GLSL Lighting

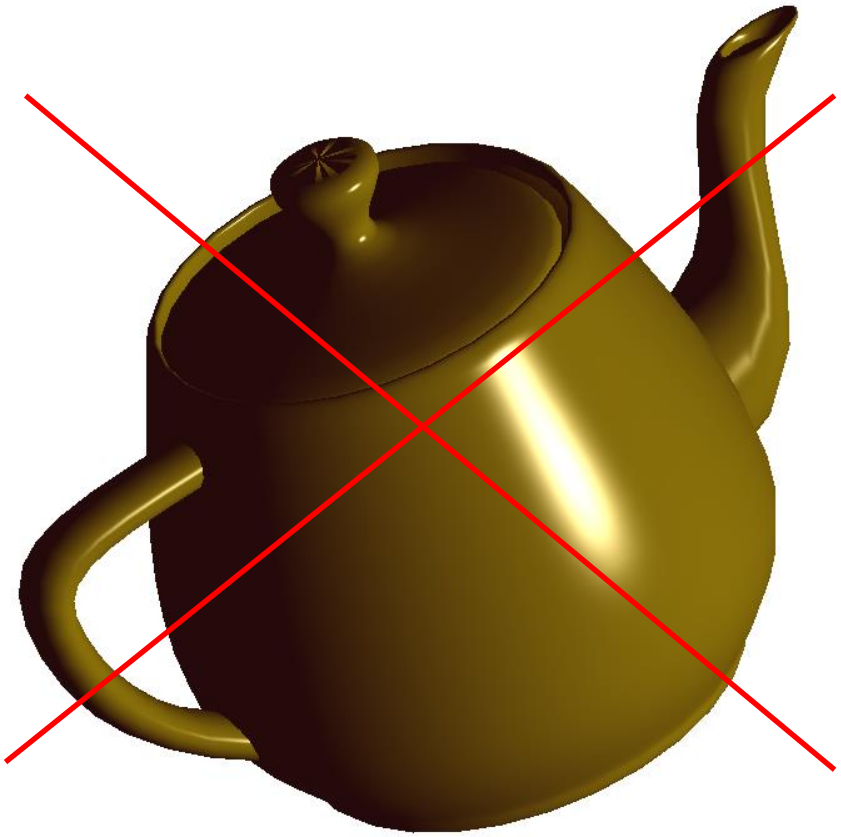
Shading Basic

What is Shading?

- In computer graphics, shading refers to the process of altering the color of an object/surface/polygon in the 3D scene, based on its angle to lights and its distance from lights to create a photorealistic effect. Shading is performed during the rendering process by a program called a shader.

Shading with OpenGL

- OpenGL is focused on real-time applications.
- OpenGL provides very basic shading models.

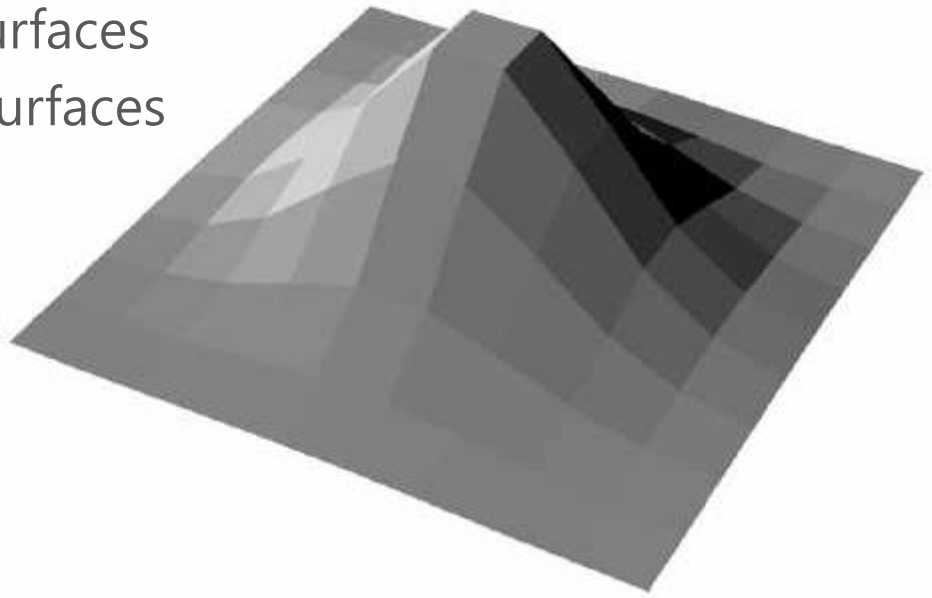


GL_FLAT

GL_SMOOTH

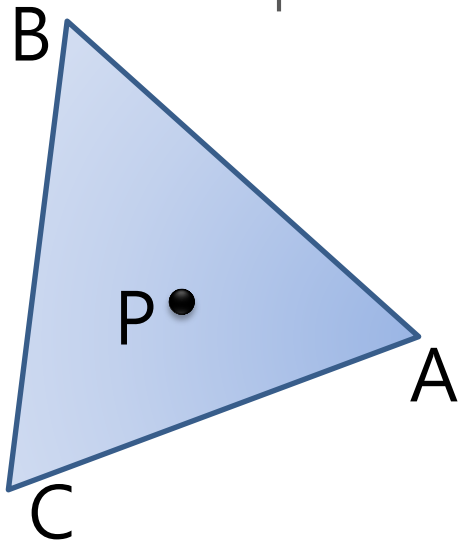
Flat Shading

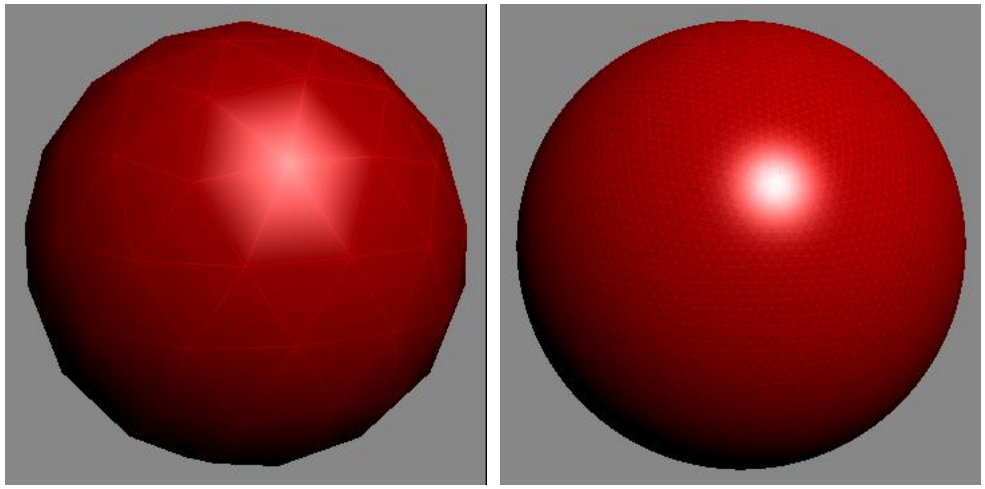
- **Enable with glShadeModel(GL_FLAT);**
- **Selects the computed color of just one vertex and assigns it to all the pixel fragments of the polygon.**
- **Assessment**
  - Inexpensive to compute
  - Less pleasant for smooth surfaces
  - Not pleasant even for flat surfaces



