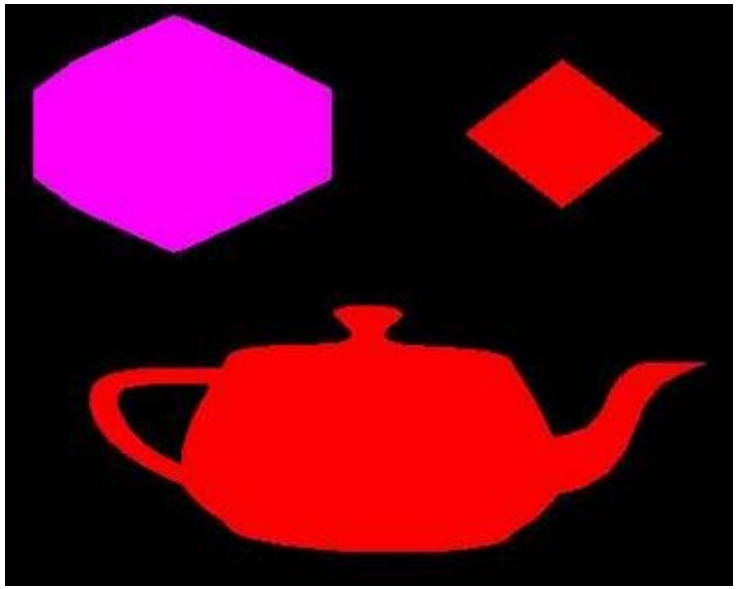
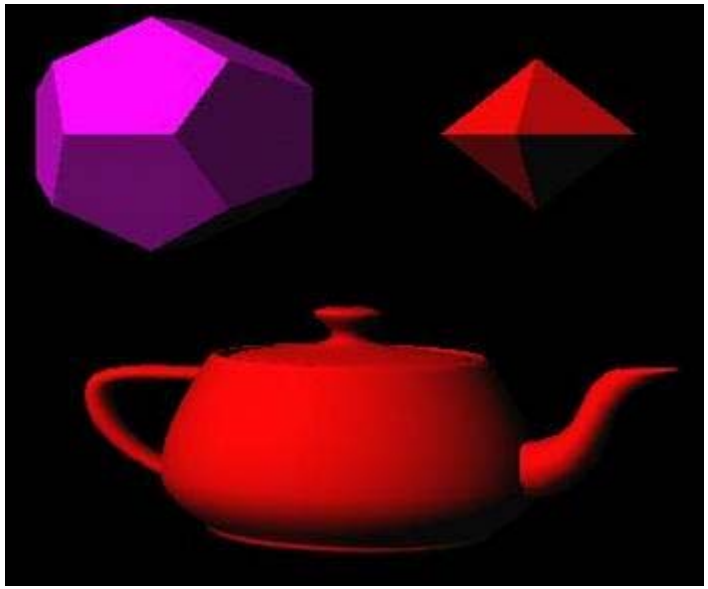
Lighting

