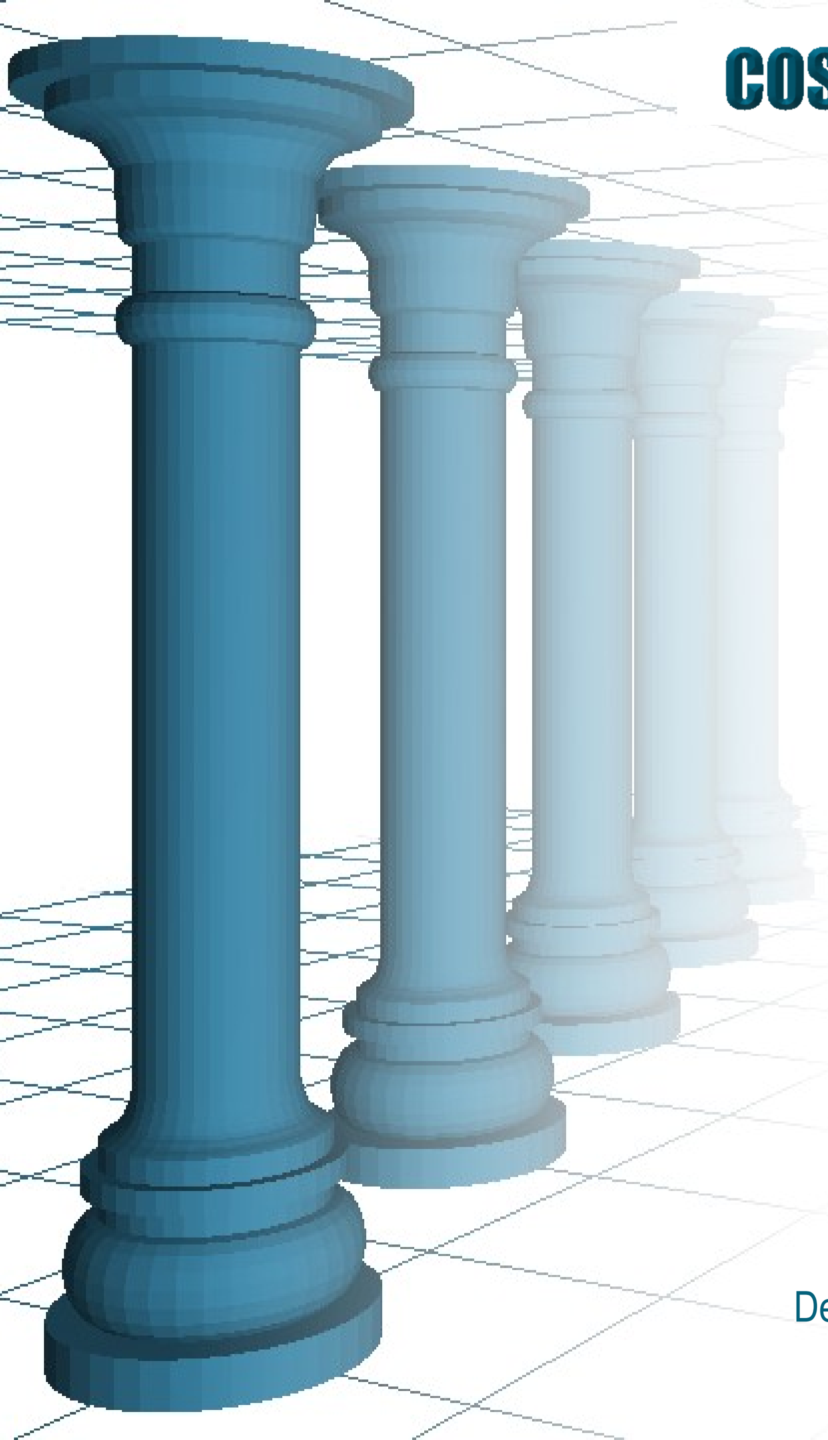


COSC363 Computer Graphics

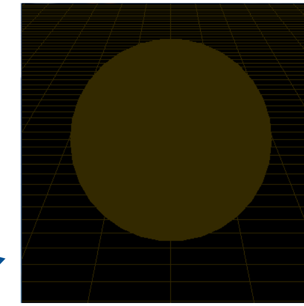
3
Illumination

Let there be light ...