Gouraud Shading

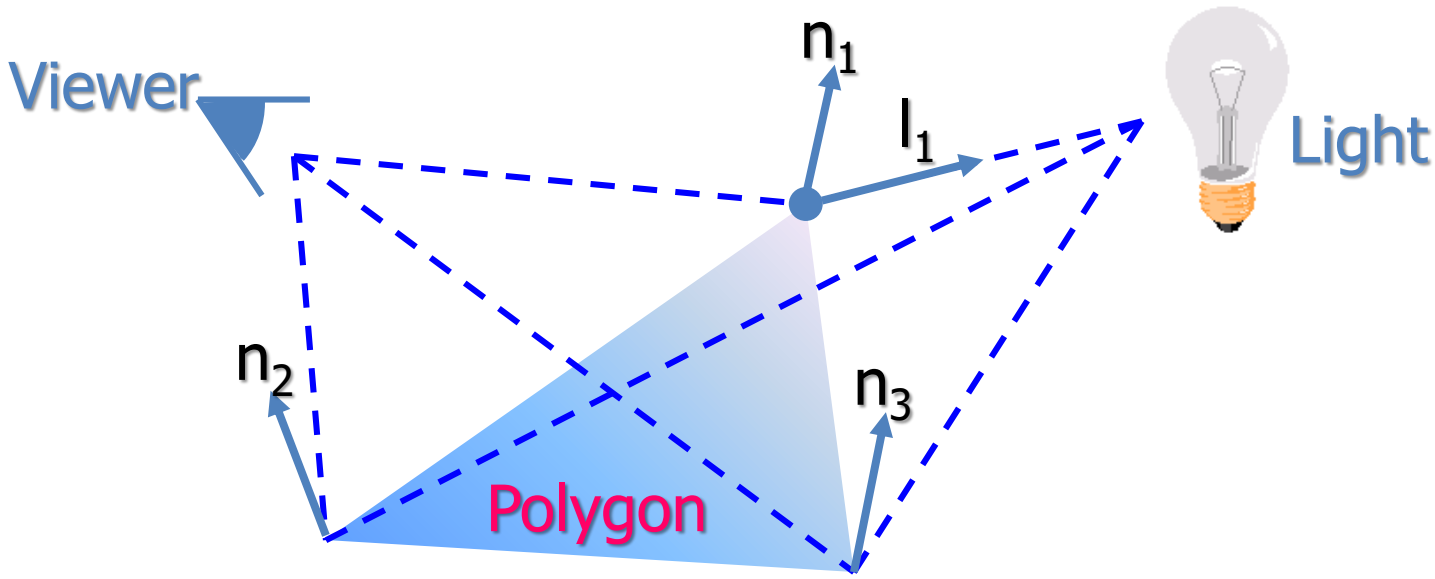
- **Enable with glShadeModel(GL_SMOOTH);**
- **Calculate color at each vertex**
- **Interpolate the vertex color for the interior**
- **Assessment**
  - Better image
  - More expensive to calculate





Gouraud Shading Process (1/2)

- **One radiance calculation per vertex**
  - Interpolates pixels inside polygon
    - By using the colors computed at vertices

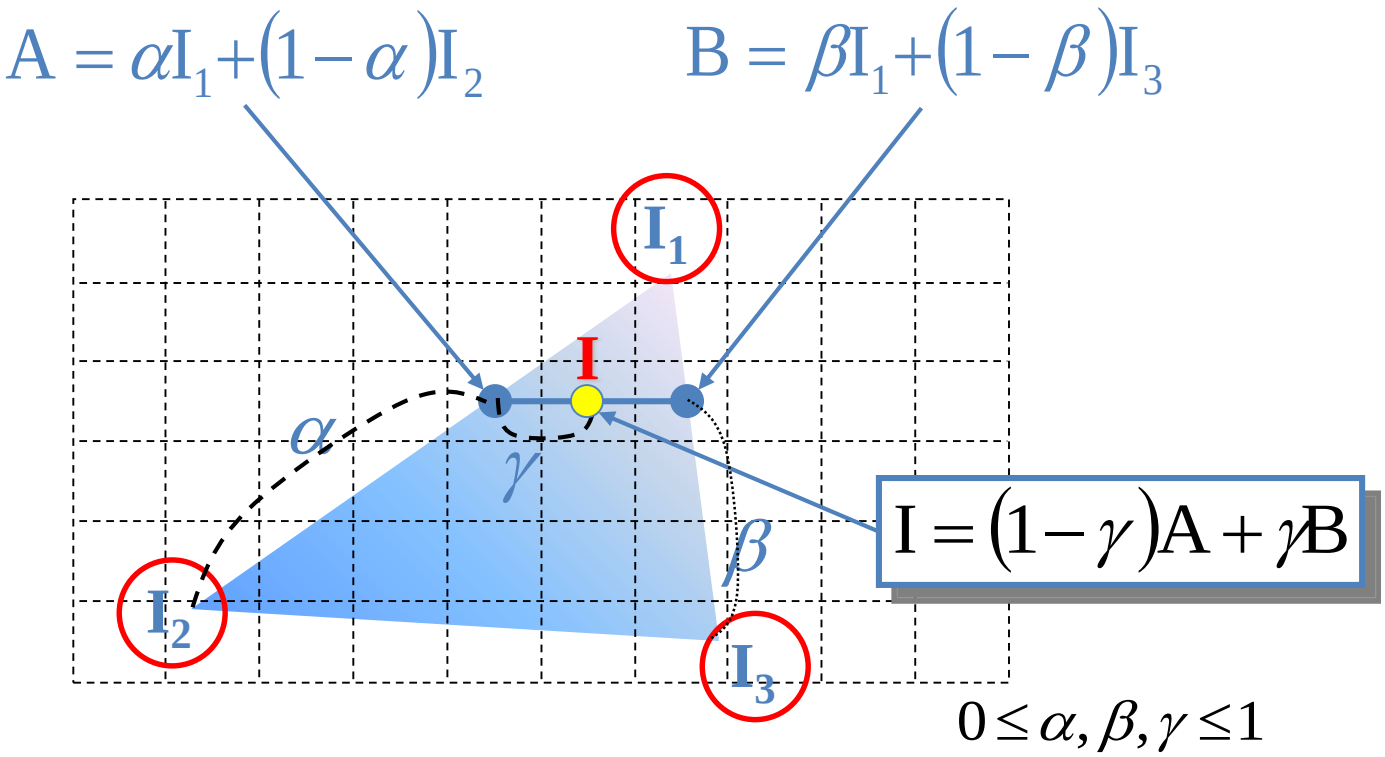


Vertex Normal Vector

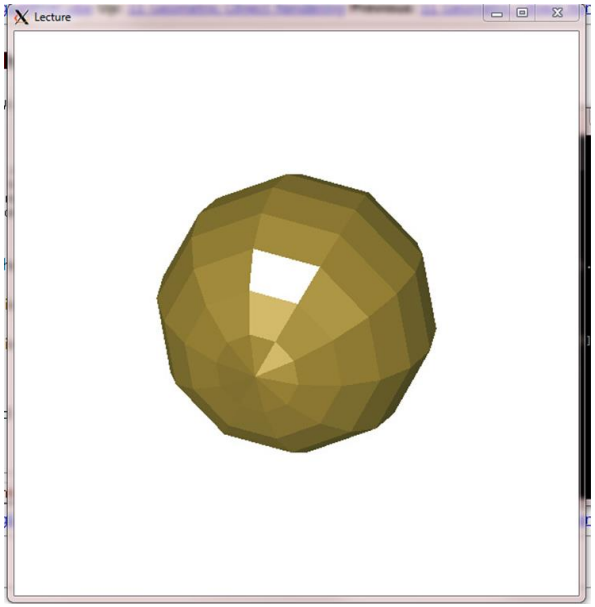
$$I = K_e + K_a L_{ga} + \sum_{lights} (Spot_i)(Att_i)(K_a L_{i,a} + K_d L_{i,d}(\mathbf{n}\cdot\mathbf{l}) + K_s L_{i,s}(\mathbf{n}\cdot\mathbf{h}_i)^{\alpha_i})$$

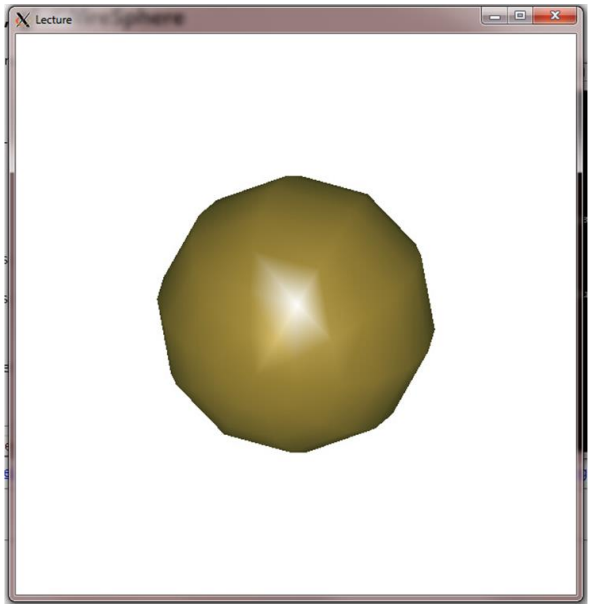
Gouraud Shading Process (2/2)

