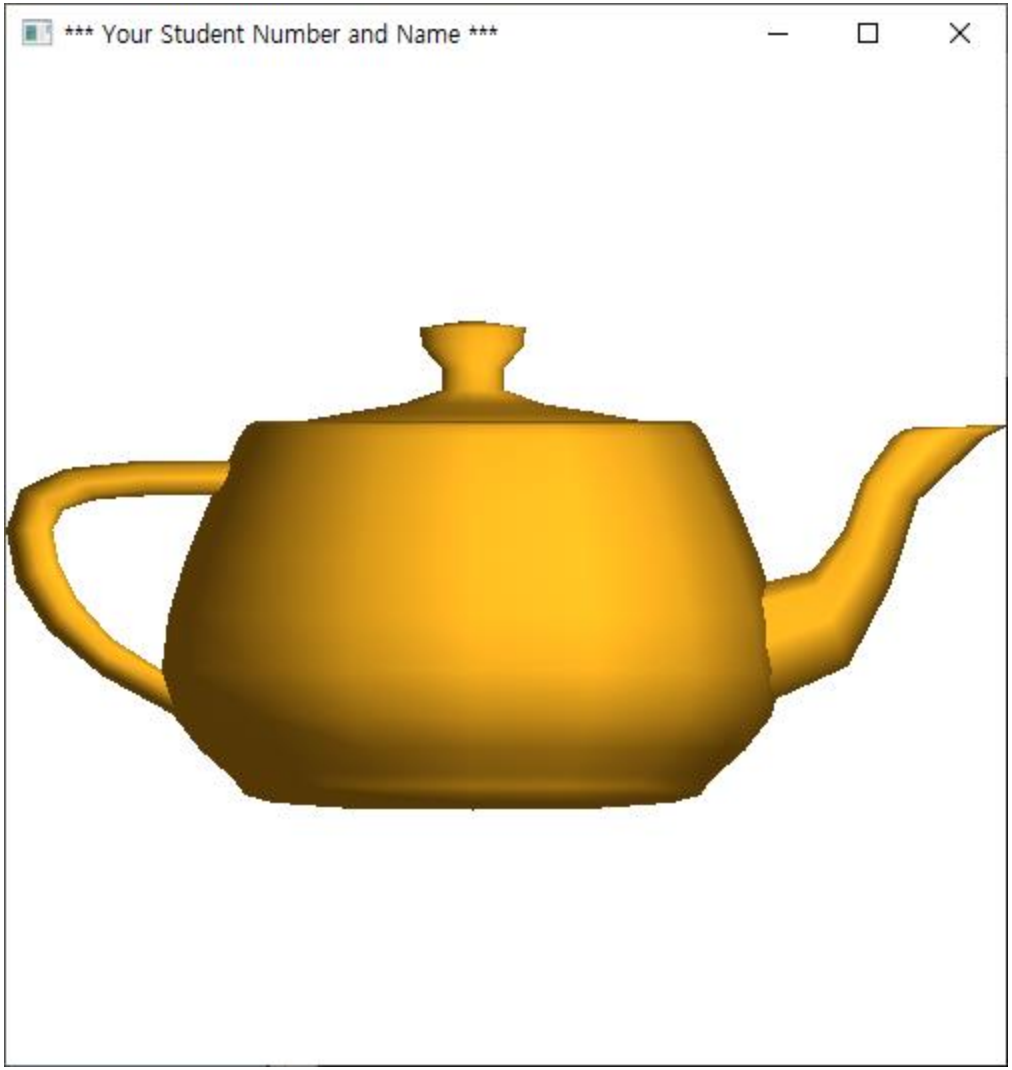
```
GLfloat light_pos[4] = { 0.5, 1.0, -2.0, 1.0 };
GLfloat light_amb[4] = { 0.5, 0.5, 0.5, 1.0 };
GLfloat light_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_spe[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] = { 0.0, 1.0, 0.0, 1.0 };
GLfloat mat_amb[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_dif[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_spe[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shi = 1000;
```

