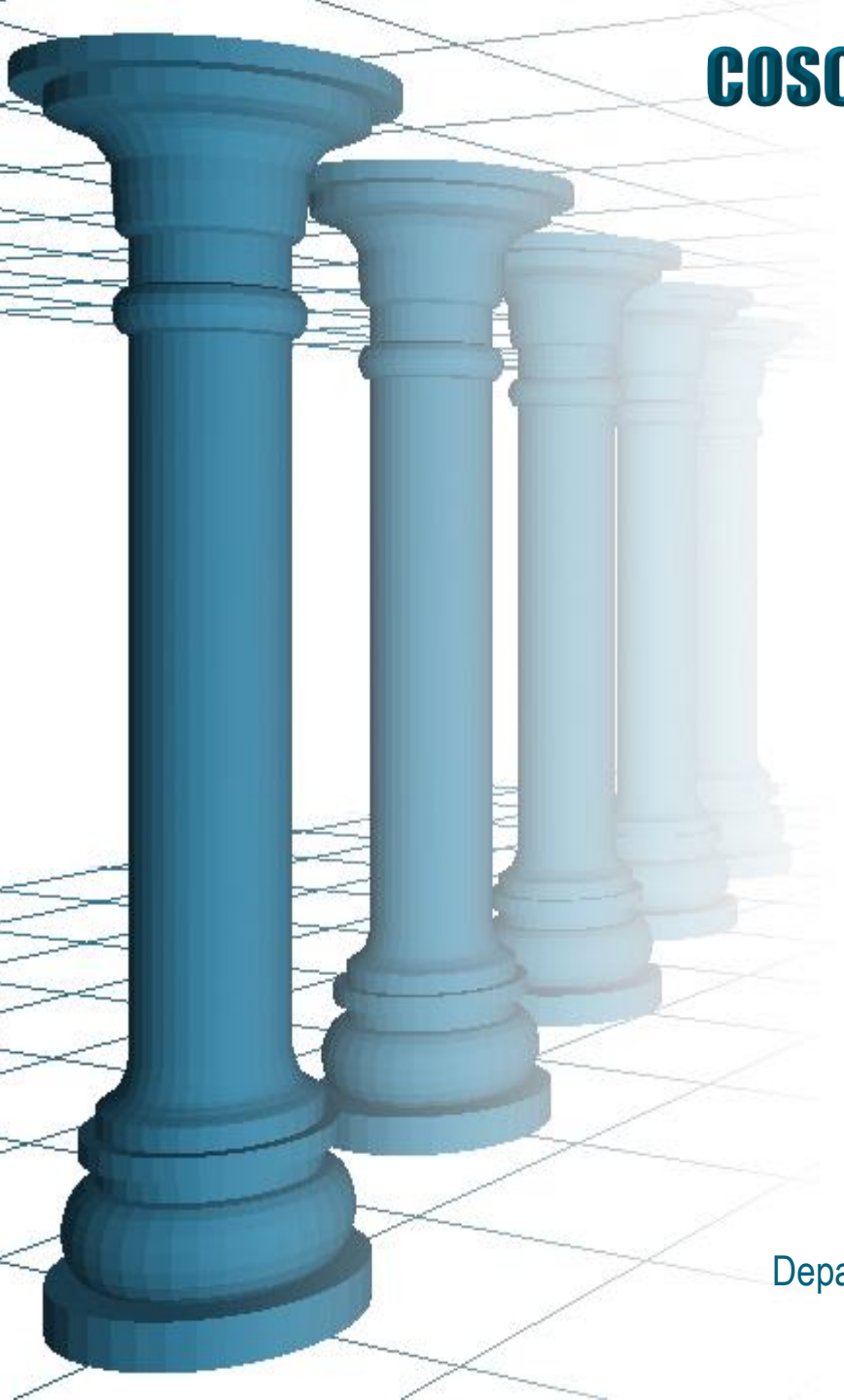


COSC363 Computer Graphics

3

Illumination

Let there be light ...

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Light

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

In "initialize()" function

In "display()" function

(x, y, z)

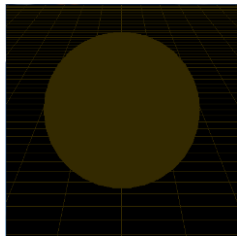
```
float light_pos[] = {20, 20, 0, 1}
glLightfv(GL_LIGHT0, GL_POSITION,  light_pos);
```

- Light's position is an imaginary point that can be transformed using transformation matrices including the view matrix.
- No 3D primitive created.
- Light's position is used for constructing a few vectors for the computation of color values.