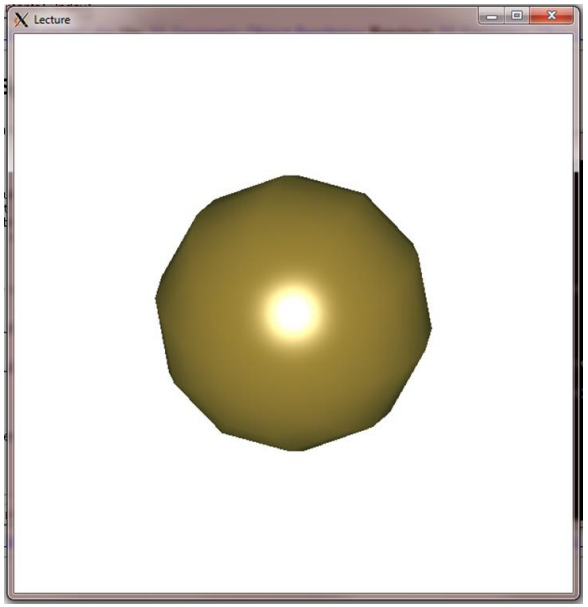
- **Bilinearly interpolate colors at vertices**
  - Down and across scan lines



Shading Model Comparison



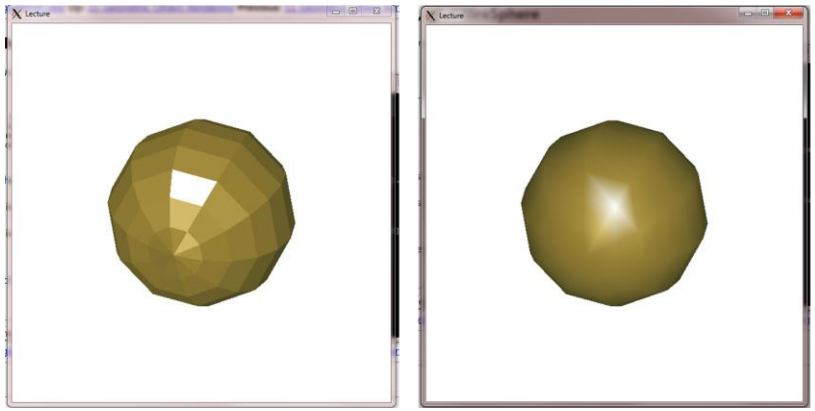




- Make Flat, Smooth, Phong shaded 3D Sphere

Draw Sphere with glShadeModel

```
void display(){
        /*initialize*/
        /*Set Shading Model*/
        glShadeModel(GL_FLAT or GL_SMOOTH);
        glutSolidSphere(0.5f,10,10);
        glXSwapBuffers(dpy, win);
}
```

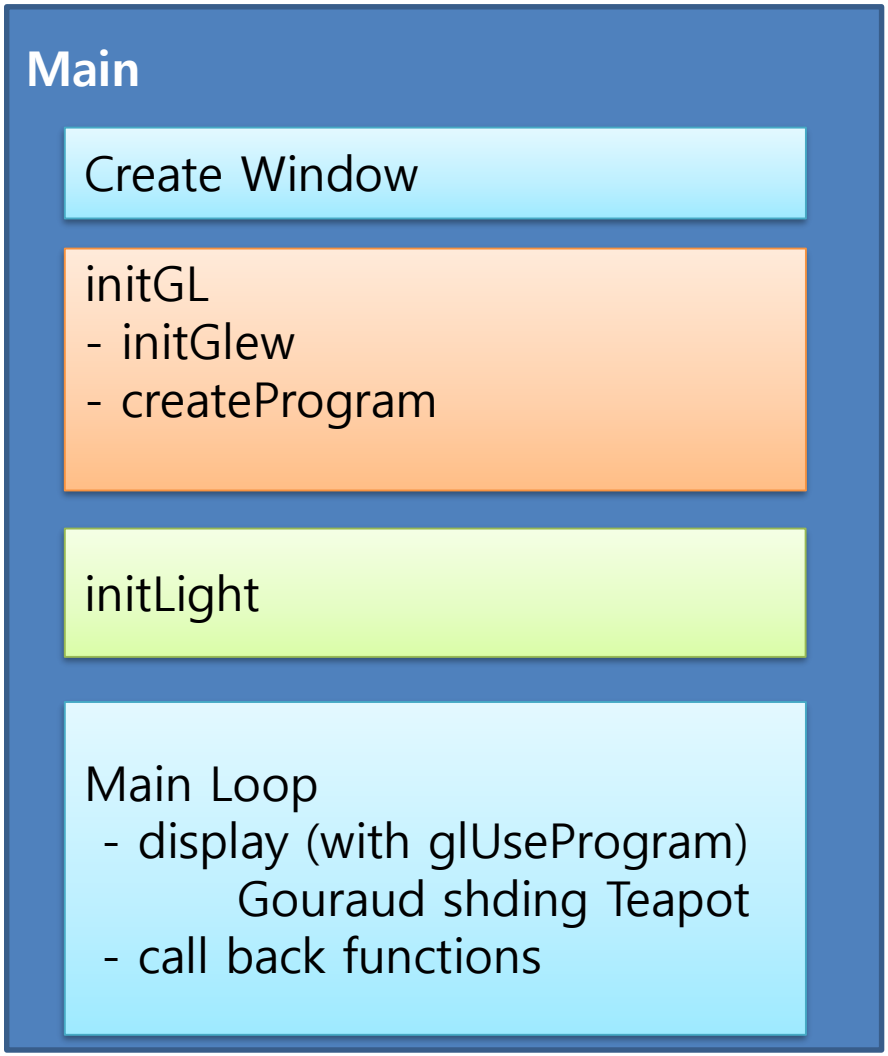


GL_FLAT GL_SMOOTH

Gouraud Shading with GLSL

- **You can draw Gouraud Shading with OpenGL Basic Rendering.**
  - With “GL_SMOOTH”
- **But if you want to draw Gouraud Shading with GLSL, you need to write Lighting per vertex.**
  - With “GL_SMOOTH” too.

Program Flow



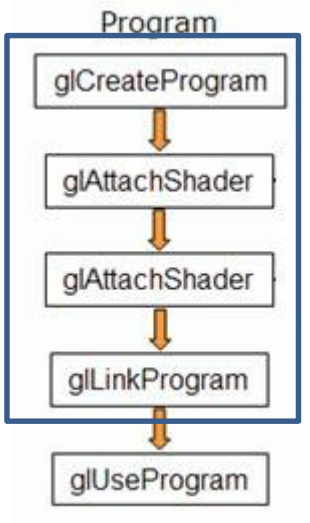
initGL()

```
void initGL()
{
        glewInit(); //glew Initialize Function;
        createProgram(); //Create Shader Program
}
```

Run-time compilation w/ shader source file

createProgram@initGL()

```
void createProgram(){
        char* vert;
        char* frag;
        vert = readShader("Vertex.glsl");
        frag = readShader("Fragment.glsl");
        vertShader = createShader(vert, GL_FRAGMENT_SHADER);
        fragShader = createShader(frag, GL_FRAGMENT_SHADER);
        GLuint p = glCreateProgram();
        glAttachShader(p, vertShader);
        glAttachShader(p, fragShader);
        glLinkProgram(p);
        program = p;
}
```



initLight(): Light Initialize

```
void initLight(){
        /*Set Light and Material Properties with Array*/
        /*Set light properties*/
        glLightfv(GL_LIGHT0, GL_AMBIENT, lightKa);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightKd);
        glLightfv(GL_LIGHT0, GL_SPECULAR, lightKs);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
        /*Set material properties*/
        glMaterialfv(GL_FRONT, GL_AMBIENT, matKa);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matKd);
        glMaterialfv(GL_FRONT, GL_SPECULAR, matKs);
        glMaterialfv(GL_FRONT, GL_SHININESS, &matShininess);
         /*Enable Light*/
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
}
```

