Blinn-Phong Shading in OpenGL

Computer Graphics
Instructor: Sungkil Lee

Shading

- **Shading:**
  - is a process of altering the color of an object/surface/polygon in a 3D scene, based on how lights and materials interact.

Color or albedo

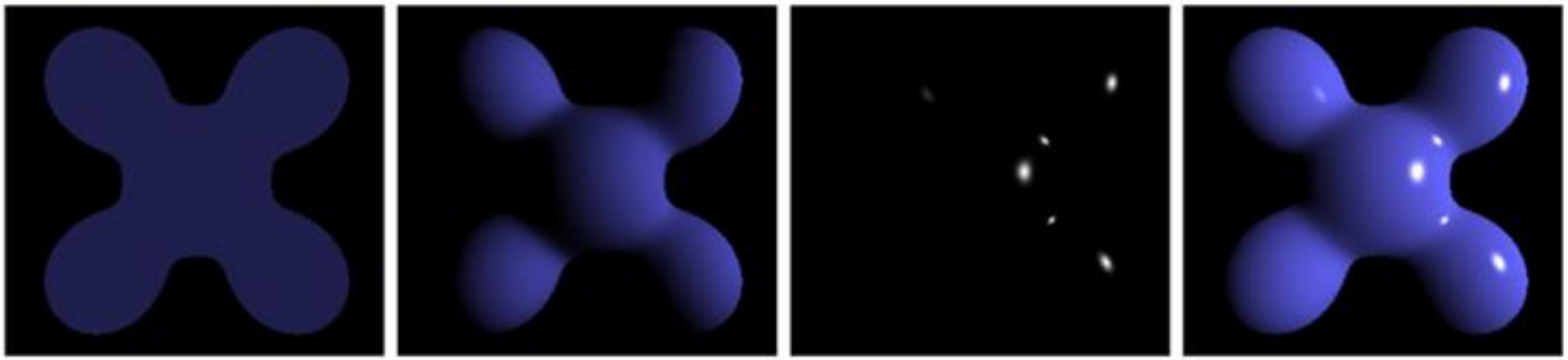
vs.

Shading

Phong Illumination Model

- **A simple empirical model that**
  - can be computed rapidly in the pipeline approach.
  - the form does not have perfect physical justification.

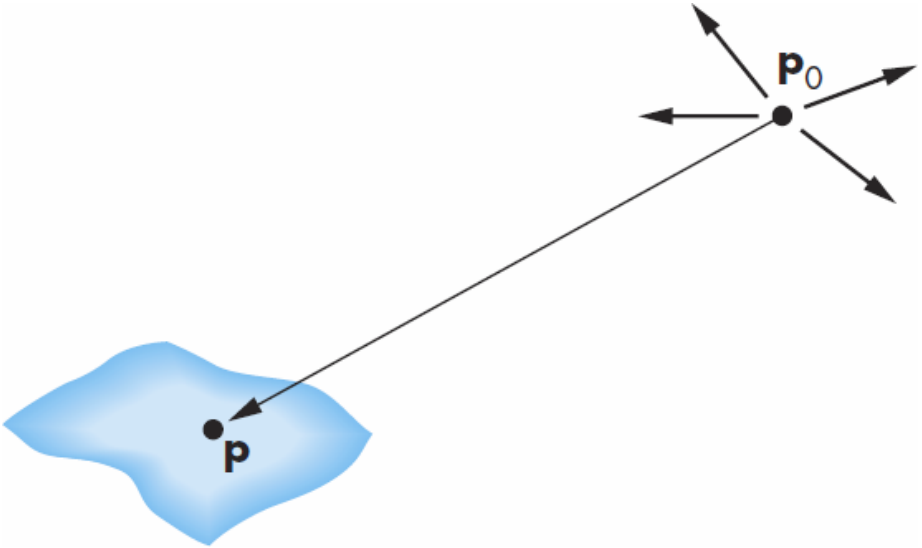
- **Three illumination components:**
  - Ambient, diffuse, and specular