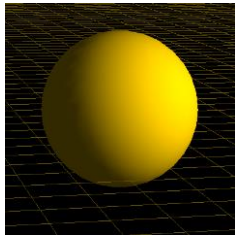
Three Types of Reflections

There can be three types of reflections from a surface:

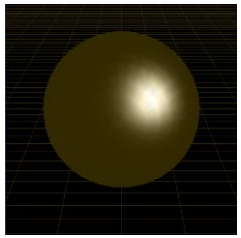
- **Ambient reflection**: This is caused by the ambient light. Ambient light (a.k.a background light) is the base level of constant brightness for a scene.



- **Diffuse reflection**: The most common form of reflection where the intensity varies according to the angle between the light's direction and the surface normal vector.

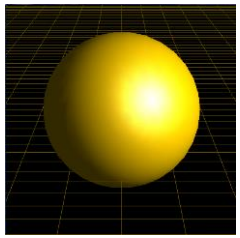


- **Specular reflection**: This is a mirror-like reflection of high intensity along a narrow cone around the direction of reflection. This reflection is view-dependent, unlike the other two.



- In general, the above three components are added together to get the net reflection.

Ambient+Diffuse+Specular=



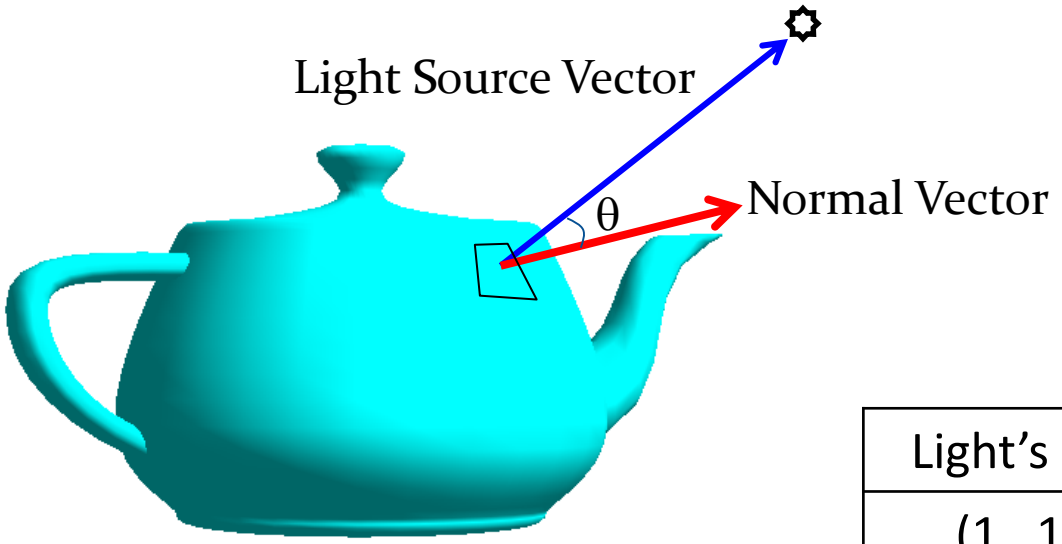
Ambient Reflection

- Ambient light is constant everywhere in the scene.
  - It does not depend on light's position, viewer's position or surface orientation.
- Ambient light is typically defined as a low intensity gray value. Example: (0.2, 0.2, 0.2)
- Ambient light interacts with the material colour to provide a **uniform dark shade** of the material colour

Light's Color	Material's Color
(0.2, 0.2, 0.2)	(0, 1, 1)

Diffuse Reflection

The intensity of reflection depends on the orientation of the surface relative to light's direction. It reduces when the angle θ between the light source vector and the surface normal vector increases. Diffuse reflection becomes maximum when $\theta=0$, and minimum (zero) when $\theta \geq 90°$.