Draw Gouraud Shaded Teapot

```
void display() {
    ...
    glUseProgram(program);  // activate shader program
    glShadeMode(GL_SMOOTH);
    glutSolidTeapot(0.5f);
    glUseProgram(0);        // deactivate shader program
    ...
}
```

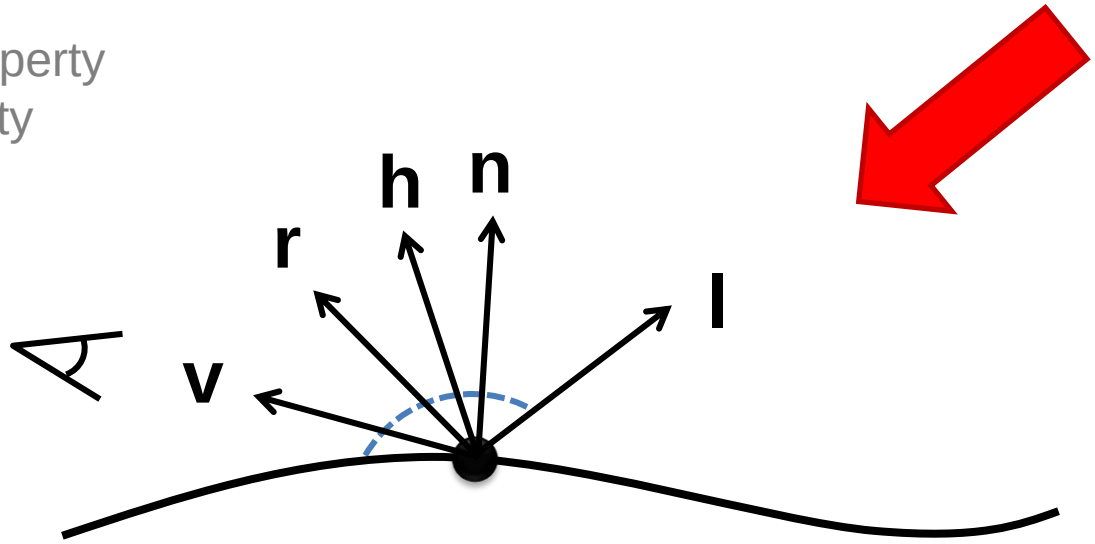
Simplified Lighting Model

- Let's assume **directional light**.

$$I = I_{ambient} + I_{diffuse} + I_{specular}$$

$$= K_a L_{g,a} + K_a L_{0,a} + K_d L_{0,d}(\mathbf{n}\cdot\mathbf{l}) + K_s L_{0,s}(\mathbf{n}\cdot\mathbf{h}_0)^{\alpha_0}$$

K: Material property
L: Light property

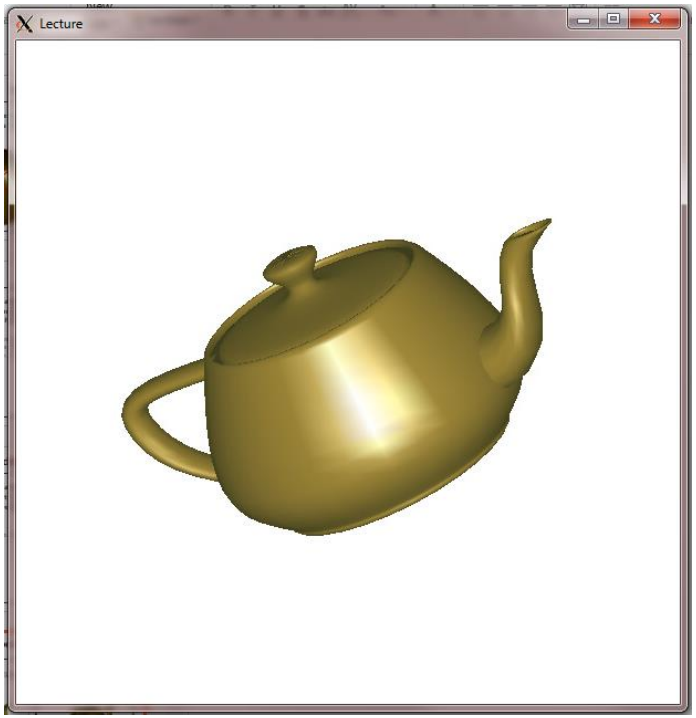


Radiance Calculation: Vertex Shader

```
#version 130
void main() {
        vec3 L1 = normalize(gl_LightSource[0].position.xyz);
        vec3 n = normalize(gl_NormalMatrix*gl_Normal);
        vec3 h = normalize(gl_LightSource[0].halfVector.xyz);
        float NdotL, NdotH;
        vec4 color = gl_FrontMaterial.ambient * gl_LightSource[0].ambient +
            gl_FrontMaterial.ambient * gl_LightModel.ambient;
        NdotL = max(dot(n,L1),0.0);
        if (NdotL > 0.0) {
                color += gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse * NdotL;
                NdotH = max(dot(n,h),0.0);
                color = color;
                color += gl_FrontMaterial.specular * gl_LightSource[0].specular *
                pow(NdotH, gl_FrontMaterial.shininess);
        }
        gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
        gl_FrontColor = color;
}
```

Gouraud Shading: Fragment&Result

```
#version 130
void main() {
        gl_FragColor = gl_Color;
}
```



Gouraud Shading Coding Exercise

- **Copy Sample Skeleton Code**
  - vglconnect ID@163.152.20.246
  - cp –r /home/share/Gouraud ./
  - cd Gouraud
- **Notepad: shader code 작성**
  - Fragment.glsl
  - Vertex.glsl
- **Compile program**
  - make
  - vglrun ./Gouraud

Let's Coding Stripped Teapot

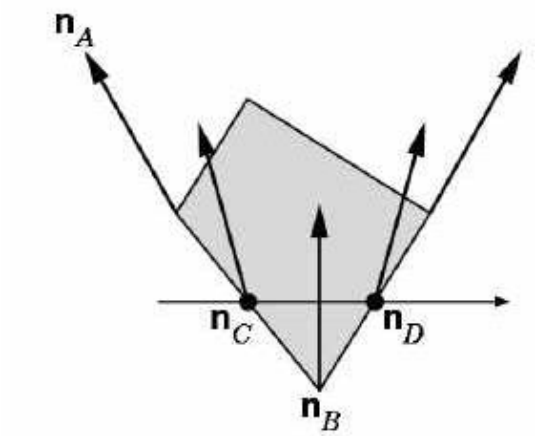
Stripped Teapot Code

```
// in vertex shader
varying float x;
Void main(){
   /*compute light per vertex*/
   x = gl_Vertex.x;
}


// in fragment shader
varying float x;
void main() {
  float stripe = sin(10.0*x);
  if (stripe < 0.0) stripe = 0.25;
  else stripe = 1.0;
  gl_FragColor = stripe*gl_Color;
}
```

Let's Program Phong Shader

- **Phong shading interpolates normals rather than colors**
- **Not supported in OpenGL**
  - There is no **GL_PHONG**