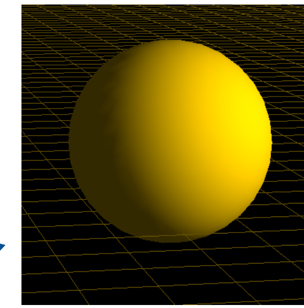
R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Three Types of Reflections

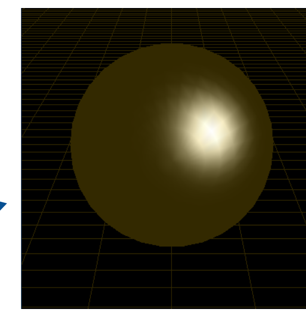
In general, there can be three types of reflections from a surface:



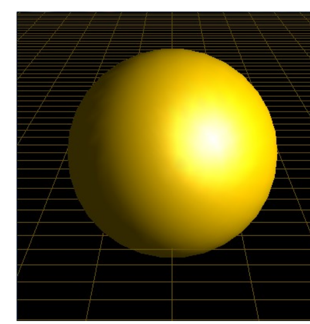
- **Ambient reflection**: This is caused by the ambient light. Ambient light (a.k.a background light) is the base level of constant brightness for a scene.



- **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.



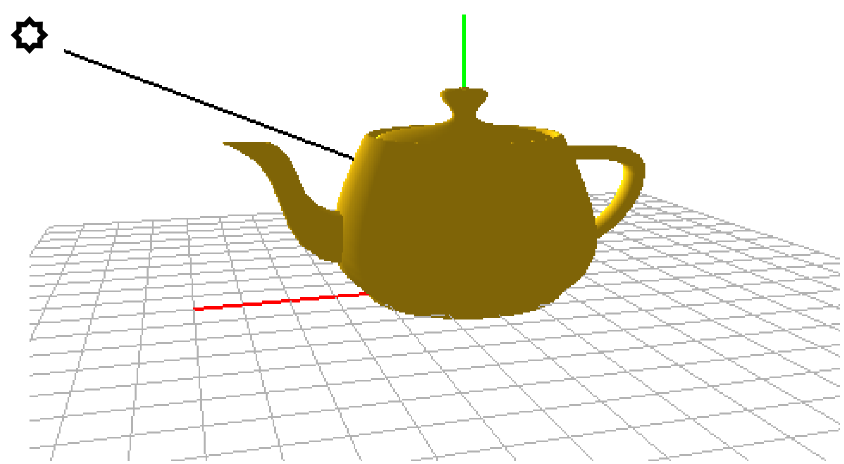
- **Specular reflection**: This is a mirror-like reflection of high intensity along a narrow cone around the direction of reflection. This reflection is view-dependent, unlike the other two.