527970
Fall 2020
11/12/2020
Kyoung Shin Park
Computer Engineering
Dankook University

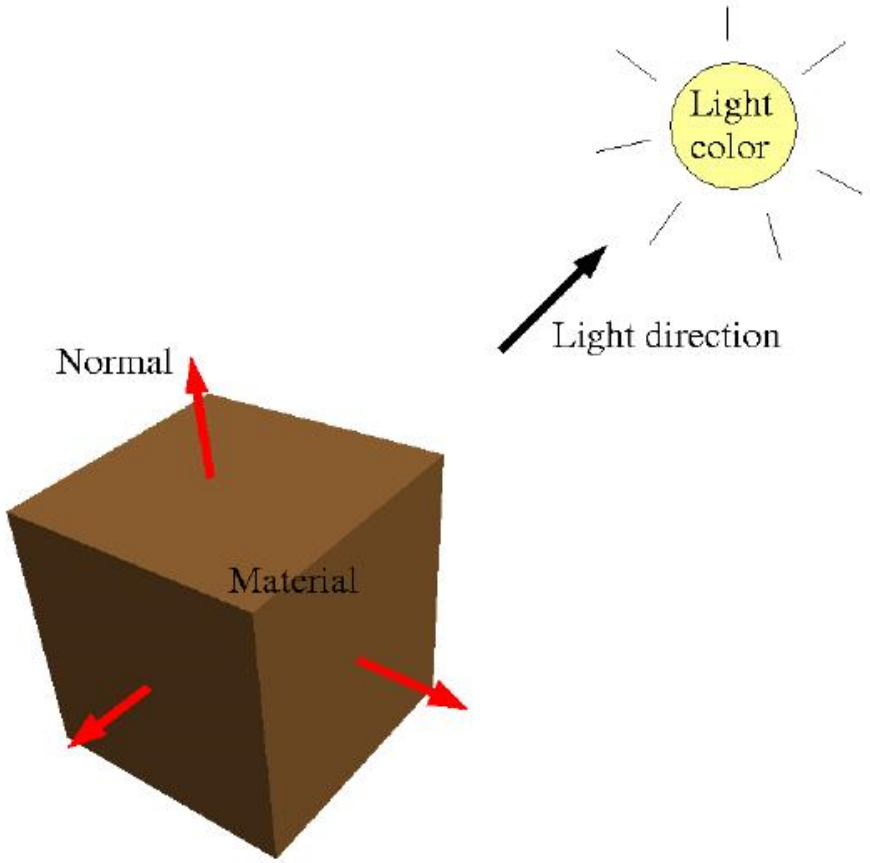
OpenGL Lighting

- OpenGL Lighting
  - Light source
  - Materials
  - Surface normals



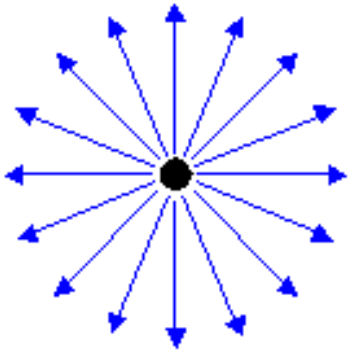


OpenGL Lighting



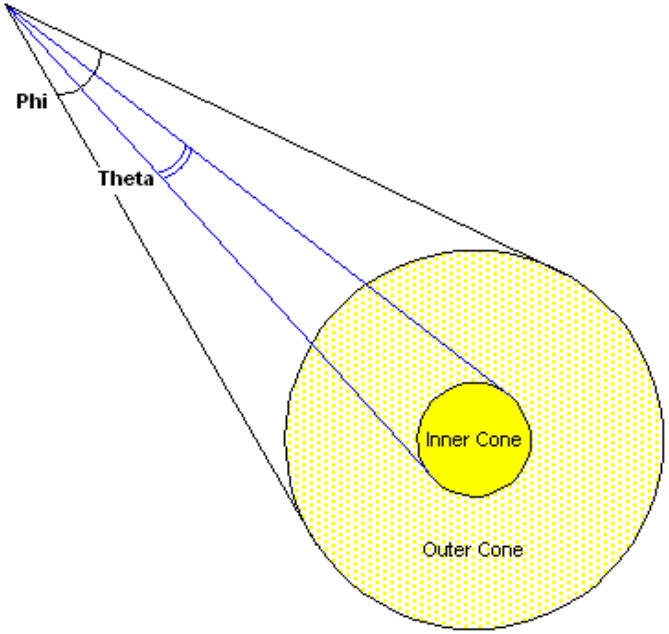
OpenGL Lighting

- OpenGL light source
  - Ambient lights
  - Point lights
  - Directional lights
  - Spot lights



