Discussion 3
CSE 167

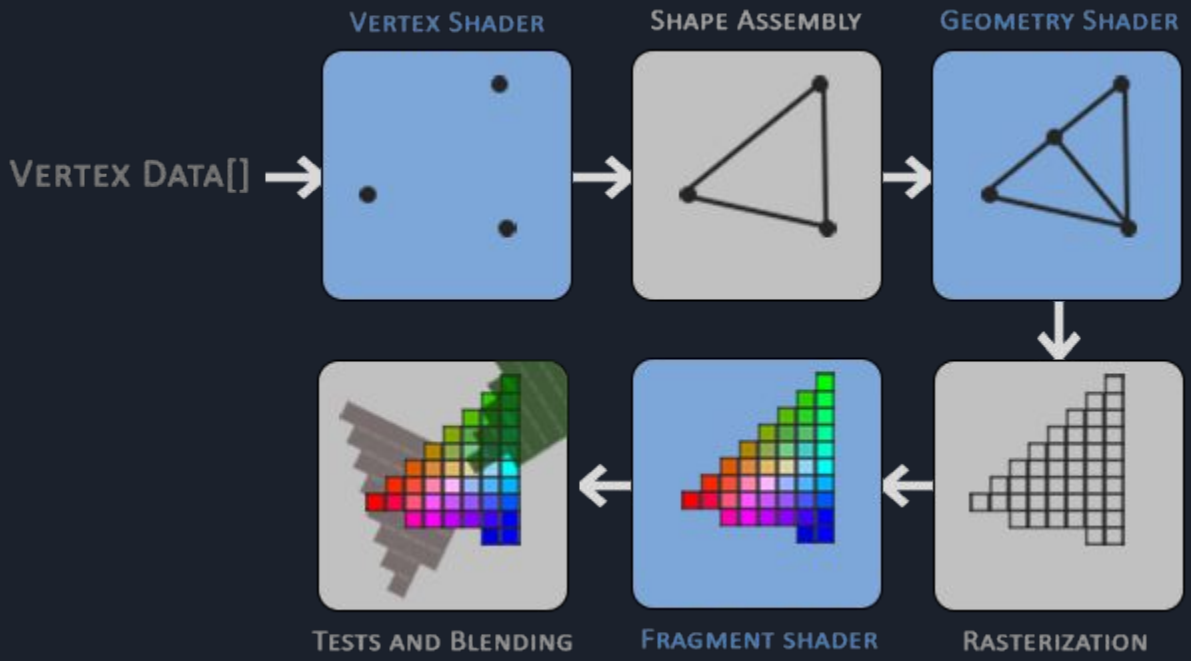
Any questions regarding part 1 and 2?

- Project 2 revisit.

Outline

- Shaders
- Lights & Materials
- Coordinate systems for interactive control

Shaders


VERTEX SHADER
SHAPE ASSEMBLY
GEOMETRY SHADER
VERTEX DATA[]
TESTS AND BLENDING
FRAGMENT SHADER
RASTERIZATION

Shaders - Vertex Shader

```
1  #version 330 core
2  layout (location = 0) in vec3 position;
3  layout (location = 1) in vec3 normal;
4
5  uniform mat4 projection;
6  uniform mat4 view;
7  uniform mat4 model;
8
9  out vec3 normalOutput;
10 out vec3 posOutput;
11
12 void main()
13 {
14     gl_Position = projection * view * model * vec4(position, 1.0);
15
16     ...
17 }
```

TODO: Transform vertices and normals from local coordinate to world coordinate before passing it to fragment shaders.

Warning: please read here on **normal matrix** to avoid transforming normals incorrectly.

Shaders - Vertex Shader

```
2  layout (location = 0) in vec3 position;
3  layout (location = 1) in vec3 normal;
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
```

- In PointCloud.cpp, the first parameter of glVertexAttribPointer should be the same as the location number in the shader.

```
5  uniform mat4 projection;
6  uniform mat4 view;
7  uniform mat4 model;
```

```
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

- In Windows.cpp, the glUniformLocation should have the same name as the parameters in the shader.

Shaders - Fragment Shader

- normalOutput and posOutput are the output from Vertex Shader.
- You should pass light attributes (e.g. color) as glUniforms to specify the attributes of light source.
- fragColor is the final color of the pixel coming out of the shader.
- TODO: Use phong lighting and linear attenuation to calculate fragment color here.