Light's Color	Material's Color
(1, 1, 1)	(0, 1, 1)

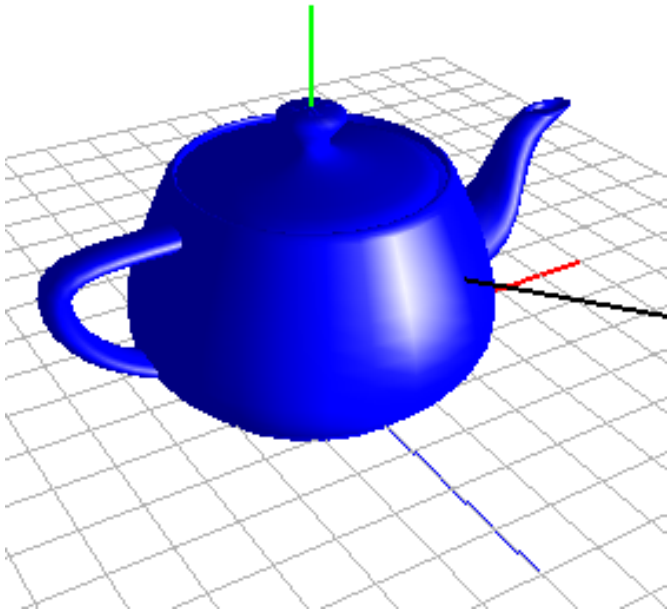
Specular Reflection

- Specular highlights are bright reflections from mirror-like or polished surfaces.
- The reflection is directional and **view dependent**.
- The material colour for specular reflection is usually set as white, to provide a bright highlight.

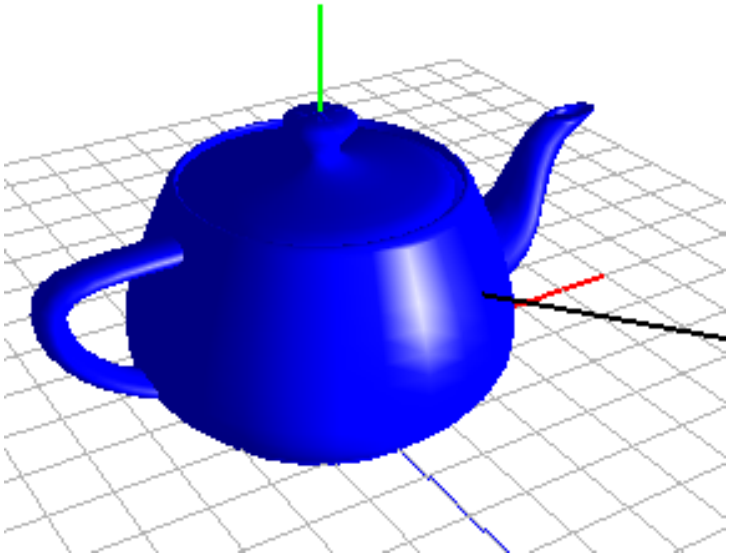
Light's Color	Material's Color
(1, 1, 1)	(1, 1, 1)

“Shininess” of Specular Reflection

- The term “shininess” (GL_SHININESS) refers to the width (or spread) of the specular highlight.
- Increasing the value of the shininess term reduces the spread of the highlight, making it more concentrated



Shininess = 40



Shininess = 100

Ambient, Diffuse and Specular Components

Both light and material have 3 components:

- Ambient color, diffuse color and specular color.

```
float white[3] = {1, 1, 1};
float black[3] = {0, 0, 0};
float gray[3] = {0.2, 0.2, 0.2};
float cyan[3] = {0, 1, 1};

glLightfv(GL_LIGHT0, GL_AMBIENT, gray);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, cyan);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, cyan);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, white);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100);
```

Note: Material's ambient and diffuse colours are the same.
To disable specular highlights from a surface, set its specular material colour to 0.

Material Color

- When lighting is enabled, OpenGL ignores the colour values specified using functions glColor3f(...), glColor4f(...) etc, and uses material colour specified using glMaterialfv(..)
- Defining material properties using glMaterialfv(...) for each object in a scene may lead to cumbersome code.
- We can force OpenGL to use the colour value defined using glColor3f(...) for the current material's ambient and diffuse properties. This is done using the following functions:

In `initialize()`

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
```

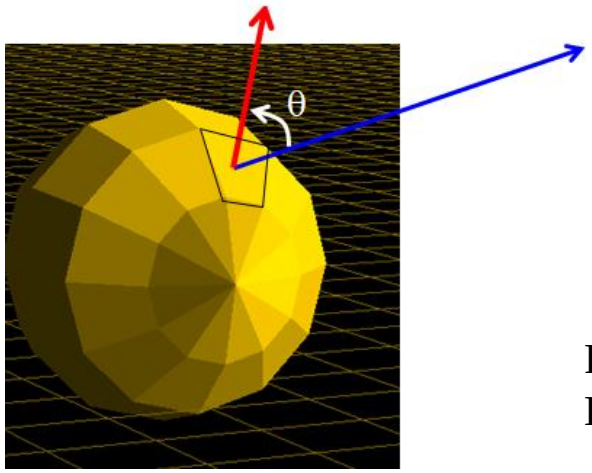
In `display()`

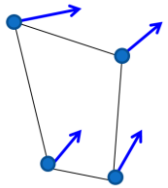
```
glColor3f (0, 1, 0);
glutSolidTeapot(1);
```

Does not require separate array initializations

Per-Vertex Lighting

- OpenGL performs lighting calculations only at the vertices of polygonal elements.
- The interior of the polygon is then filled using interpolation
- Lighting calculations require the normal vector at each vertex. This vector must be supplied by the user.





Light source

Polygonal Element

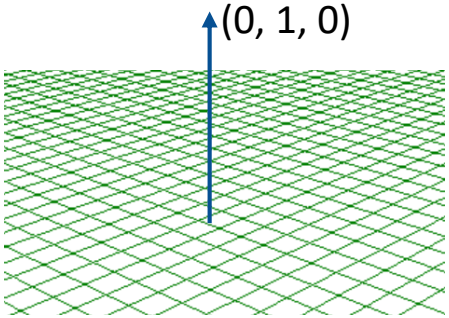
```
glEnable(GL_NORMALIZE);

glBegin(GL_TRIANGLES);
  glNormal3f(nx1, ny1, nz1);
  glVertex3f(x1, y1, z1);
  glNormal3f(nx2, ny2, nz2);
  glVertex3f(x2, y2, z2);
  glNormal3f(nx3, ny3, nz3);
  glVertex3f(x3, y3, z3);
  ...
  ...
glEnd();
```

Per-vertex normal definition

Surface Normal Vectors

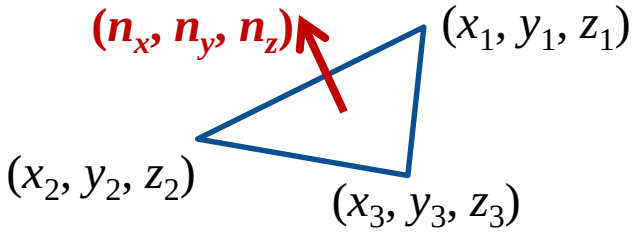
- The code on the previous slide specifies the normal vector for each vertex. For objects such as a sphere, we can easily compute per-vertex normals.
- Objects like the floor plane need to have only one normal definition for the entire object.




```
glEnable(GL_NORMALIZE);

glBegin(GL_TRIANGLES);
  glNormal3f(nx, ny, nz);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  ...
glEnd();
```

- Surface normals may also be defined on a per-face basis (see next slide)

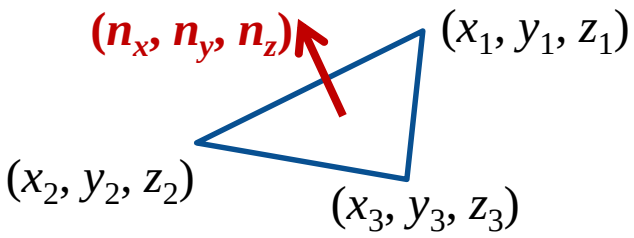


Per-face normal definition

```
glEnable(GL_NORMALIZE);