Ambient + Diffuse + Specular Example (2)

```
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 = 1000;
```

Ambient + Diffuse + Specular Example (3)

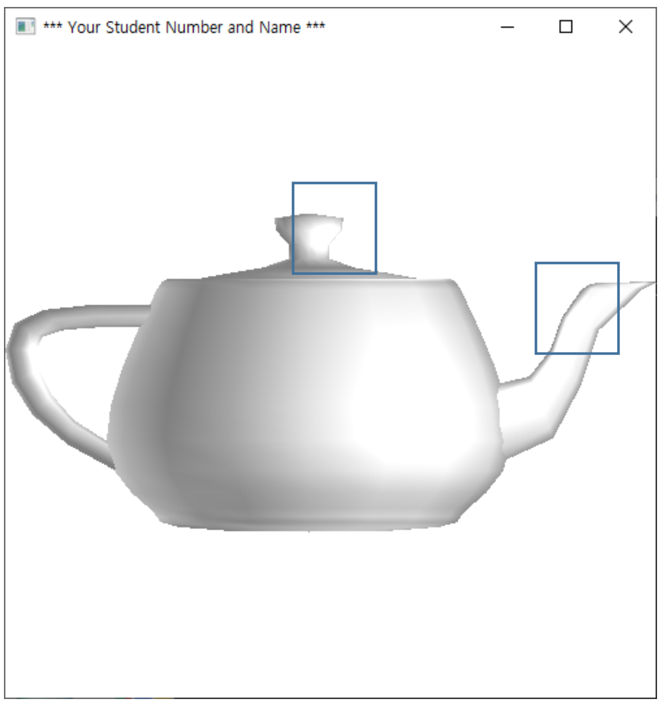
```
GLfloat light_pos[4] =
{ 0.5, 1.0, -2.0, 1.0 };
GLfloat light_amb[4] =
{ 0.5, 0.5, 0.5, 1.0 };
GLfloat light_dif[4] =
{ 1.0, 1.0, 1.0, 1.0 };
GLfloat light_spe[4] =
{ 1.0, 1.0, 1.0, 1.0 };
GLfloat light_att[4] =
{ 0.0, 1.0, 0.0, 1.0 };
GLfloat mat_amb[4] =
{0.329412f, 0.223529f,
0.027451f, 1.0f   };
GLfloat mat_dif[4] =
{0.780392f, 0.568627f,
0.113725f, 1.0f   };
GLfloat mat_spe[4] =
{0.992157f, 0.941176f,
0.807843f, 1.0f   };
GLfloat mat_shi =
27.8974f;
```

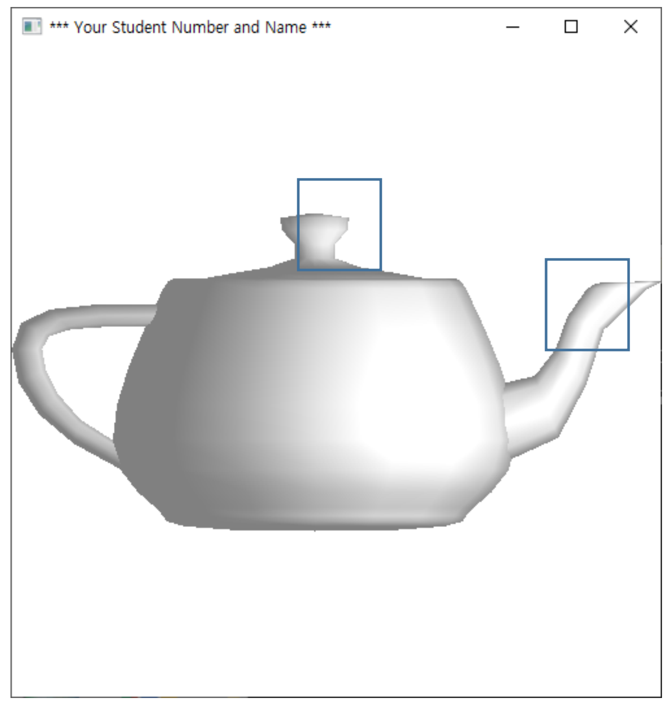


Shininess effect

```
GLfloat light_pos[4] = { 1.2, 0.0, -1.8, 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; // shininess
```

```
GLfloat mat_shi = 100; // shininess
```