Ambient+Diffuse+Specular

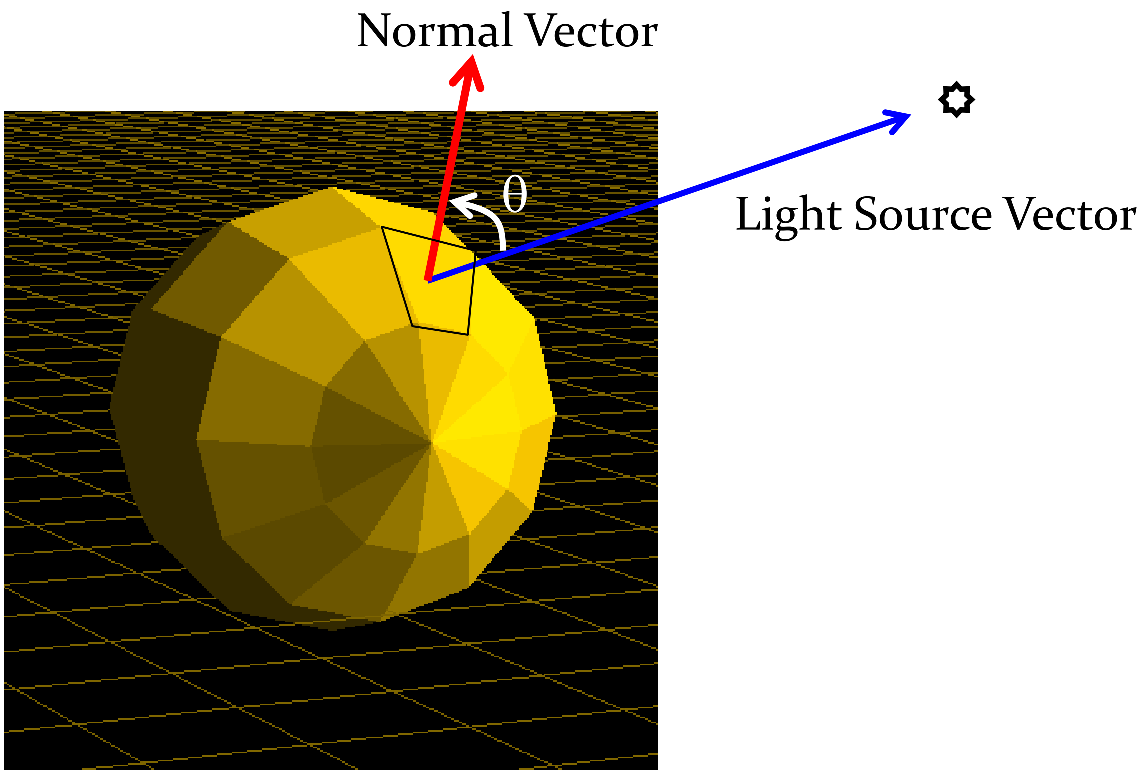
Ambient Reflection

- 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, 1)
- Ambient light interacts with the material colour to provide a **uniform dark shade** of the material colour in shadow regions.



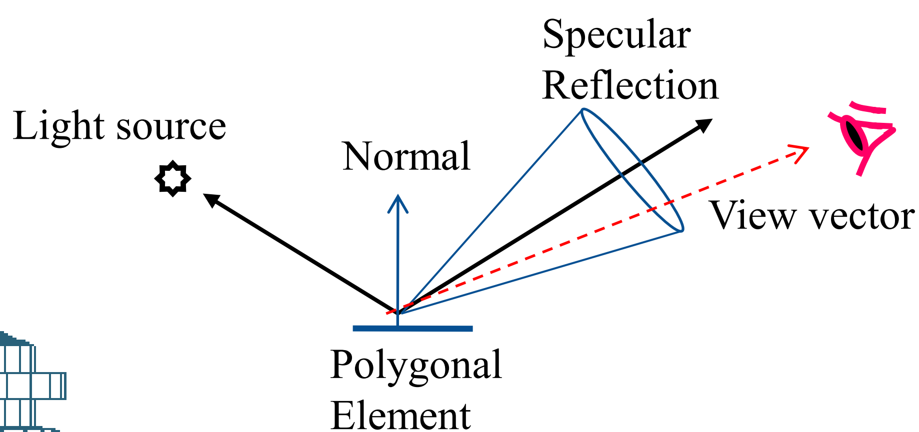
Diffuse Reflection

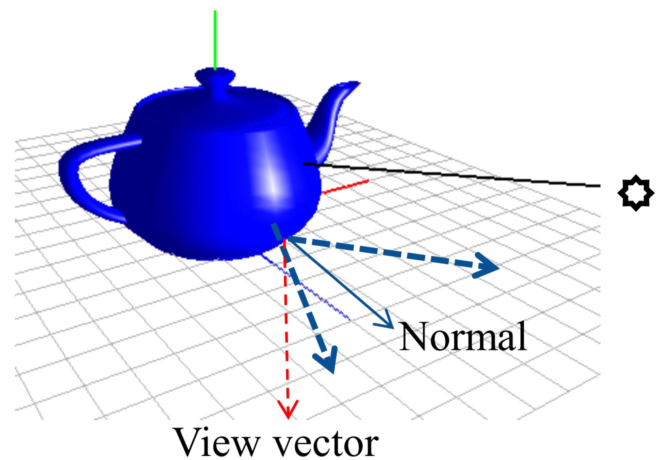
The intensity of reflection depends on the orientation of the surface relative to light's direction. It reduces when the angle θ between the light source vector and the surface normal vector increases. Diffuse reflection becomes (i) maximum when $\theta=0$, (ii) minimum (zero) when $\theta \geq 90°$.