Light Source Models

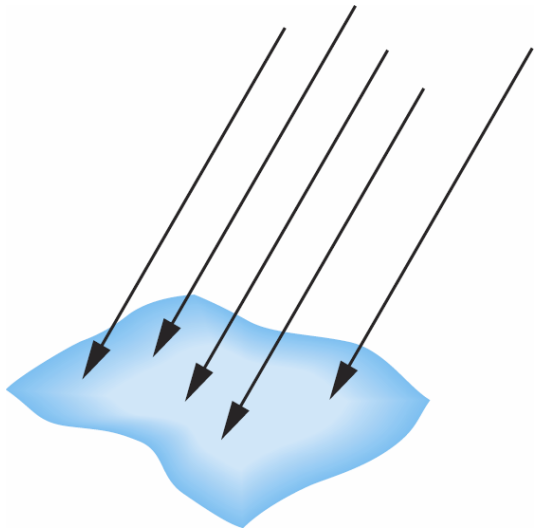
Light Source Models

- **Point light source**
  - Model with position and color.
  - The light source position is a point.
  - We may apply distance attenuation, inversely scaling with the square of distances from a light source to surface points.



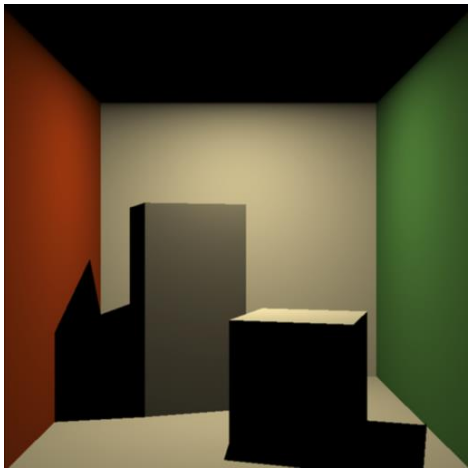
Light Source Models

- **Distant (directional) light source**
  - Infinite distance away (e.g., parallel sunlight)
  - The light source has only a direction vector.
    - In case of HC (i.e., vec4), the last element w is 0.



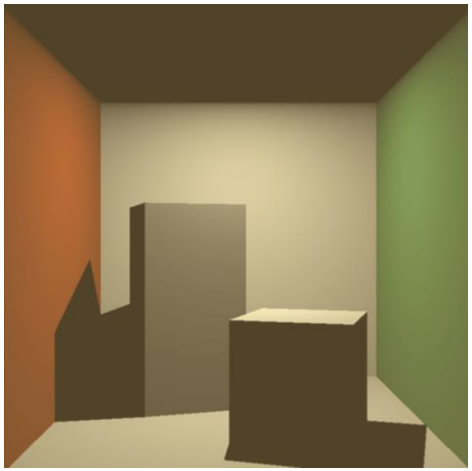
Light Source Models

- **Ambient light**
  - Same amount of light everywhere in a scene, like a shading offset
  - Can model contribution of many sources and indirect reflections.
  - This, however, is significant approximation, and limited in practice.



Direct

Ambient



Direct + Ambient

Light Source Definition

- **Each light source has:**
  - a position and 3 color terms

- **4-D Position:** $\mathbf{l} = (x, y, z, w)$
  - (x, y, z) indicates the position or direction to the light source
  - $w = 1$ for point/spot lights, $w = 0$ for directional lights

- **Three color terms**
  - Ambient colors: $\mathbf{I}_a$
  - Diffuse colors: $\mathbf{I}_d$
  - Specular colors: $\mathbf{I}_s$

Light Source Definition

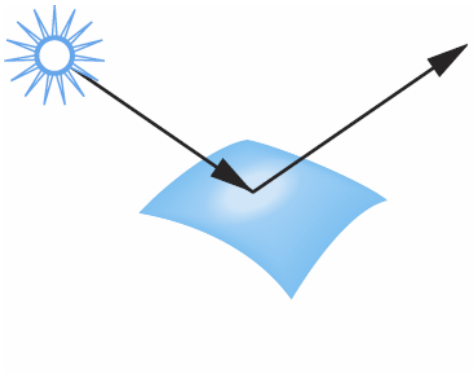
- **An example layout:**

```
struct light_t
{
  vec4 position = vec4( 10.0f, -10.0f, 10.0f, 0.0f ); // directional
  vec4 ambient  = vec4( 0.2f, 0.2f, 0.2f, 1.0f );
  vec4 diffuse  = vec4( 0.8f, 0.8f, 0.8f, 1.0f );
  vec4 specular = vec4( 1.0f, 1.0f, 1.0f, 1.0f );
};
```