Program Example with Applying View Matrix (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 mat4 urotate; ₩n₩
uniform mat4 uView; ₩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 Example with Applying View Matrix (2)

```
void main(void) { ₩n₩
vec4 vPosition = uView * urotate * uscale * utranslate * aPosition; ₩n₩
mat4 mNormal = transpose(inverse(uView* urotate * uscale * utranslate)); ₩n₩
vec4 vNormal = mNormal * aNormal; // normal vector in view frame ₩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 R = normalize(2 * dot(L, N) * N - L); ₩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(R, V), 0.0), material_shineness) * light_specular * material_specular / denom; ₩n₩
vColor = ambient + diffuse + specular; ₩n₩
gl_Position = vPosition; ₩n₩
}";
```

Material Effect

- http://www.it.hiof.no/~borres/j3d/explain/light/p-materials.html

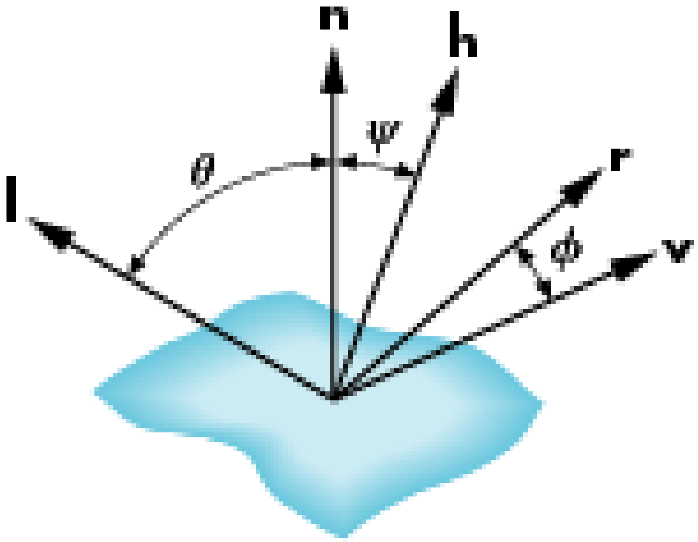
Modified Phong Model

- The specular term in the Phong model is problematic because it requires the calculation of a new reflection vector and view vector for each vertex
- Blinn suggested an approximation using the halfway vector that is more efficient

The Halfway Vector

- **h** is normalized vector halfway between **l** and **v**

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / \|\mathbf{l} + \mathbf{v}\|$$