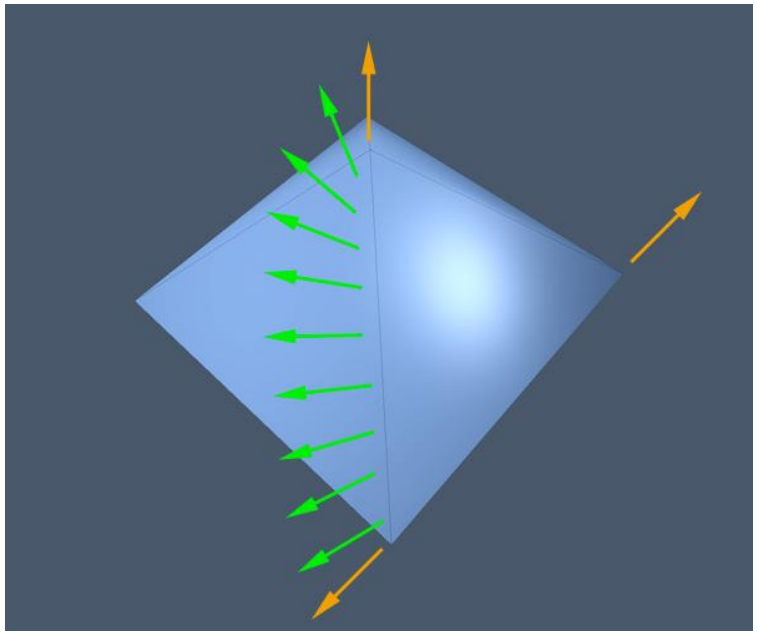


Gouraud

Phong

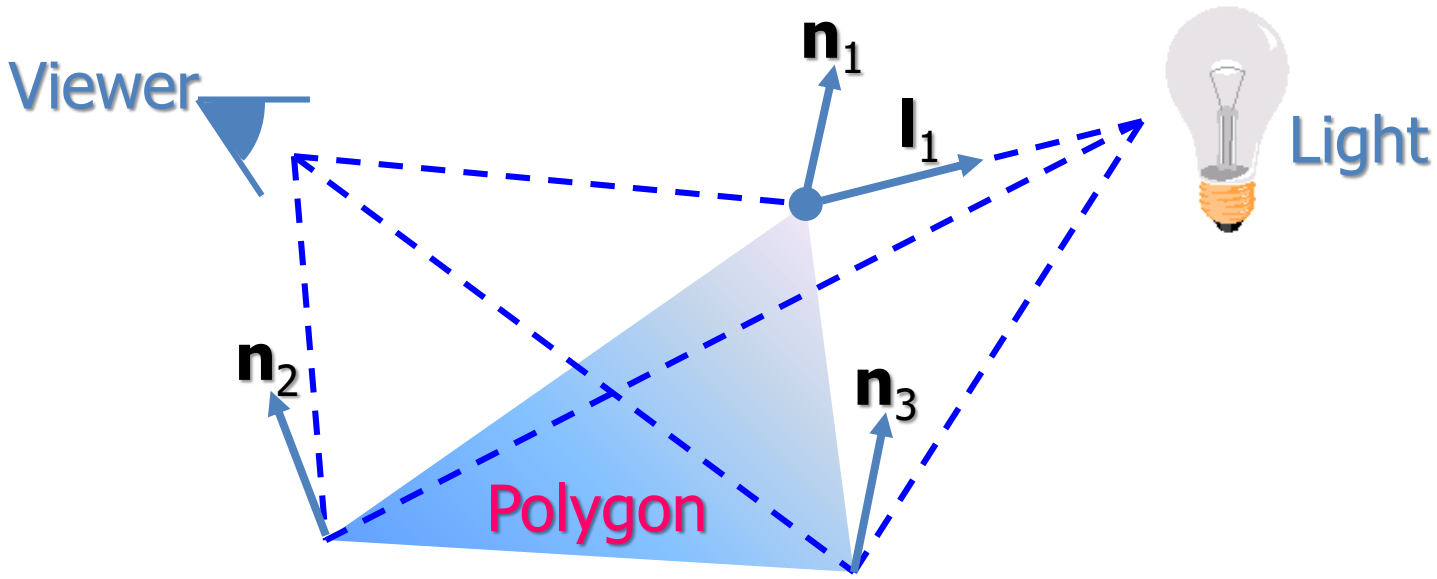
Phong Shading, 1973

- **Normal-vector interpolation shading**
  - Can capture subtle illumination effects in polygon interiors
- **Not yet supported by OpenGL/DirectX**



Phong Shading Process (1/2)

- **One radiance calculation per pixel**
  - Approximate surface normals for inside points



Pixel Normal Vector

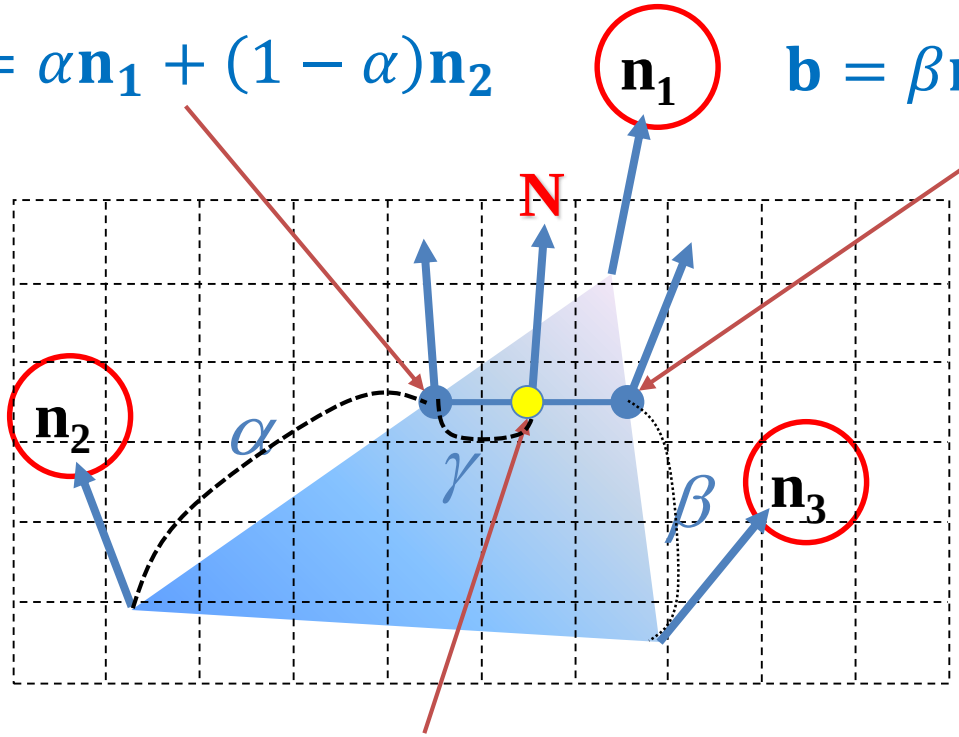
$$I = K_e + K_a L_{ga} + \sum_{lights} (Spot_i)(Att_i)(K_a L_{i,a} + K_d L_{i,d}(\mathbf{n}\cdot\mathbf{l}) + K_s L_{i,s}(\mathbf{n}\cdot\mathbf{h}_i)^{\alpha_i})$$

Phong Shading Process (2/2)

- **Bilinearly interpolate surface normals at vertices**
  - Pixel normal vector

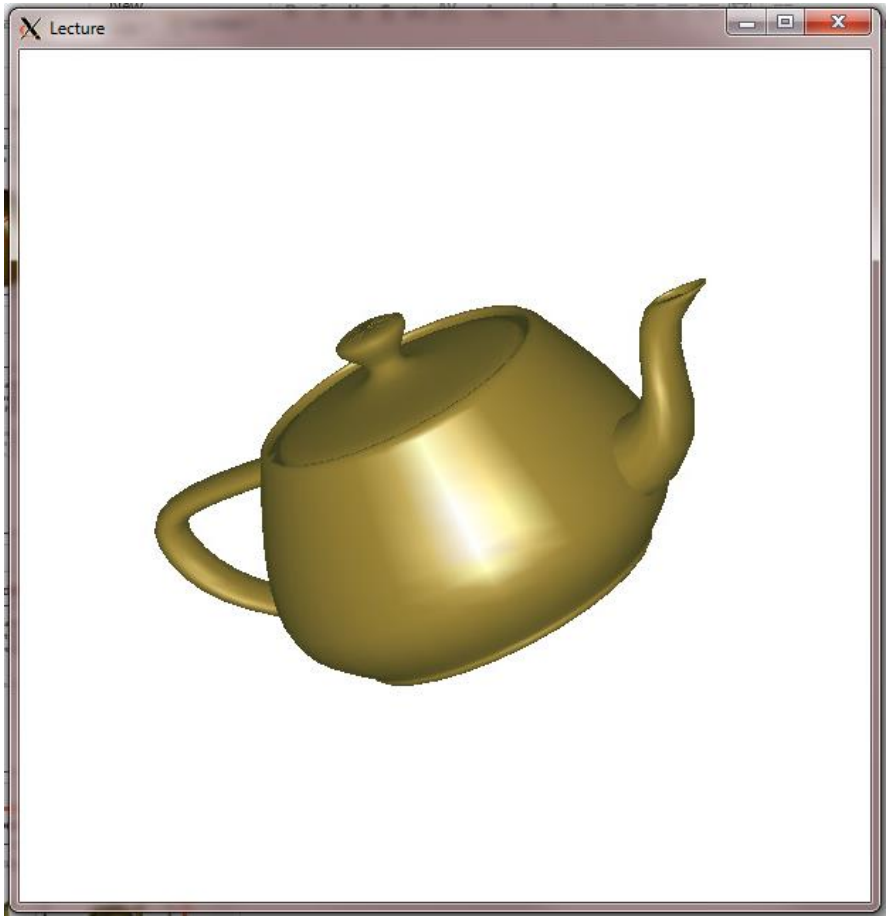
$$\mathbf{a} = \alpha\mathbf{n_1} + (1-\alpha)\mathbf{n_2}$$