glBegin(GL_TRIANGLES);
  normal(..);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
glEnd();
```

Surface Normal Computation



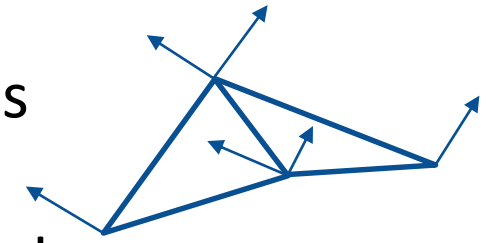
See Model3D.cpp

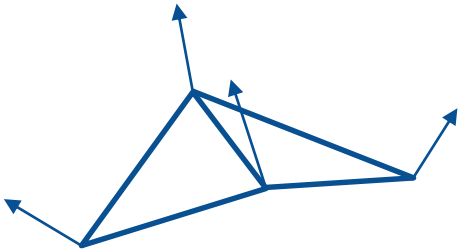
```
void normal(int index)
{
    x1 = ...
    y1 = ...
    z1 = ...
    ...
    ...
    nx = y1*(z2-z3) + y2*(z3-z1) + y3*(z1-z2);
    ny = z1*(x2-x3) + z2*(x3-x1) + z3*(x1-x2);
    nz = x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2);
    glNormal3f(nx, ny, nz);
}
```

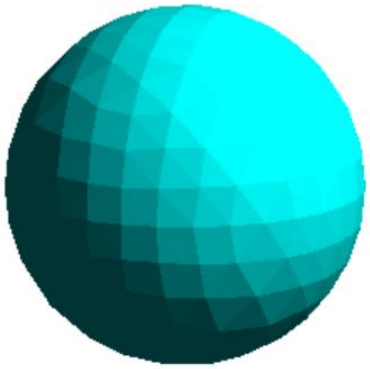
Computation of face normals

Face normal vs Vertex normal

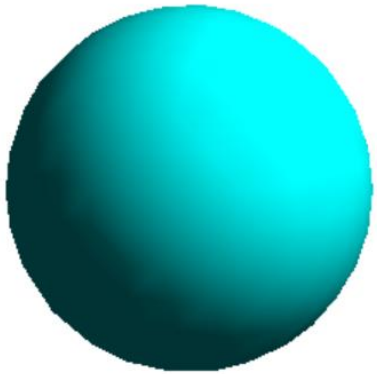
- In a face-based normal definition, the same normal vector is assigned to all three vertices of a triangle. This gives a nearly constant colour for each face and reveals the polygonal structure of the object.
- Per-vertex normal vectors generate a smooth variation of shades across a surface.







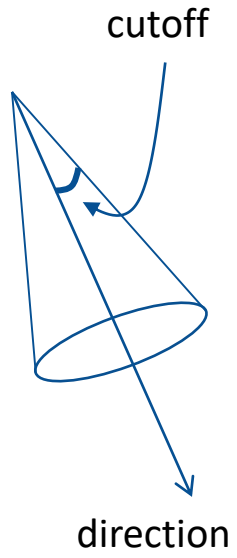
Using face normals



Using vertex normals

Spot Lights

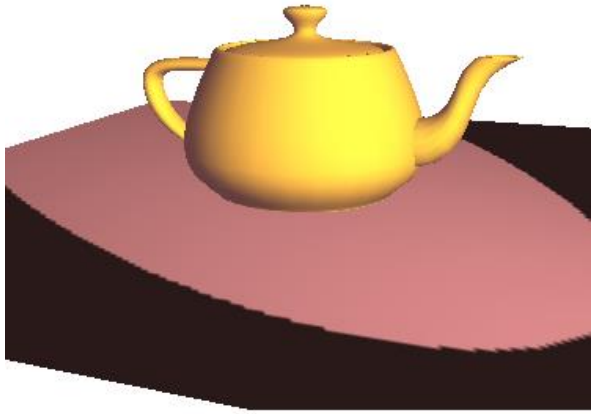
- By default, all OpenGL lights are omni-directional lights. They behave as if they "emit" light in all directions.
- A light can be converted to a spot light by specifying
  - A spot cutoff angle. This is the half cone angle of the spotlight.
  - A spot direction. This is a vector specifying the cone's axis.
  - A spot exponent. This specifies how fast the intensity drops off as a vertex is moved from the centre of the spotlight towards its edge.