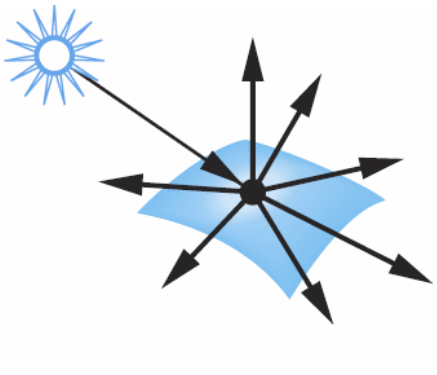
Material Models

Types of Surface Materials

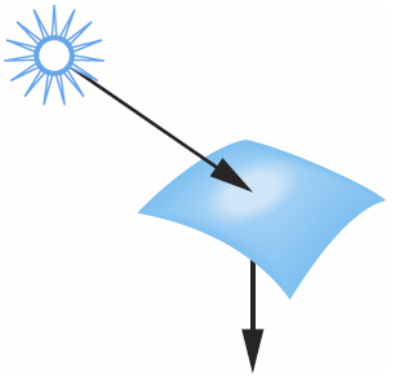
- **The types of surface materials**
  - In local lighting models, translucent surfaces are not treated well.



smooth surface
(specular reflection)



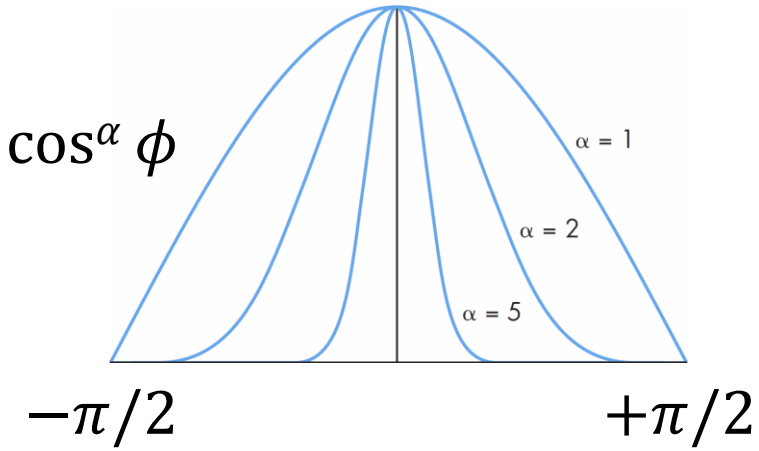
rough surface
(diffuse reflection)



translucent surface

Materials

- **Material properties match light source properties**
  - 3 reflection coefficient vectors: $\mathbf{k}_a, \mathbf{k}_d, \mathbf{k}_s \in [0,1]$
  - Shininess coefficient α



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

Materials

- **A simple example layout:**

```
struct material_t
{
    vec4  ambient  = vec4( 0.2f, 0.2f, 0.2f, 1.0f );
    vec4  diffuse  = vec4( 0.8f, 0.8f, 0.8f, 1.0f );
    vec4  specular = vec4( 1.0f, 1.0f, 1.0f, 1.0f );
    float shininess = 1000.0f;
};
```

Materials

- **Material definition with textures**
  - If geometry uses textures for color or others (bump mapping, using cube map, ...), material has additional properties.
  - An example layout with texture maps:

```
struct texmaterial_t : public material_t
{
    GLuint diffuse_texture;
    GLuint bump_texture;
    GLuint cubemap_texture;
};
```

Update()