Point light

Directional light



Spot light

OpenGL Light Sources

- Light sources has a position and color (ambient/diffuse/specular).
- The intensity of the light source is determined by the intensity of the color.

```
struct lightSource {
  vec4 position;
  vec4 diffuse;
  vec4 specular;
  float constantAttenuation, linearAttenuation,
   quadraticAttenuation;
  float spotCutoff, spotExponent;
  vec3 spotDirection;
};
```

OpenGL Light Source Position

- Directional light(or infinite light) and Point light (or local light)
  - Directional light source has lights with the same direction.
  - Point light source is a light coming from a specific point in space.
  - If the 4th value of the position of the light source is 0, it is the directional light, and if it's 1, it is point light.
- GLSL 예제:

```
if (light0.position.w == 0.0)  { // directional light
      attenuation = 1.0; // no attenuation
      lightDirection = normalize(vec3(light0.position));
}
else  { // point or spot light (or other kind of light)
    ...
}
```

OpenGL Light Source Position

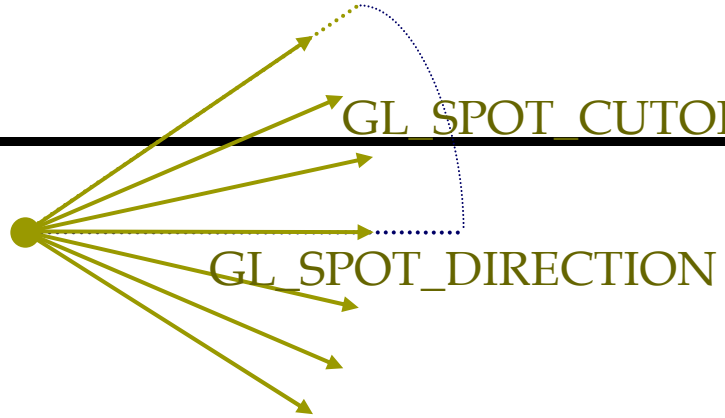
- The position of the light source is affected by the world transformation.
- It is recommended to specify the light source after the camera transformation, so that it can be defined in the world coordinate system.
- Light is also be moved by world transformation matrix, like objects.

OpenGL Light Sources

- Light intensity attenuation according to distance
  - Physical attenuation: $I(P) = \frac{I(P_0)}{\|P - P_0\|^2}$
  - OpenGL attenuation:
    - Default a = 1, b = 0, c = 0 $I(P) = \frac{I(P_0)}{a + bd + cd^2}$

```
vec3 vertexToLightSource = vec3(light0.position - worldPosition);
float distance = length(vertexToLightSource);
lightDirection = normalize(vertexToLightSource);
attenuation = 1.0 / (light0.constantAttenuation
                + light0.linearAttenuation * distance
                + light0.quadraticAttenuation * distance * distance);
```

OpenGL Light Sources



- Spot light source
  - Spot light direction
  - Spot light cutoff
  - Using a higher spot light exponent will result in more spot lights.

```
// continue from light attenuation..
if (light0.spotCutoff <= 90.0) { // spotlight
float clampedCosine
        = max(0.0, dot(lightDirection, normalize(light0.spotDirection)));
if (clampedCosine < cos(radians(light0.spotCutoff))) // outside of spotlight
    cone {
        attenuation = 0.0;
} else  {
        attenuation = attenuation * pow(clampedCosine,light0.spotExponent);
}
```

OpenGL Multiple Lights

- In OpenGL, multiple light sources can be activated simultaneously by specifying GL_LIGHT0, GL_LIGHT1, ...
- Using too many lights can increase the amount of lighting computation and thus slow down rendering.