```
1  #version 330 core
2
3  in vec3 normalOutput;
4  in vec3 posOutput;
5
6  uniform vec3 lightAttr1;
7  uniform vec3 lightAttr2;
8  ...
9
10 out vec4 fragColor;
11
12 void main()
13 {
14     vec3 ambient = ...
15
16     vec3 diffuse = ...
17
18     vec3 specular = ...
19
20     fragColor = ...
21 }
```

Shaders

- Shader loading is given in shader.cpp, please refer to this link for further understanding if interested: https://learnopengl.com/Getting-started/Hello-Triangle

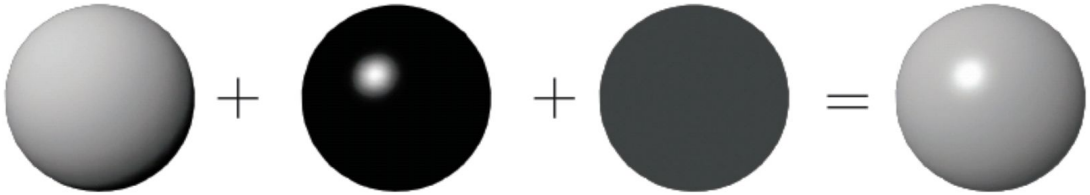
Lights and Materials

c - lighting
k - material

Local Illumination

Simplified model

- Sum of 3 components
- Covers a large class of real surfaces



UCSD

Diffuse

diffuse

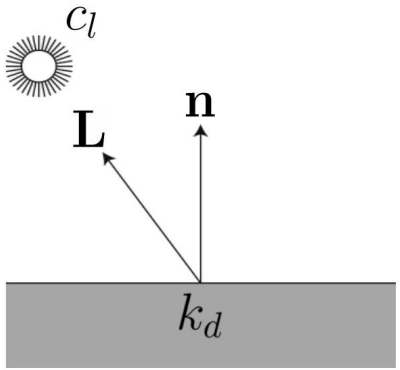
Diffuse Reflection

- Given
  - Unit (normalized!) surface normal **n**
  - Unit (normalized!) light direction **L**
  - Material diffuse reflectance (material color) k_d
  - Light color (intensity) c_l

- Diffuse color c_d is:

$$c_d = c_l k_d (\mathbf{n} \cdot \mathbf{L})$$

Proportional to cosine between normal and light



UCSD

Specular

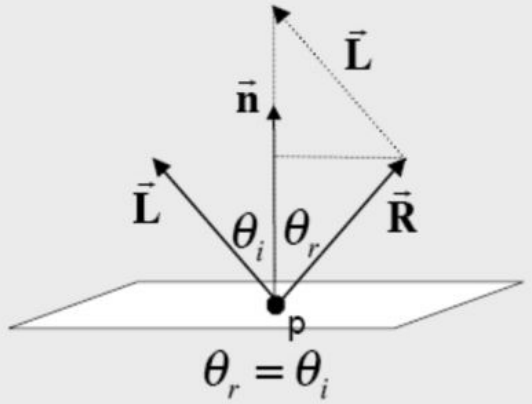
specular

Law of Reflection

- Angle of incidence equals angle of reflection

$$\vec{\mathbf{R}} + \vec{\mathbf{L}} = 2\cos\theta\ \vec{\mathbf{n}} = 2\left(\vec{\mathbf{L}}\cdot\vec{\mathbf{n}}\right)\vec{\mathbf{n}}$$

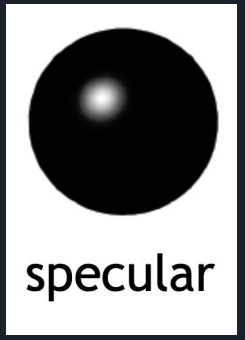
$$\vec{\mathbf{R}} = 2\left(\vec{\mathbf{L}}\cdot\vec{\mathbf{n}}\right)\vec{\mathbf{n}} - \vec{\mathbf{L}}$$



UCSD