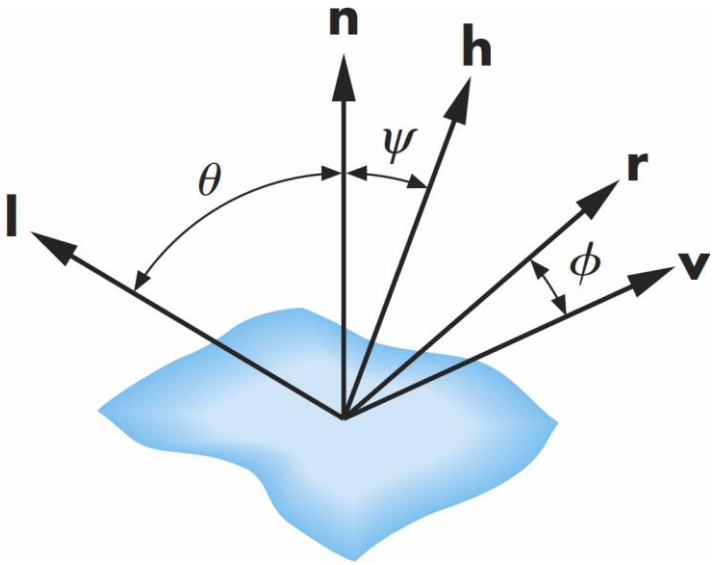
- **Update uniform variables**

```
void update()
{
   // setup light properties
   glUniform4fv( glGetUniformLocation( program, "light_position" ), 1, light.position );
   glUniform4fv( glGetUniformLocation( program, "Ia" ), 1, light.ambient );
   glUniform4fv( glGetUniformLocation( program, "Id" ), 1, light.diffuse );
   glUniform4fv( glGetUniformLocation( program, "Is" ), 1, light.specular );

   // setup material properties
   glUniform4fv( glGetUniformLocation( program, "Ka" ), 1, material.ambient );
   glUniform4fv( glGetUniformLocation( program, "Kd" ), 1, material.diffuse );
   glUniform4fv( glGetUniformLocation( program, "Ks" ), 1, material.specular );
   glUniform1f( glGetUniformLocation( program, "shininess" ), material.shininess );
}
```

Blinn-Phong Shaders in GLSL

Blinn-Phong Illumination Model

- **Uses four vectors**
  - To light source: $\mathbf{l}$ (a vector ***not from light source but to light source***)
  - To viewer: $\mathbf{v}$
  - Normal vector: $\mathbf{n}$ (will be discussed later in more details)
  - Halfway vector: $\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{|\mathbf{l}+\mathbf{v}|}$

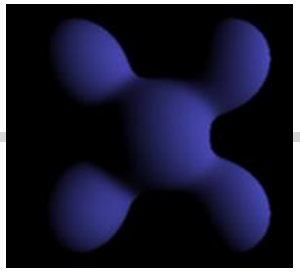


Ambient Reflections

- **Ambient reflections**
  - Amounts of reflected lights depend on both the color of the lights and the material properties (i.e., reflectance; $\mathbf{k}_a$) of the object.
  - This is a coarse approximation to the secondary (indirect) reflections among the surfaces.