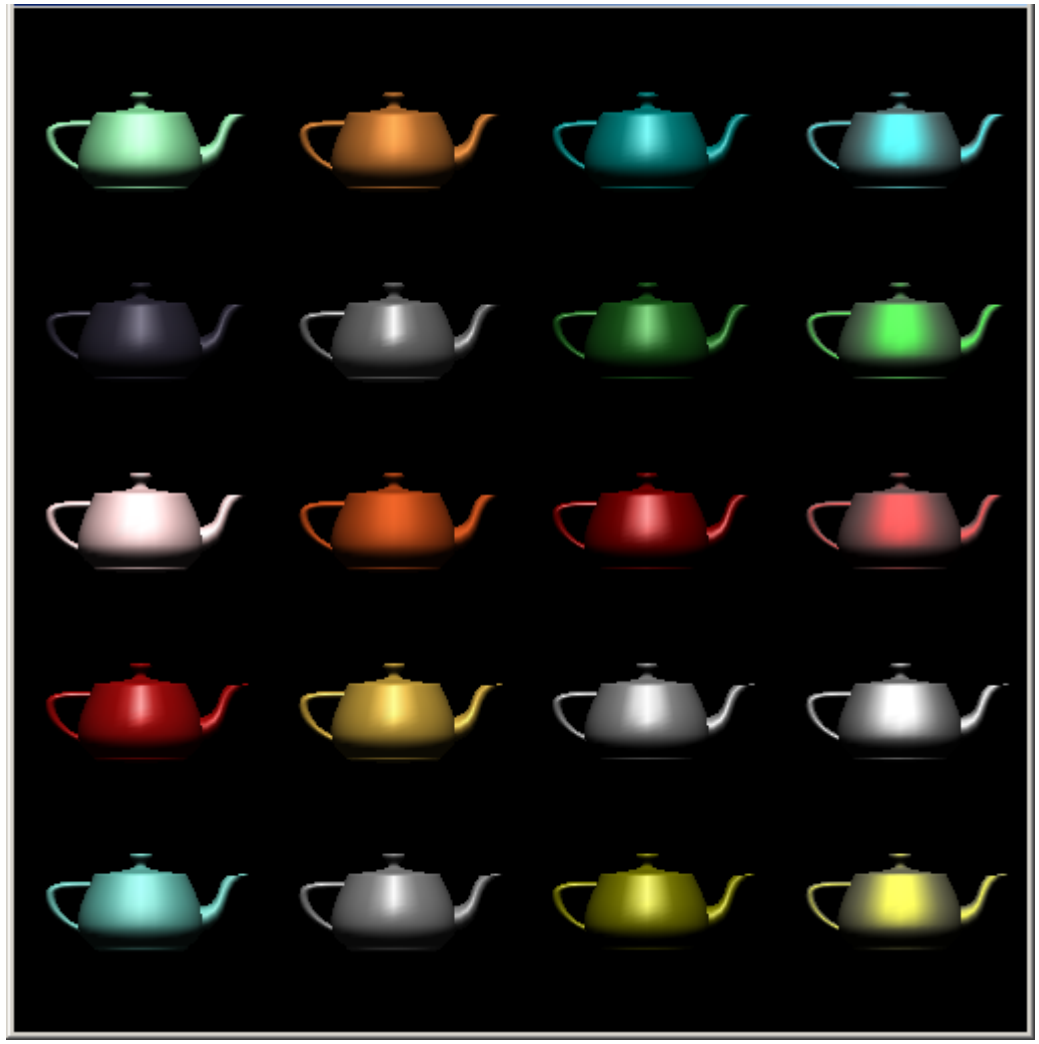
OpenGL Materials

- Material properties indicate how the surface reflects light.
- Material basically refers to the color of the surface.

```
struct material
{
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    float shininess;
};
```

Example

- Teapot materials



OpenGL Light Color

- The light sources have color and intensity, and the default is white.
- The object color we see is determined by the interaction of light color and material color.
  - Light color refers to how much RGB light shines on an object.
  - Material color refers to how much RGB light is reflected off an object.
  - White light * red material = red
  - Red light * white material = red
  - Red light * blue material = black
  - Yellow light * cyan material = green

OpenGL Ambient/Diffuse/Specular Light

- The OpenGL lighting model considers three types of reflected light.
- Ambient light: Light that is reflected off other surfaces. Brightens the overall scene.
- Diffuse light: Light that is reflected equally in all directions on the surface. It is independent of the viewer's position.
- Specular light: Light that is strongly reflected in one direction on the surface. It is used for shiny surfaces.