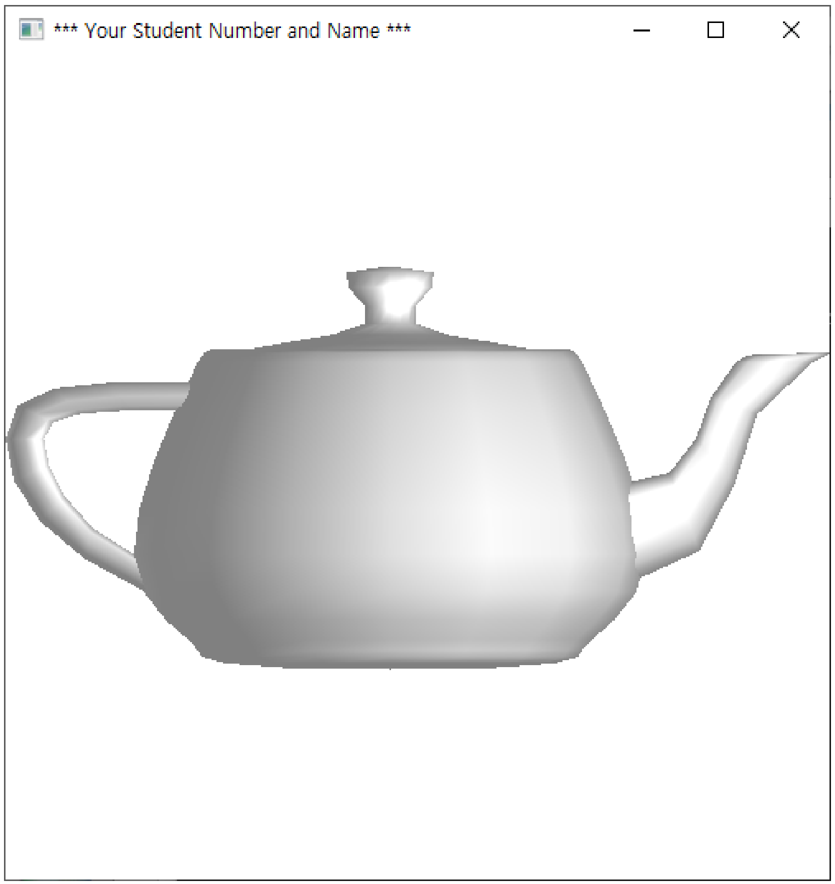


Using the halfway vector

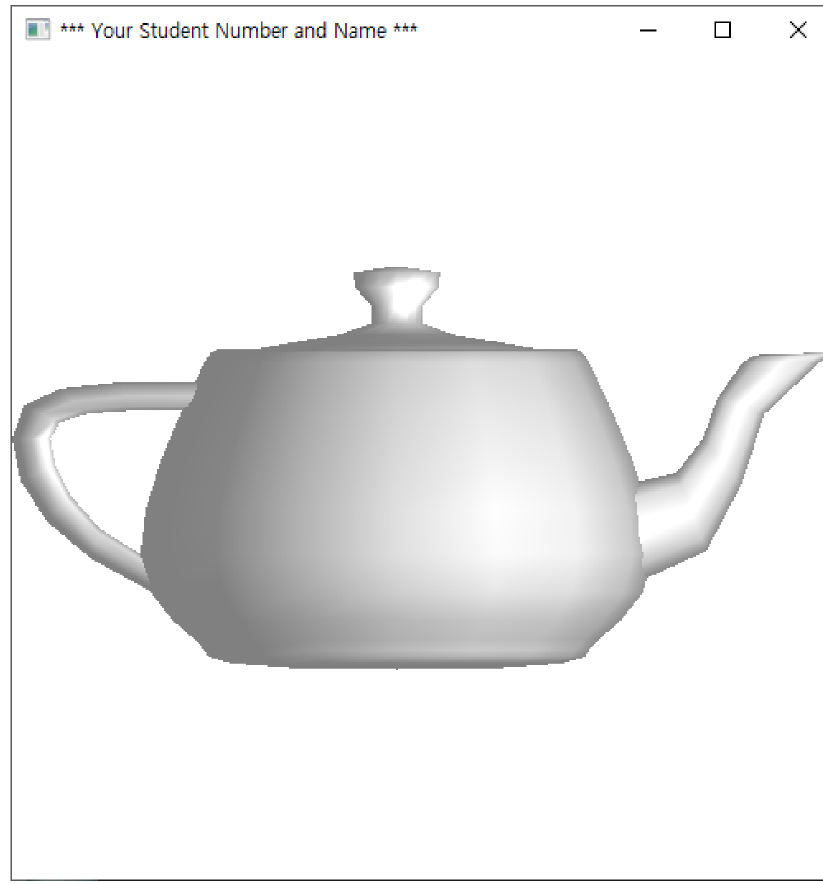
- Replace $(\mathbf{v} \cdot \mathbf{r})^{\alpha}$ by $(\mathbf{n} \cdot \mathbf{h})^{\beta}$
- β is chosen to match shininess
- Resulting model is known as the modified Phong model (or Blinn lighting model)