$$\mathbf{b} = \beta\mathbf{n_1} + (1-\beta)\mathbf{n_3}$$



$$\mathbf{n} = (1-\gamma)\mathbf{a} + \gamma\mathbf{b}$$

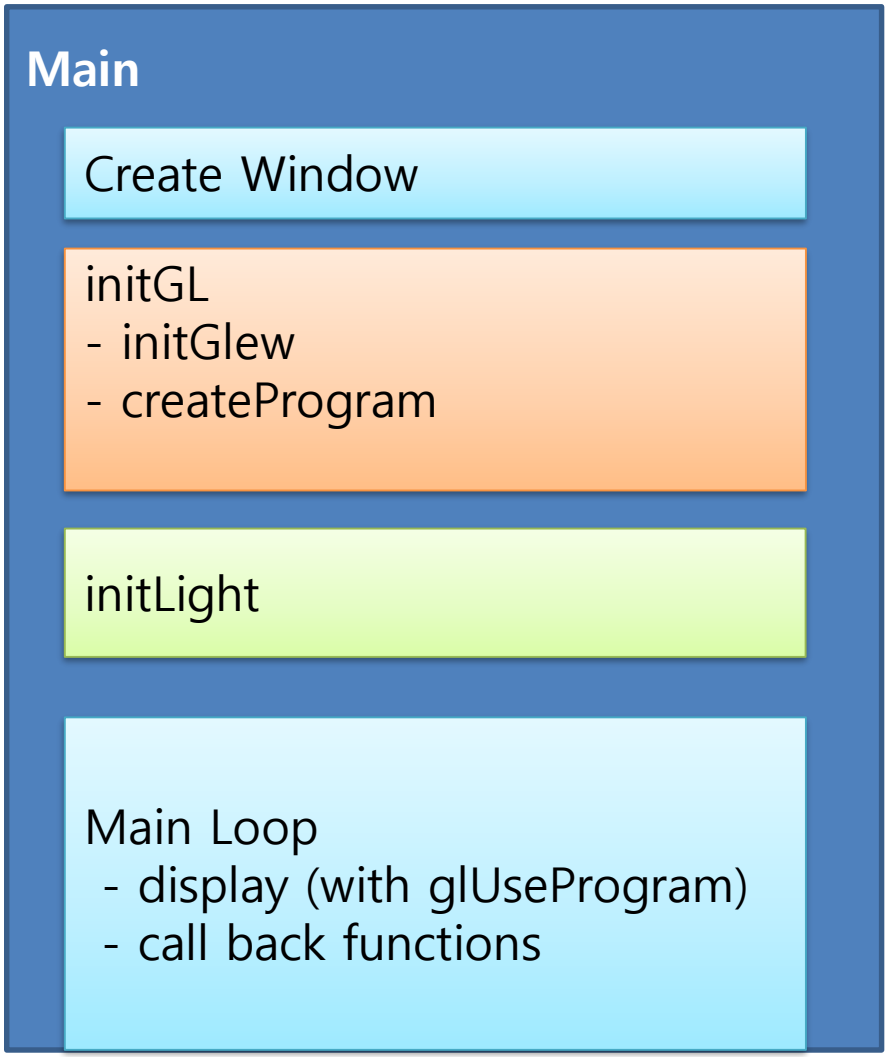
Phong Shading Teapot



Phong Shading Coding Exercise

- **Copy Sample Skeleton Code**
  - cp –r /home/share/Phong ./
  - cd Phong
- **Notepad: shader code 작성**
  - Fragment.glsl
  - Vertex.glsl
- **Compile program**
  - make
  - vglrun ./Phong

Program Flow



initLight(): Light Initialize

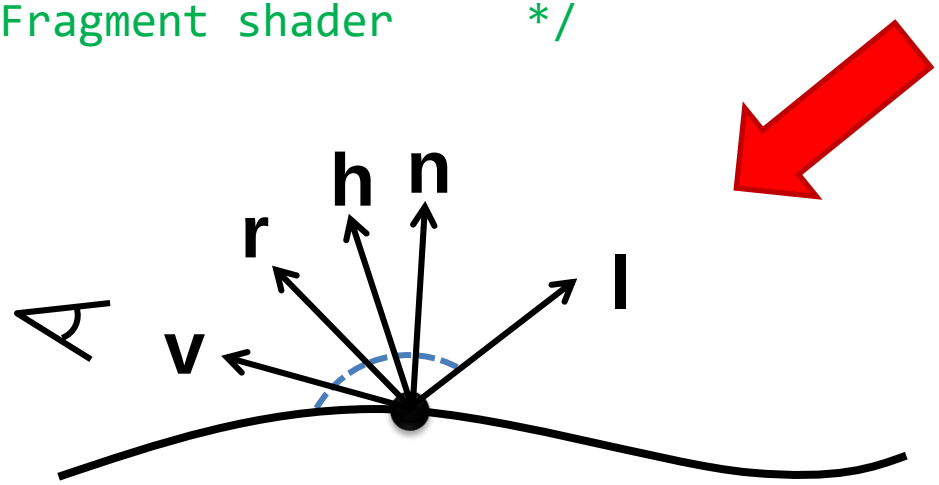
```
void initLight(){
        /*Set Light and Material Properties with Array*/
        /*Set light properties*/
        glLightfv(GL_LIGHT0, GL_AMBIENT, lightKa);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightKd);
        glLightfv(GL_LIGHT0, GL_SPECULAR, lightKs);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
        /*Set material properties*/
        glMaterialfv(GL_FRONT, GL_AMBIENT, matKa);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matKd);
        glMaterialfv(GL_FRONT, GL_SPECULAR, matKs);
        glMaterialfv(GL_FRONT, GL_SHININESS, &matShininess);
         /*Enable Light*/
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
}
```

Draw Teapot

```
void display() {
    ...
    glUseProgram(program);  // activate shader program
    glutSolidTeapot(0.5f);
    glUseProgram(0);        // deactivate shader program
    ...
}
```

Phong Shading: Vertex Shader code

```
varying vec3 normal, lightDir, halfVector;
void main() {
   normal = normalize(gl_NormalMatrix*gl_Normal);
      /*  vertex normal to fragment shader  */
   lightDir = normalize(gl_LightSource[0].position.xyz);
      /*  Light Direction Vector to Fragment shader  */
   halfVector = normalize(gl_LightSource[0].halfVector.xyz);
      /*  half Vector to Fragment shader  */
   gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
      /*  Projected Position to Fragment shader     */
}
```