```
float spotdir[]={5.0, -2.0, -4.0};
glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 10.0);
glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 2.0);
glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, spotdir);
```

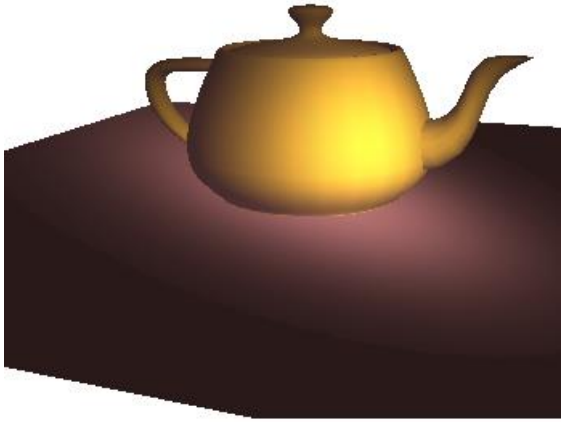
Spot Lights

Cutoff = 8 degs.
Exponent = 2



Cutoff = 15 degs.
Exponent = 2

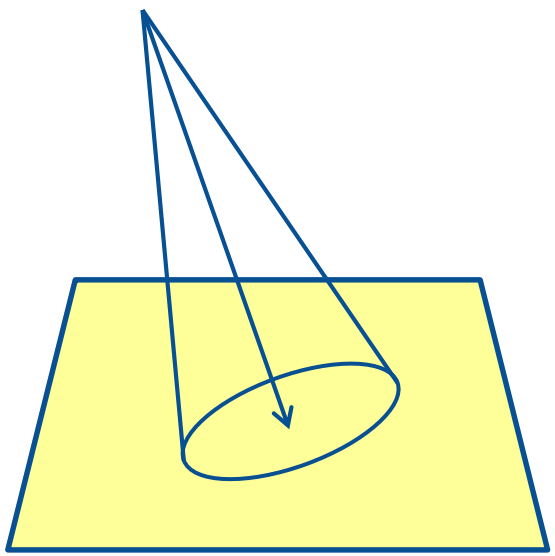
Cutoff = 15 degs.
Exponent = 50

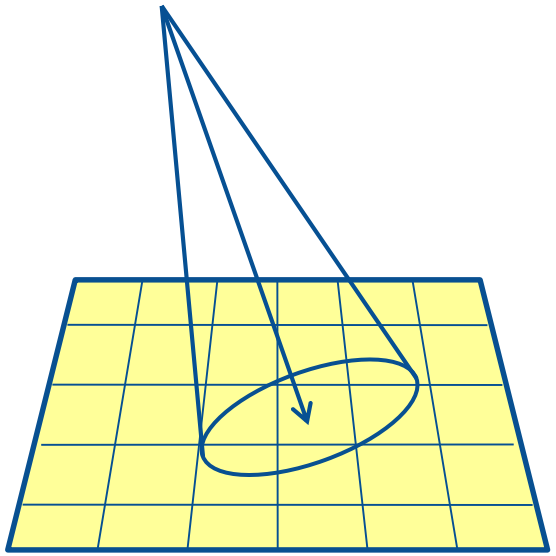


Cutoff = 15 degs.
Exponent = 100

OpenGL Lighting

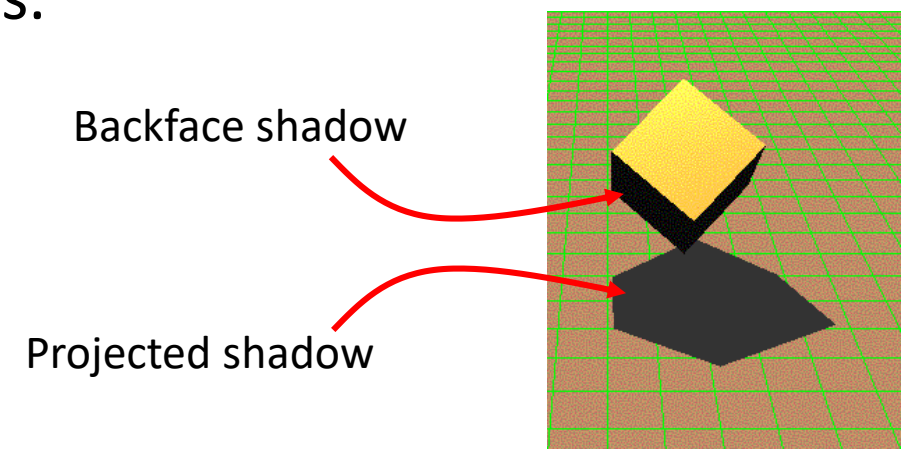
A spot light in the middle of a large quad (eg. floor plane) will not be visible unless the plane is subdivided into smaller quads. This is because lighting calculations are done only at the vertices of every polygon.





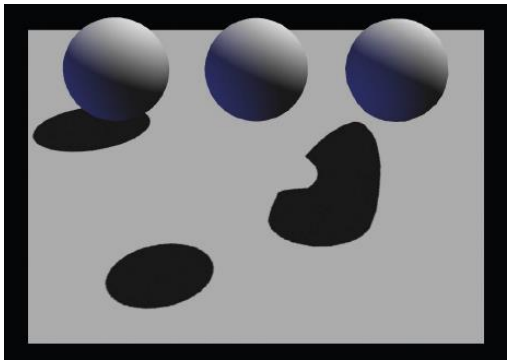
Shadows

- Two types of shadows:
  - Backface shadows: A shadow on an object's surface that is oriented away from light. This type of shadows are automatically generated by the illumination model ($\theta \geq 90°$; see slide 4)
  - Projected shadows or cast shadows: Shadows cast by a part of an object's surface on either the same or a different object.
- OpenGL's lighting model cannot generate projected shadows.



Projected Shadows

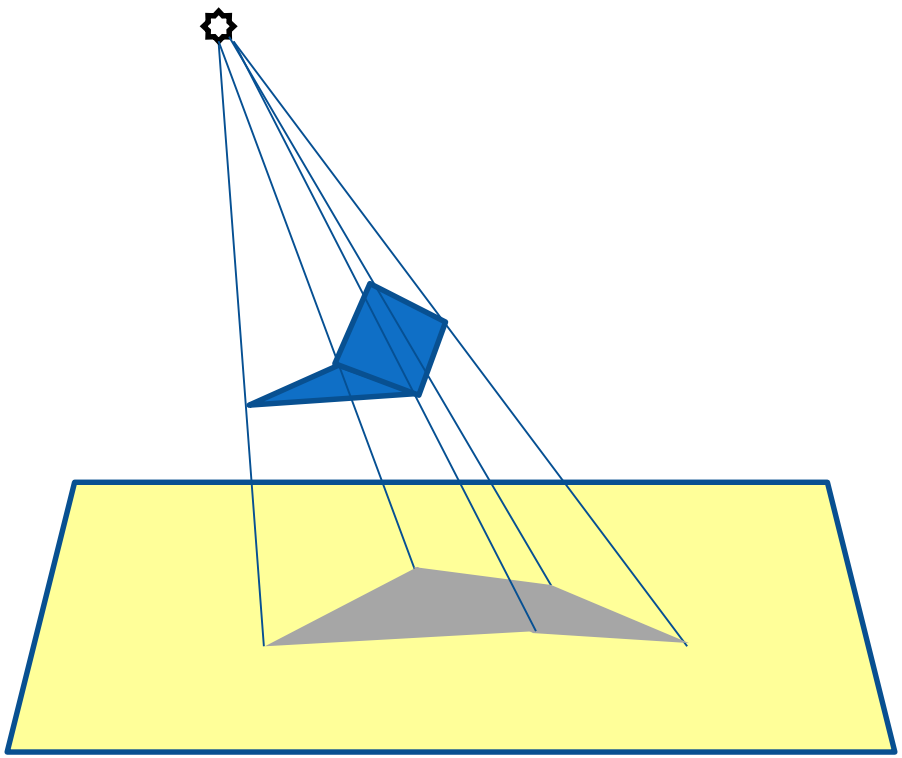