$$\mathbf{I}_{ra} = \mathbf{k}_a \mathbf{I}_a$$

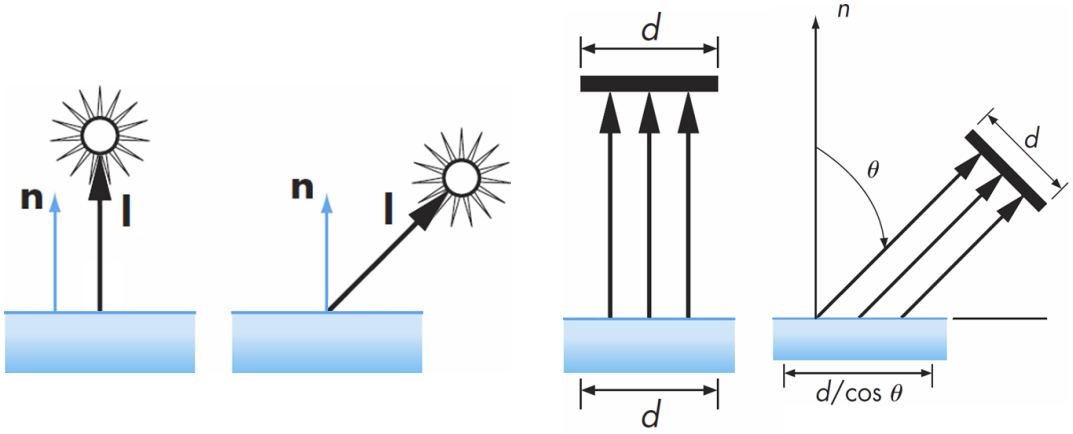
Diffuse Reflections



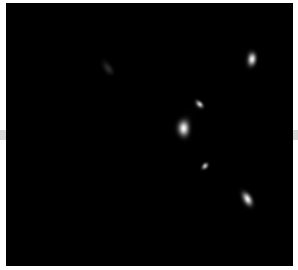
- **Diffuse reflections**
  - Similar to ambient reflection, but the angular attenuation is included here:

$$\mathbf{I}_{rd} = \max((\mathbf{l} \cdot \mathbf{n})\mathbf{k}_d \mathbf{I}_d, 0)$$

- **Angular attenuation $(\mathbf{l} \cdot \mathbf{n})$:**
  - amount of incoming light scales with the angle of incoming light.
  - reflected light $\approx \cos\theta_i = \mathbf{l} \cdot \mathbf{n}$



Specular Reflections

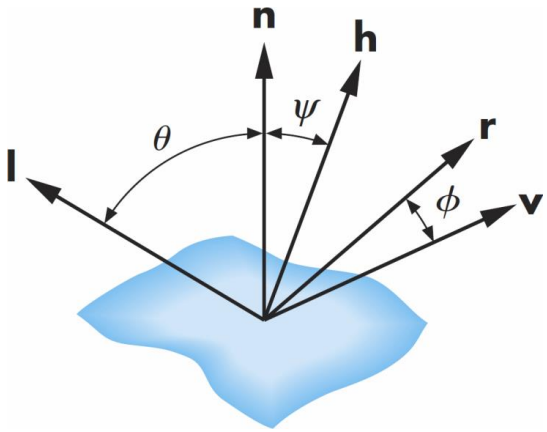


- **Specular reflections (with distance attenuation):**

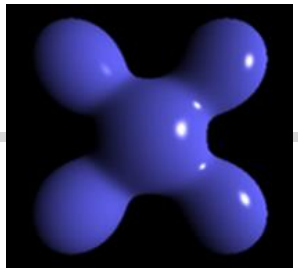
$$\mathbf{I}_{rs} = \max(\mathbf{k}_s \mathbf{I}_s (\mathbf{n} \cdot \mathbf{h})^{\beta}, 0)$$

  - The halfway vector $\mathbf{h}$ closer to the normal $\mathbf{n}$ shines more.
  - Hence, the angle between $\mathbf{h}$ the $\mathbf{n}$ can be a drop-off term.

$$\cos^{\beta} \psi = (\mathbf{n} \cdot \mathbf{h})^{\beta}$$



Putting them altogether



- **For each light source and each color component,**
  - Blinn-Phong model can be written (without distance terms) as:

$$\mathbf{I}_r = \mathbf{I}_{ra} + \mathbf{I}_{rd} + \mathbf{I}_{rs}$$
$$= \mathbf{k}_a \, \mathbf{I}_a + \max(\mathbf{k}_d(\mathbf{l} \cdot \mathbf{n})\mathbf{I}_d, 0) + \max(\mathbf{k}_s(\mathbf{n} \cdot \mathbf{h})^{\beta}\mathbf{I}_s, 0)$$

  - This model has been a standard in the old-style OpenGL.