Phong Shading Code: Fragment

```
varying vec3 normal, lightDir, halfVector;
void main() {
    vec3 n, h;
    float NdotL, NdotH;
    vec4 color = gl_FrontMaterial.ambient * gl_LightSource[0].ambient +
                 gl_FrontMaterial.ambient * gl_LightModel.ambient;
      /* Compute Ambient Light color        */
    n = normalize(normal);
    NdotL = max(dot(n,lightDir),0.0);
    if (NdotL > 0.0) {
       color += gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse *
    NdotL;
       /*   Compute Diffuse Light color       */
       h = normalize(halfVector);
       NdotH = max(dot(n,h),0.0);
       color += gl_FrontMaterial.specular * gl_LightSource[0].specular *
                                pow(NdotH, gl_FrontMaterial.shininess);
       /*    Compute Specular Light color      */
    }
    gl_FragColor = color;
}
```

$$I_{ambient} = K_a L_{g,a} + K_a L_{0,a}$$

$$I_{diffuse} = K_d L_{0,d} (\mathbf{n} \cdot \mathbf{l})$$

$$I_{specular} = K_s L_{0,s} (\mathbf{n} \cdot \mathbf{h}_0)^{\alpha_0}$$

$$= K_a L_{g,a} + K_a L_{0,a} + K_d L_{0,d} (\mathbf{n} \cdot \mathbf{l}) + K_s L_{0,s} (\mathbf{n} \cdot \mathbf{h}_0)^{\alpha_0}$$

Adding Stripe to Teapot

Stripe Teapot Coding Exercise

- **Notepad: shader code 수정**
  - Fragment.glsl
  - Vertex.glsl

- **vglrun ./Phong**

Adding Stripe Vertex Shader

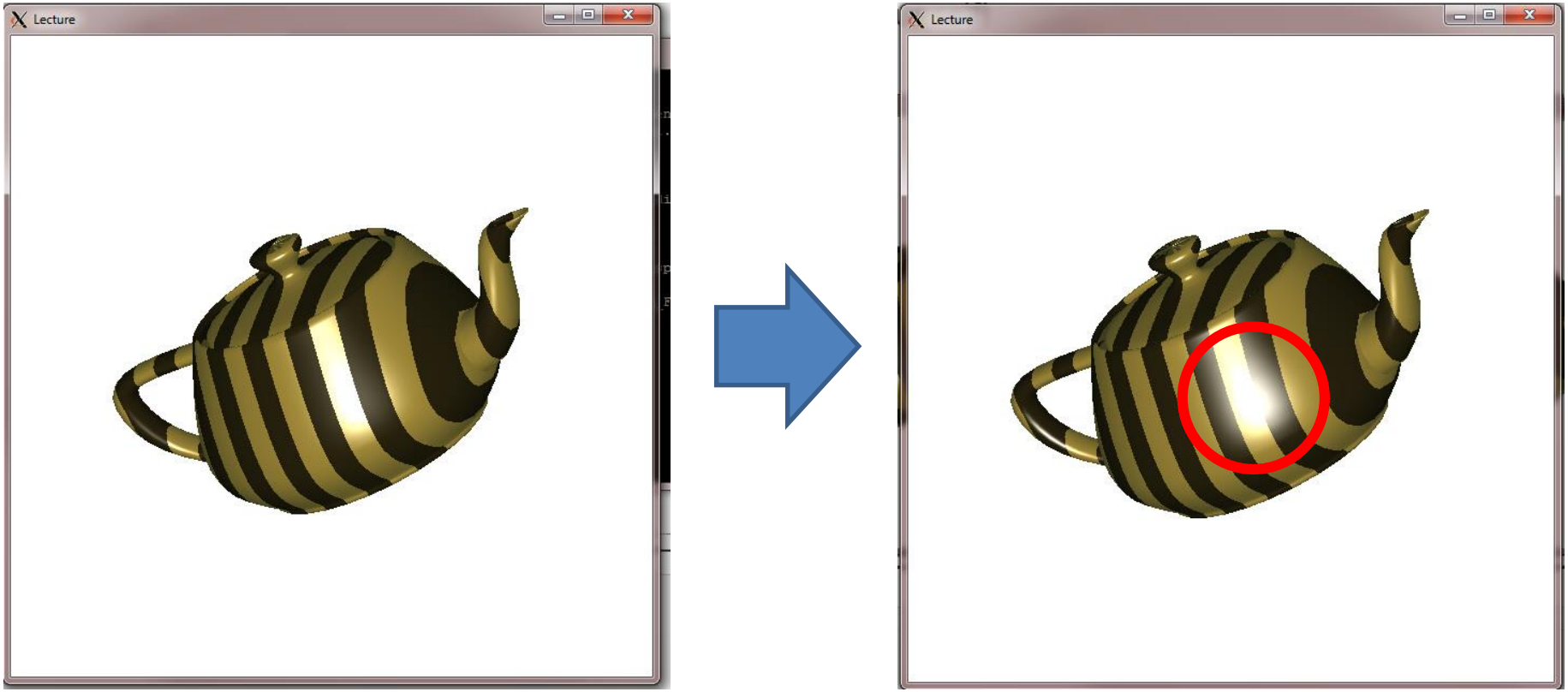
```
// in vertex shader
varying vec3 normal, lightDir, halfVector;
varying float x;
void main() {
  normal = normalize(gl_NormalMatrix*gl_Normal);
  lightDir = normalize(gl_LightSource[0].position.xyz);
  halfVector = normalize(gl_LightSource[0].halfVector.xyz);
  x = gl_Vertex.x;
  gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Adding Stripe Fragment Shader

```
// in fragment shader
varying vec3 normal, lightDir, halfVector;
varying float x;
void main() {
  ...
  float stripe = sin(10.0*x);
  if (stripe < 0.0) stripe = 0.25;
  else stripe = 1.0;
  color *= stripe;
  gl_FragColor = color;
}
```

Other Striped Example

- **Modified Fragment Shader can make following effect.**
  - Try it your self!



Solution of Other Example

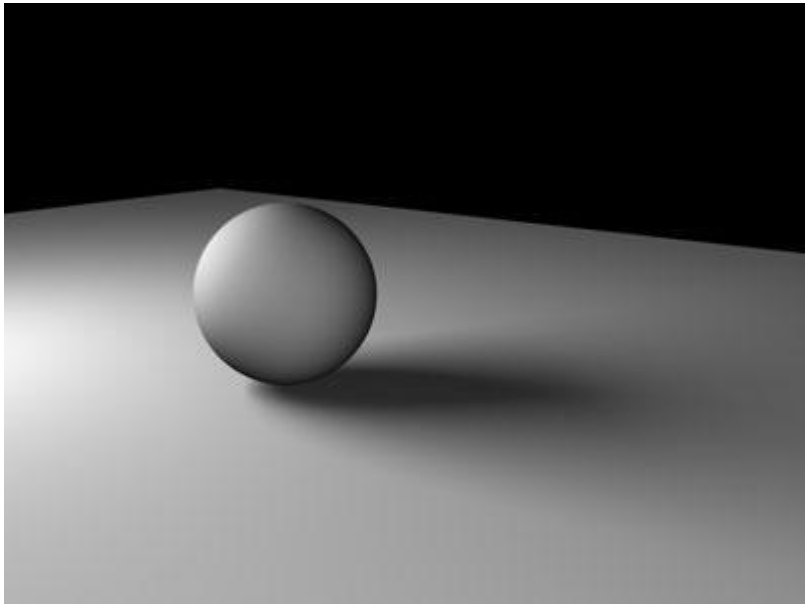
- You should compute stripe before computing specular light.

```
// in fragment shader
varying vec3 normal, lightDir, halfVector;
varying float x;
void main() {
  ...
      float stripe = sin(10.0*x);
      if (stripe < 0.0) stripe = 0.25;
      else stripe = 1.0;
      /*Compute Ambient*/
      /*Compute Diffuse*/
      color*= stripe;
      /*Compute Specular*/
      gl_FragColor = color;
}
```

Light Model @ Shader

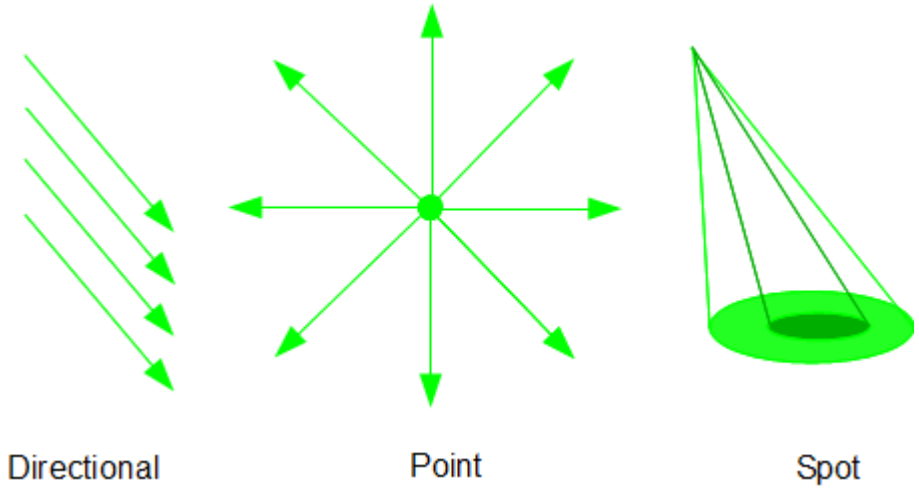
GLSL Light

- When rendering with GLSL, you have to implement **lights yourself.**



Light types

- **OpenGL lights**
  - Directional light
  - Point light
  - Spot light



Creating light sources in OpenGL

- **Point light source**