OpenGL Ambient Light

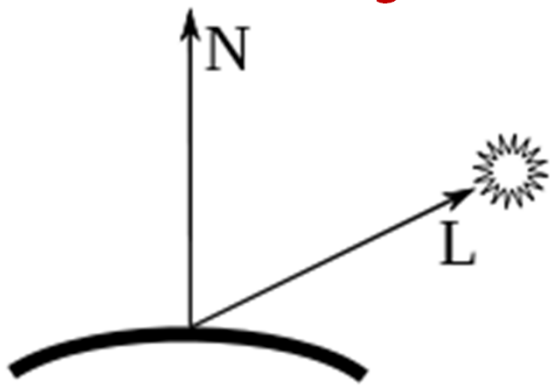
- Ambient light simulates indirect incoming light.
- It is assumed that light of constant brightness is spread evenly all over the place, regardless of the direction of the surface.
- The ambient light is specified by the ambient light color of the light source and the ambient light color of the material.

```
vec3 ambientLighting
    = vec3(scene_ambient) * vec3(mymaterial.ambient);
```

OpenGL Diffuse Light

- Diffuse light simulates light reflected equally in all directions on the surface.
- It is most common form and it is not dependent on the viewer's position.

```
vec3 diffuseReflection
    = attenuation * vec3(light0.diffuse) * vec3(mymaterial.diffuse)
    * max(0.0, dot(normalDirection, lightDirection));
```



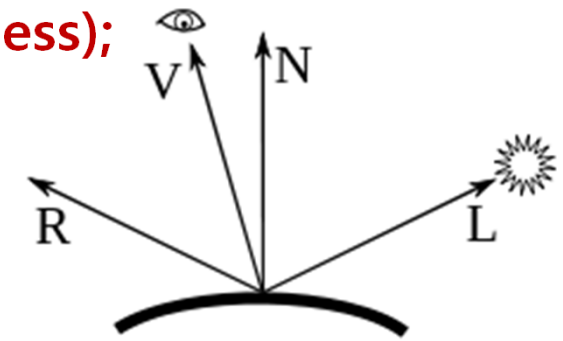
OpenGL Specular Light

- Specular light creates shiny highlights on the surface.
- Three properties of specular light:
  - Specular color of the material
  - Specular color of the light source
  - Shininess of the material – a high shininess creates a small area of highlight.

specularReflection

= attenuation * vec3(light0.specular) * vec3(mymaterial.specular)

*** pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)), mymaterial.shininess);**

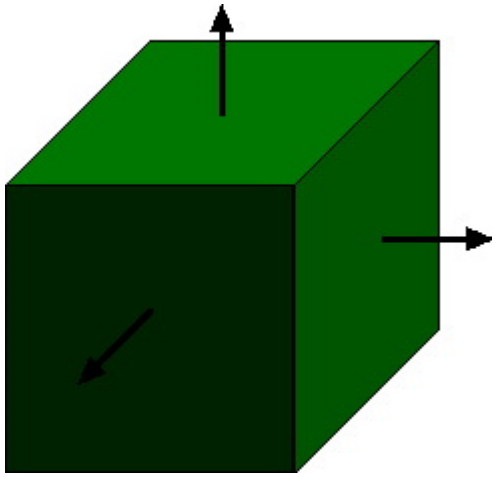


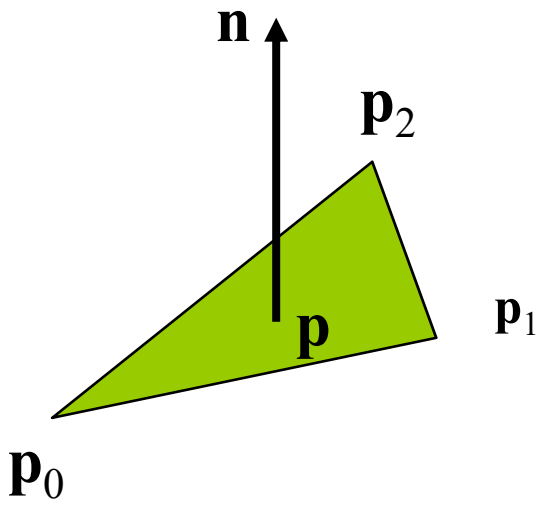
Surface Orientation

- To calculate how much light is reflected off a surface in lighting, we needs to know in which direction the surface is facing.
- The surface orientation and light direction determine the brightness of the diffuse light.
- The surface orientation, light direction, and direction to the viewer determine the brightness of the specular light.

Surface Normal

- The surface orientation is defined by a normal vector perpendicular to the surface.
- The plane's normal vector must be a unit vector (the length of the unit vector is 1).





$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$

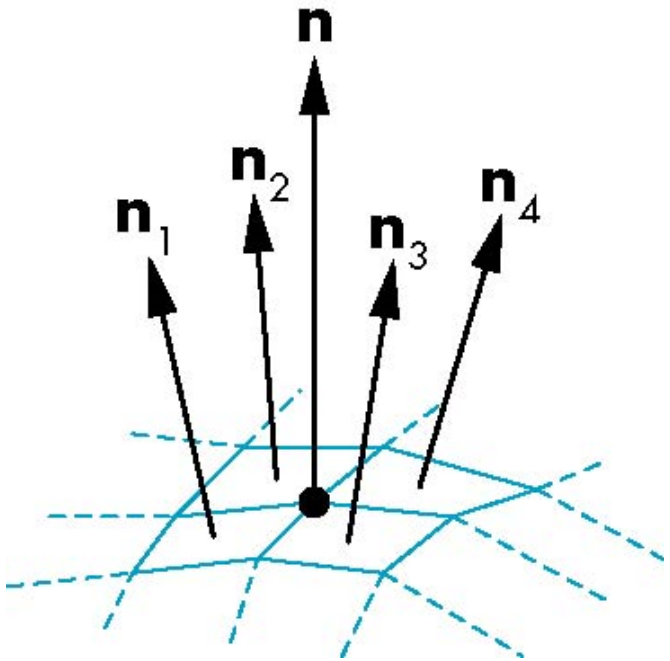
$$\mathbf{n} = (\mathbf{p}2 - \mathbf{p}0) \times (\mathbf{p}1 - \mathbf{p}0)$$

normalize $\mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|$

Vertex Normal

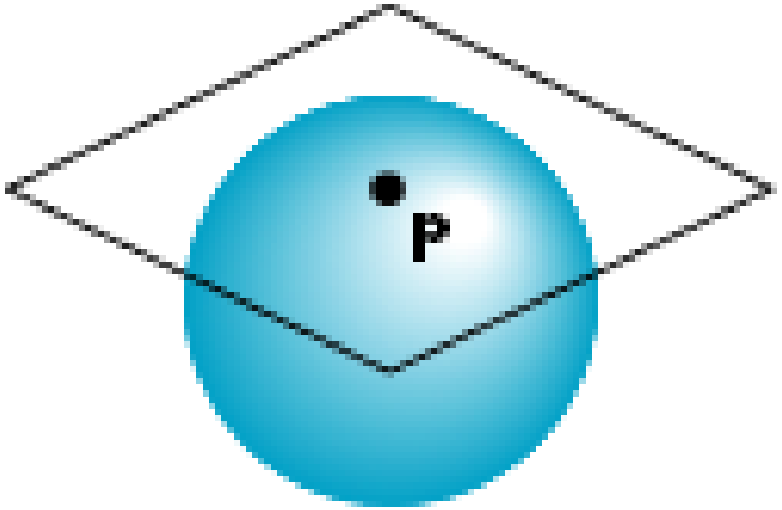
- Vertex normal
  - Average the value of the normal vectors of the adjacent faces that share the vertex.