```
12  glm::vec3 Window::eye(0, 0, 20); // Camera position.
```

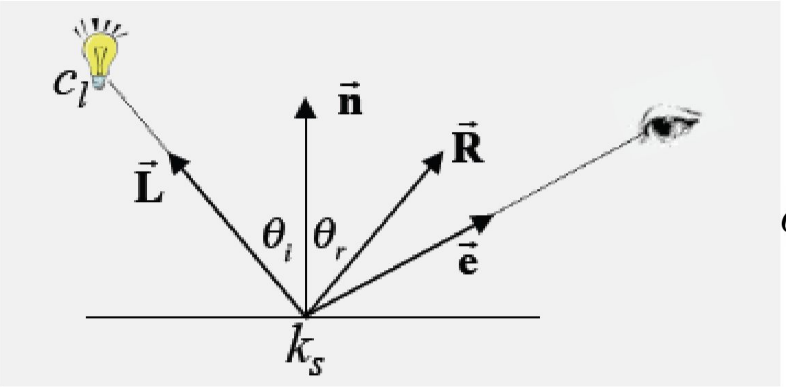
Specular



e - eye direction
(unit vector)

Phong Shading Model

- Developed by Bui Tuong Phong in 1973
- Specular reflectance coefficient k_s
- Phong exponent p
  - Greater p means smaller (sharper) highlight



$$c_s = k_s c_l (\mathbf{R} \cdot \mathbf{e})^p$$

Ambient

Ambient Light

- In real world, light is bounced all around scene
- Could use global illumination techniques to simulate
- Simple approximation
  - Add constant ambient light at each point: $k_a c_a$
  - Ambient light color: c_a
  - Ambient reflection coefficient: k_a
- Areas with no direct illumination are not completely dark

Complete Phong Shading Model

- Phong model supports multiple light sources
- All light colors *c* and material coefficients *k* are 3-component vectors for red, green, blue

$$c = \sum_i c_{l_i}(k_d(L_i \cdot n) + k_s(R \cdot e)^p + k_a)$$

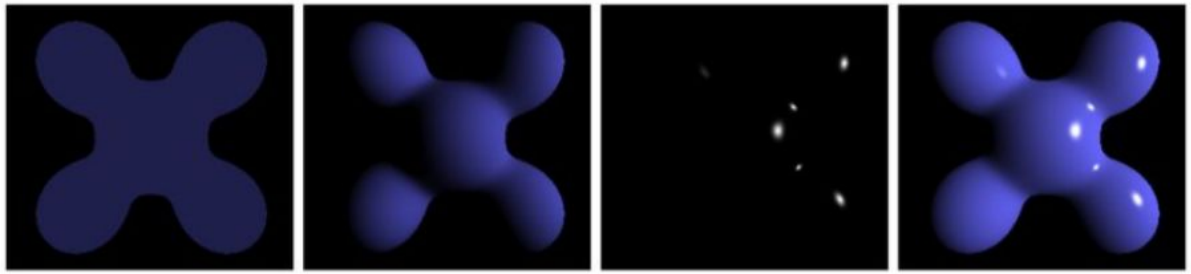
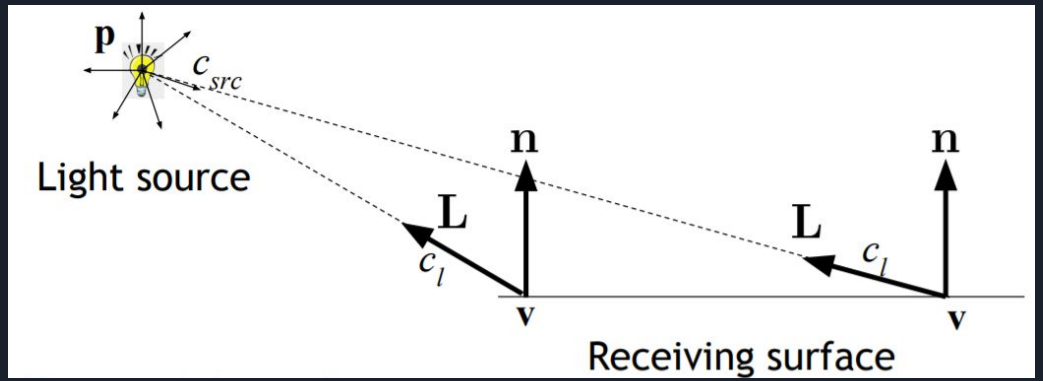
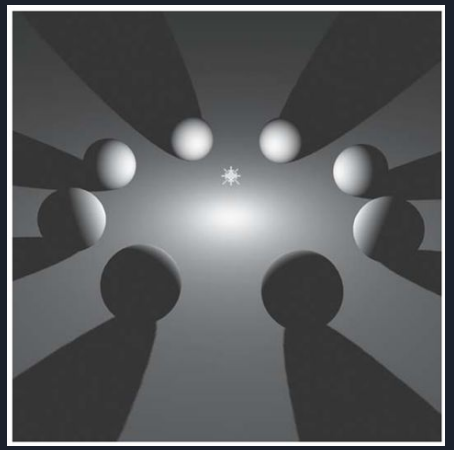


Image by Brad Smith

Point Lights

- Similar to light bulbs
- Infinitely small point radiates light equally in all directions





Light Attenuation

- Adding constant factor k to denominator for better control