Specular Reflection

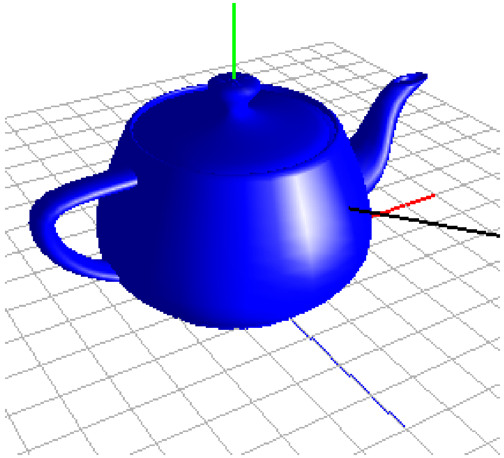
- Specular highlights are bright reflections from mirror-like or polished surfaces.
- The reflection is directional and **view dependent**.
- The material colour for specular reflection is usually set as white, to provide a bright highlight.



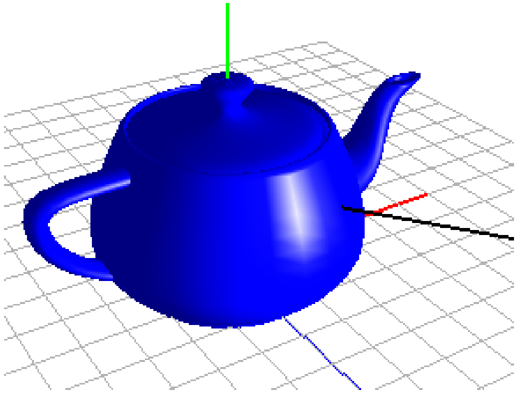


"Shininess" of Specular Reflection

- 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 around a vertex.



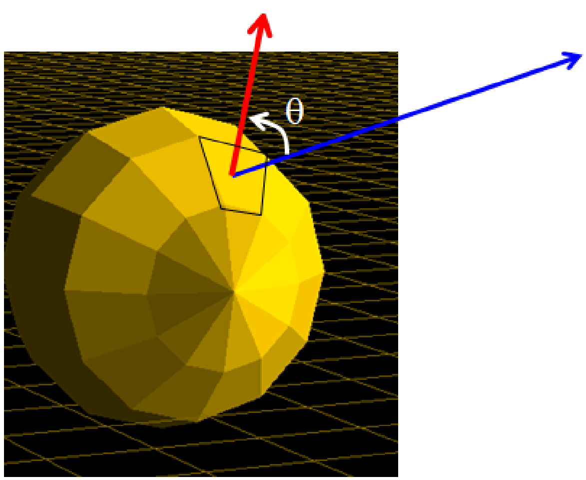
Shininess = 40

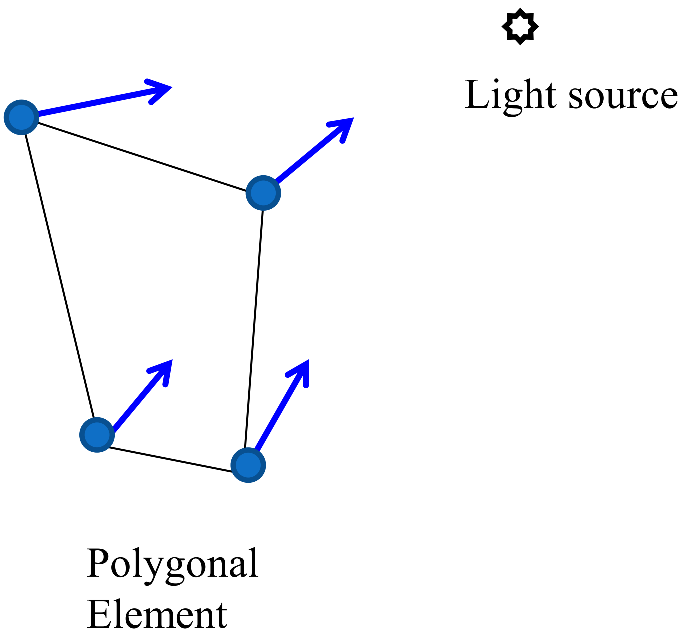


Shininess = 100

Per-Vertex Lighting

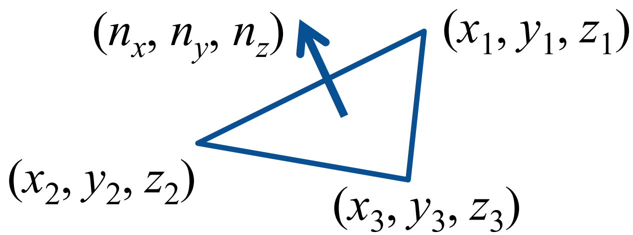
- OpenGL performs lighting calculations only at the vertices of polygonal elements.
- The interior of the polygon is then filled using interpolation
- OpenGL can compute light source vector, reflection vector, view vector **at each vertex**, but not the normal vector.