Comparison of Phong & Modified Phong

Phong model
pow(max(dot(R, V), 0.0), 1)

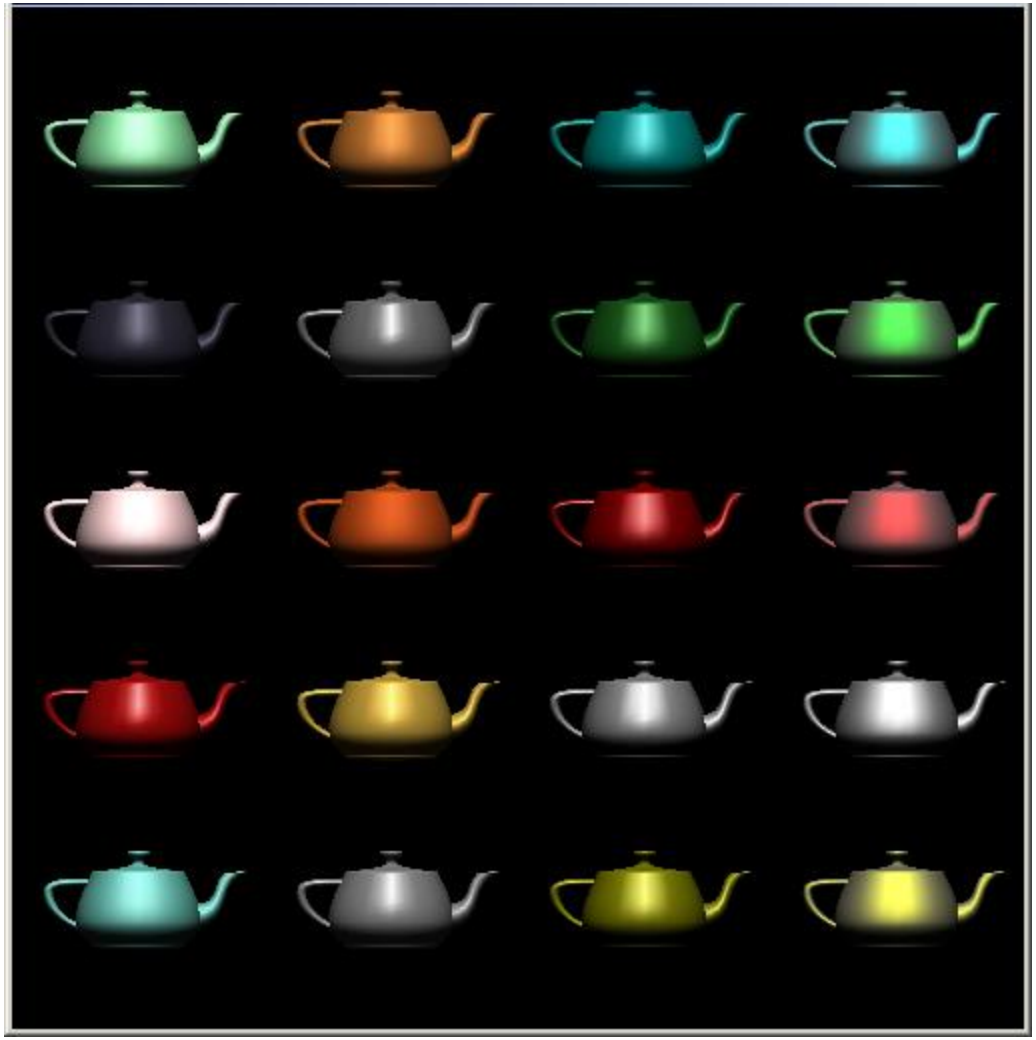


modified Phong model
pow(max(dot(N, H), 0.0), 7)



Shading Example

- Only differences in these teapots are the parameters in the modified Phong model



HW#23 Implement modified Phong model.

- No submission is required.
- Try to implement modified Phong model.