- Quadratic attenuation: $k*(p-v)^2$
  - Most computationally expensive, most physically correct
- Linear attenuation: $k*(p-v)$
  - Less expensive, less accurate
- Constant attenuation: k
  - Fastest computation, least accurate

Attenuation: $$c_l = \frac{c_{src}}{\|\mathbf{p} - \mathbf{v}\|}$$

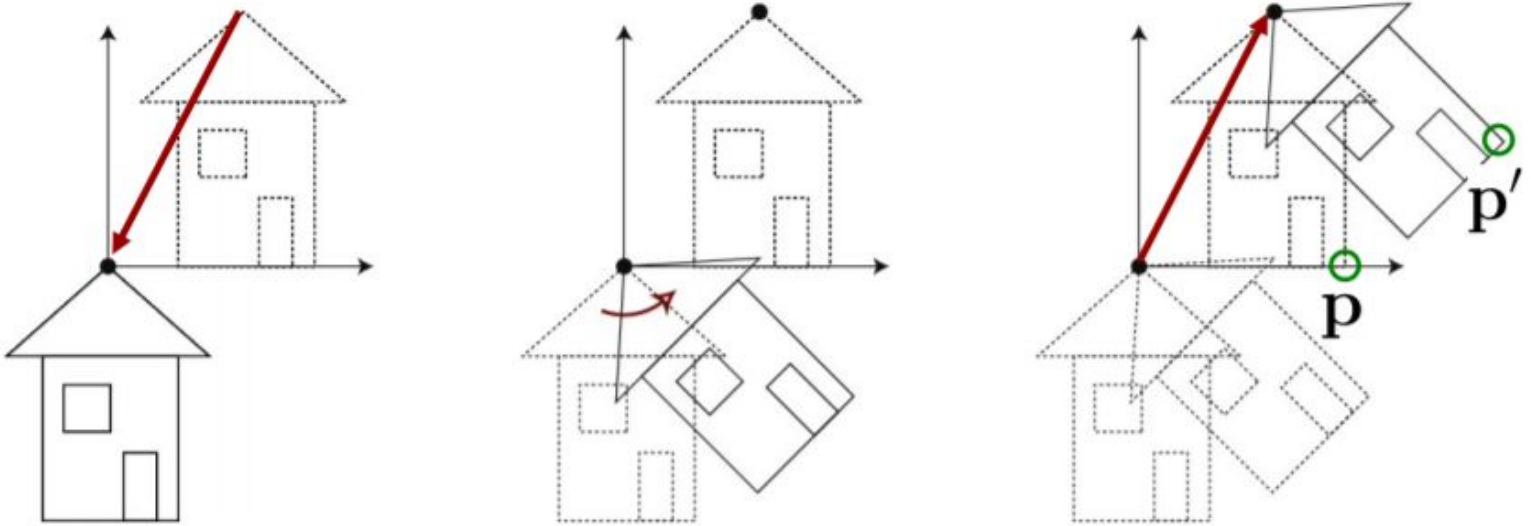
Helpful Resources

- Tutorial on how to code spot light and material is here.
  - We are using only the linear term, so we are not looking for a complete copy of this piece of code.
  - You will need to tune your color so that the object looks reasonable.

Coordinate Systems for Interactive Control

Mode	Mouse Button	Mouse Wheel (or similar)
'1'	Rotates 3D model about its center. Light source stays fixed for the viewer.	Scales 3D model about its center. Light source stays fixed for the viewer.
'2'	Rotates the light source around the center of the 3D model. 3D model stays fixed for the viewer.	Moves the light source closer to or further from the center of the 3D model.
'3'	Rotates both 3D model and light source by the same amount.	Scales the 3D model about its center and moves the light source like above.

Rotating point p around a pivot point



1. Translation T 2. Rotation R 3. Translation T^{-1}

$$p' = T^{-1} R T p$$

Demo Time

Any questions?