```
GLfloat light_position[] = {0.0, 0.0, 10.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- **Directional light source**

```
GLfloat light_position[] = {0.0, 0.0, 1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- **Spotlight light source**

```
GLfloat sd[] = {0.0, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, sd);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);
```

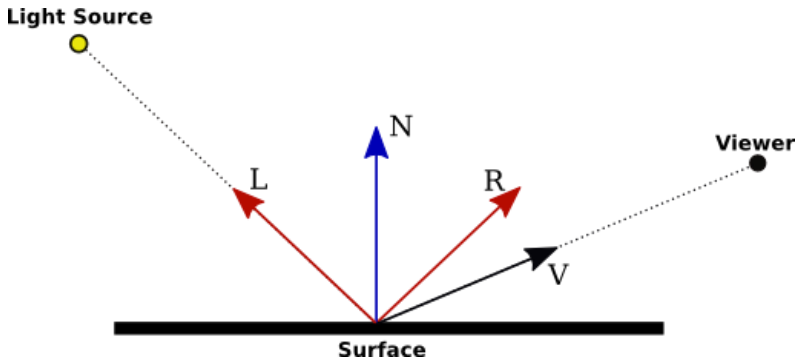
Shading Eq. in OpenGL

- **Shading equation.**

$$I = I_{emissive} + I_{ambient} + I_{diffuse} + I_{specular}$$

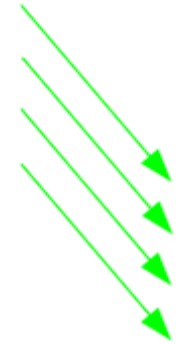
$$= K_e + K_a L_{ga} + \sum_{lights} (Spot_i)(Att_i)(K_a L_{i,a} + K_d L_{i,d} (\mathbf{n} \cdot \mathbf{l}) + K_s L_{i,s} (\mathbf{n} \cdot \mathbf{h}_i)^{\alpha_i})$$

- **In GLSL…..**
  - Implement!



Directional lights with Shader

- **Directional light**
  - Light direction is the same everywhere.



-Vertex Shader

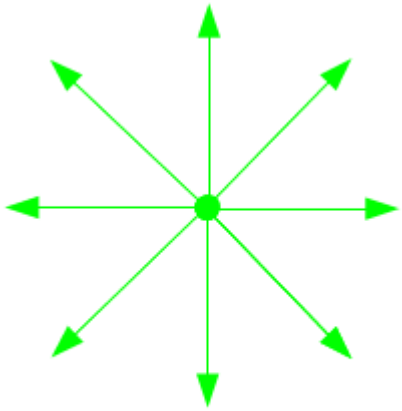
```
gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
pos=gl_ModelViewMatrix*gl_Vertex;
normal = normalize(gl_NormalMatrix*gl_Normal);
lightDir = gl_LightSource[0].position.xyz;
```

-Fragment Shader

```
vec3 L1=normalize((lightDir).xyz);
float NdotL = max(dot(normalize(normal),L1),0.0);
float intensity=NdotL;
color+=vec3(intensity);
```

Point lights with Shader

- **Point light**
  - Should compute the light vector for each surface point.



-Vertex Shader

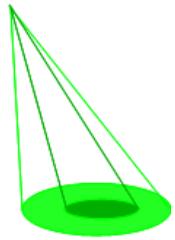
```
gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
pos=gl_ModelViewMatrix*gl_Vertex;
normal = normalize(gl_NormalMatrix*gl_Normal);
lightPos = gl_LightSource[0].position.xyz;
```

-Fragment Shader

```
vec3 L1=normalize((pos-lightPos).xyz);
float NdotL = max(dot(normalize(normal),L1),0.0);
float intensity=NdotL;
color+=vec3(intensity);
```

Spot lights with Shader

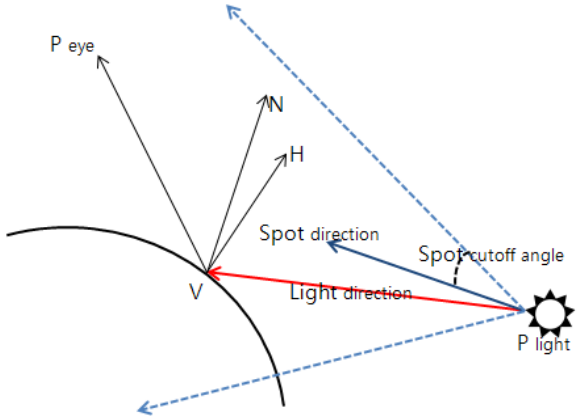
- **Spot light**
  - Position + Spot direction + Cut-off Angle + (attenuation factor)



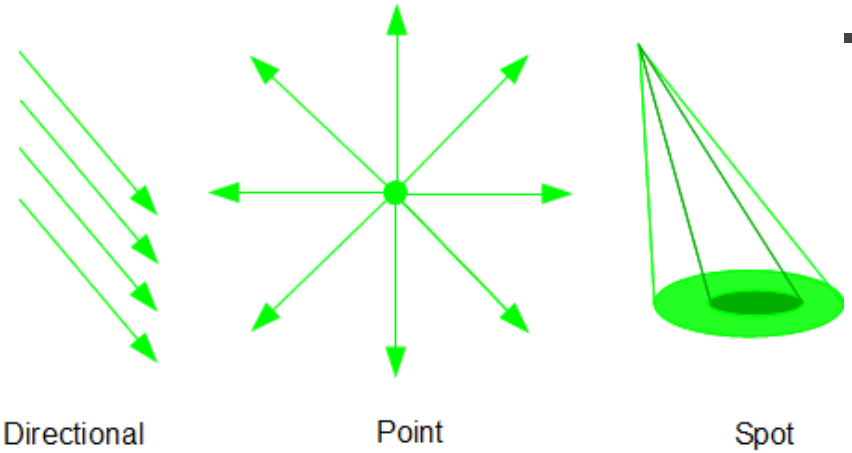
Spot

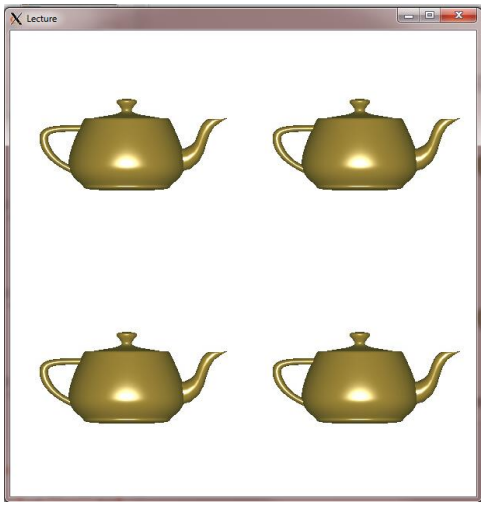
- **Calculate shading as if it is a positional light**
  - Optionally, can have an attenuation factor.

```
-Fragment Shader
vec3 L1=normalize((pos-lightPos).xyz);
float NdotL = max(dot(normalize(normal),L1),0.0);
float cut=dot(-L1, gl_LightSource[0].spotDirection);
if(cut>=gl_LightSource[0].spotCosCutoff){
        float intensity=NdotL;
        color+=vec3(intensity*pow(cut,
        gl_LightSource[0].spotExponent));
}
```



Light Model Results





Directional

Point Light

Spot Light