- Shadows add a great amount of realism to a scene.
- Shadows provide a second view of an object.
- Shadows convey additional information such as depth cues (eg. object's height from a floor plane) and object's shape.



Planar Shadows on Floor Plane (y=0)

Two-pass rendering method:

- Project all vertices of an object to the floor plane.
- Temporarily disable lighting
- Render the projected object using shadow colour
- Enable lighting
- Draw the object



Planar Shadows on Floor Plane (y=0)

To project vertices to the floor plane:

Let the light's position be given by (gx, gy, gz)

Create a 16 element array as follows. This array represents the transformation matrix that projects vertices to the floor plane.

```
float shadowMat[16] =
{ gy,0,0,0, -gx,0,-gz,-1,  0,0,gy,0,  0,0,0,gy };
```

Apply this transformation to the object using

```
glMultMatrixf(shadowMat);
```

Planar Shadows: Code

```
float shadowMat[16] = { gy,0,0,0, -gx,0,-gz,-1,
                        0,0,gy,0,  0,0,0,gy };
 glDisable(GL_LIGHTING);
 glPushMatrix();     //Draw Shadow Object
   glMultMatrixf(shadowMat);
   /* Object Transformations */
   glColor4f(0.2, 0.2, 0.2, 1.0);
   drawObject();
 glPopMatrix();

 glEnable(GL_LIGHTING);
 glPushMatrix();     //Draw Actual Object
    /* Transformations */
    drawObject();
 glPopMatrix();
```