$$n = \frac{n_1 + n_2 + n_3 + n_4}{\left| n_1 + n_2 + n_3 + n_4 \right|}$$



Normal to Sphere

- Implicit function of Sphere:
  - $f(x,y.z)=0$
- Unit Sphere:
  - $f(\mathbf{p})=\mathbf{p}\cdot\mathbf{p}-1=0$
- Sphere normal vector:
  - $n = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]^T = \mathbf{p}$



OpenGL Normal

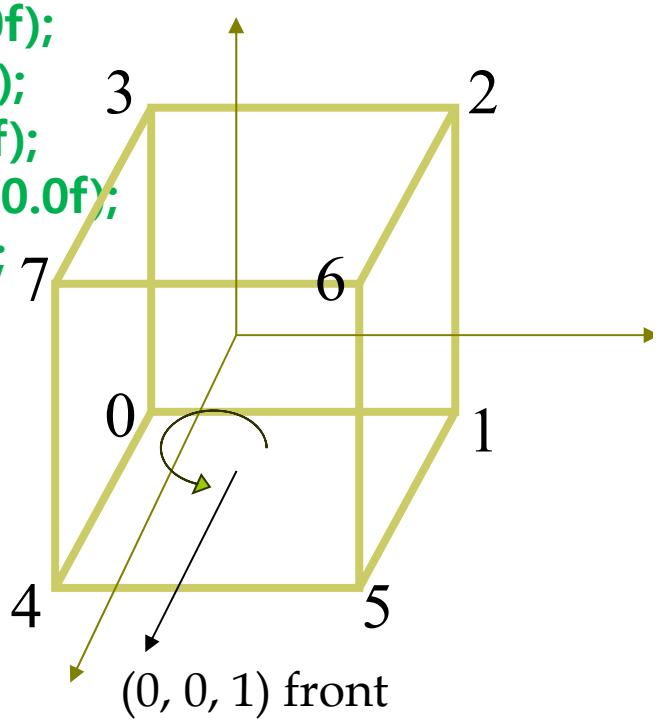
- OpenGL's lighting calculation uses the normal vector of each vertex.
- Example:

```
// Vertice Positions
squareVertices.push_back(glm::vec3(-1.5f, -1.5f, 0.0f));
squareVertices.push_back(glm::vec3( 1.5f, -1.5f, 0.0f));
squareVertices.push_back(glm::vec3( 1.5f,  1.5f, 0.0f));
squareVertices.push_back(glm::vec3(-1.5f,  1.5f, 0.0f));
// Vertices Normals
squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
```

OpenGL Normal

- To draw the Shaded Cube.

```
glm::vec3 frontNormal = glm::vec3(0.0f, 0.0f, 1.0f);
glm::vec3 backNormal = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 leftNormal = glm::vec3(-1.0f, 0.0f, 0.0f);
glm::vec3 rightNormal = glm::vec3(1.0f, 0.0f, 0.0f);
glm::vec3 bottomNormal = glm::vec3(0.0f, -1.0f, 0.0f);
glm::vec3 topNormal = glm::vec3(0.0f, 1.0f, 0.0f);
```



OpenGL Normal

```
// Front face
vbo.addData(&glm::vec3(-size, -size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
vbo.addData(&glm::vec3( size, -size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
vbo.addData(&glm::vec3( size,  size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
vbo.addData(&glm::vec3(-size, -size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
vbo.addData(&glm::vec3( size,  size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
vbo.addData(&glm::vec3(-size,  size,  size), sizeof(glm::vec3));
vbo.addData(&frontNormal[0], sizeof(glm::vec3)); // vertex normal
```