Blinn-Phong Shaders in OpenGL

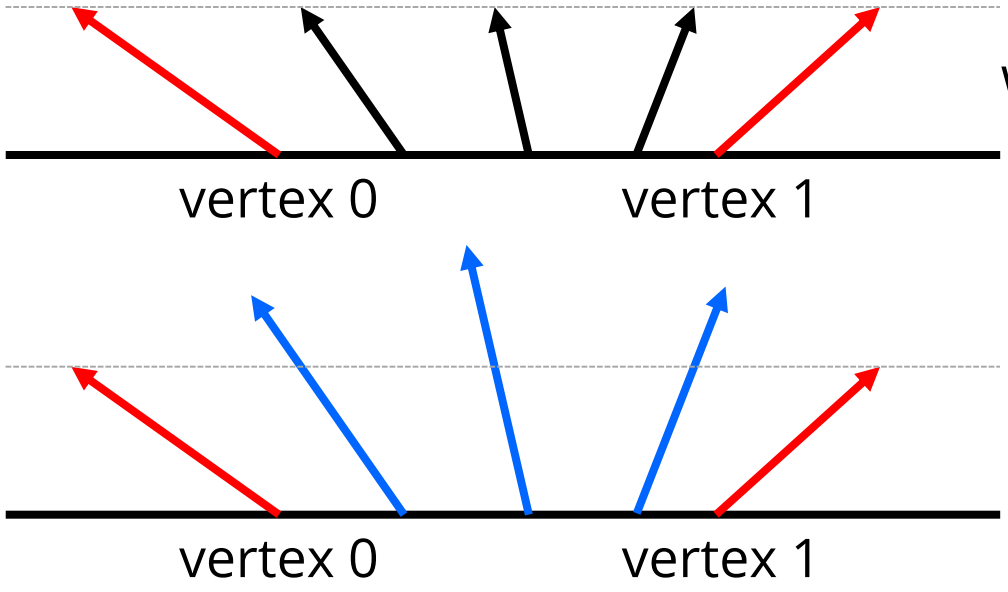
- **These example shaders demonstrate how to write Blinn-Phong shading in GLSL.**
  - The vertex shader transforms the vertex normals with model-view transformation matrix, and pass the normals to the rasterizer.
  - The rasterizer interpolates the normals, producing per-fragment normals.
  - In the fragment shader, the per-fragment normals are used for shading with Blinn-Phong illumination model.

Fragment Shader

```
// input from vertex shader
in vec4 epos;
in vec3 norm;

// the only output variable
out vec4 fragColor;

// uniform variables
uniform mat4  view_matrix;
uniform vec4  light_position, Ia, Id, Is; // light
uniform vec4  Ka, Kd, Ks;                 // material properties
uniform float shininess;

void main()
{
   // light position in the eye-space coordinate
   vec4 lpos = view_matrix*light_position;

   vec3 n = normalize(norm);   // norm interpolated via rasterizer should be normalized again here
   vec3 p = epos.xyz;          // 3D position of this fragment
   vec3 l = normalize(lpos.xyz-(lpos.a==0.0?vec3(0):p));// lpos.a==0 means directional light
   vec3 v = normalize(-p);     // eye-epos = vec3(0)-epos
   vec3 h = normalize(l+v);    // the halfway vector

   vec4 Ira = Ka*Ia;                                 // ambient reflection
   vec4 Ird = max(Kd*dot(l,n)*Id,0.0);               // diffuse reflection
   vec4 Irs = max(Ks*pow(dot(h,n),shininess)*Is,0.0); // specular reflection

   fragColor = Ira + Ird + Irs;
}
```

normalize(norm)?

- **Key question that arises in the fragment shader**
  - We already normalized the normal in vertex shader. But, why do we normalize it again in the fragment shader?
- **Fragment normals are linearly-interpolated vectors**
  - of vertex normals, causing their lengths are no more ones.
  - Hence, we re-normalize the fragment normal to make it unit-length.



without re-normalzation

with re-normalization

Example

- **Head model shaded with Blinn-Phong model.**

Exercises

Exercises

- **Try to compare directional and point light source.**
  - Hint: change from w=0 to w=1
- **Try to apply Phong Model**
  - Hint: change $n \cdot h$ to $r \cdot v$.
- **Try to implement Cel shading (Cartoon-like shading)**
  - Hint: discretization of diffuse shading.
- **Try to move light sources**
  - Hint: shaders do not have to be changed.
- **Try to add multiple light sources**
  - Hint: upload multiple lights and use loop in fragment shader
- **Try to load different 3D models**