Surface Normal

The user must compute the surface normal components for each primitive, and specify the values using the glNormal3f() function.



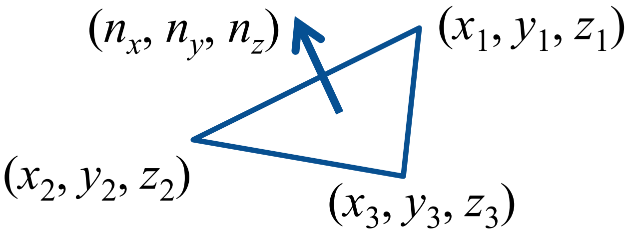

```
glEnable(GL_NORMALIZE);

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

Pre-defined normal direction (nx, ny, nz). Eg. Floor plane.

```
glEnable(GL_NORMALIZE);

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

Normal direction computed using a user-defined function. See next slide.

Surface Normal Computation




```
void computeNormal(float x1, float y1, float z1,
                   float x2, float y2, float z2,
                   float x3, float y3, float z3)
{
    float nx, ny, nz;
    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);
}
```

Ambient, Diffuse and Specular Components

Both light and material have 3 components: ambient, diffuse and specular. Each component is a colour value.

	Light	Material
Ambient:	L_a A low-intensity gray value Eg. (0.2, 0.2, 0.2, 1.0)	M_a Material Colour Eg. Yellow (1.0, 1.0, 0.0, 1.0)
Diffuse:	L_d White (1.0, 1.0, 1.0, 1.0) Assign a different value for coloured light	M_d Material Colour Eg. Yellow (1.0, 1.0, 0.0, 1.0)
Specular:	L_s White (1.0, 1.0, 1.0, 1.0)	M_s White (1.0, 1.0, 1.0, 1.0)
Shininess:		f Eg. 100

Note: Material's ambient and diffuse colours are the same.

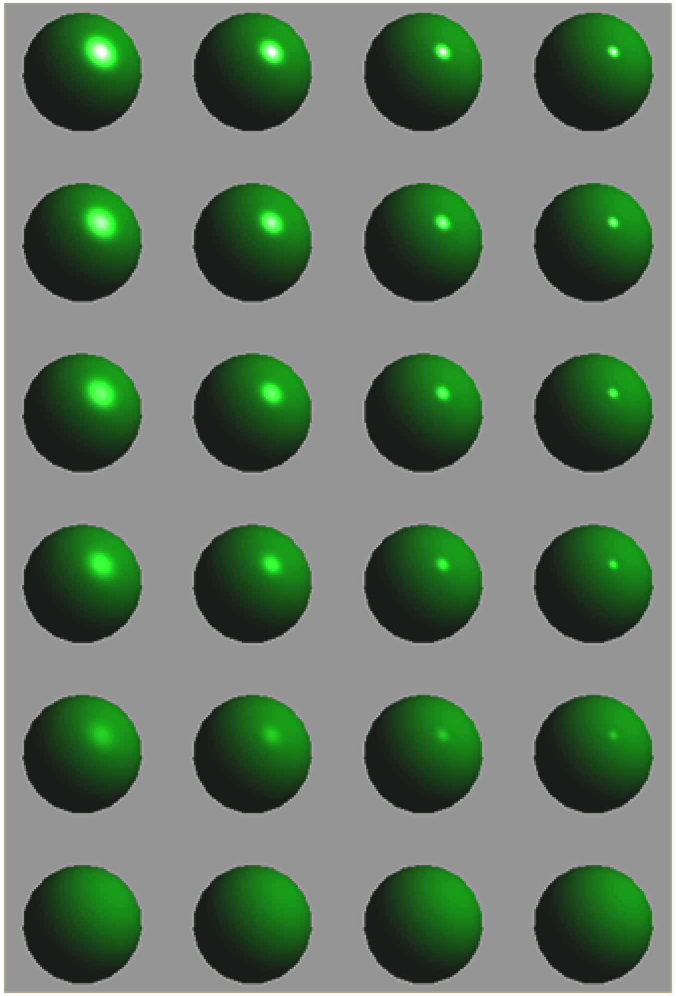
To disable specular highlights from a surface, set its specular material colour to 0.

Specular Reflection: Example

$M_s = (1,1,1)$

Material's Specular Colour

$M_s = (0,0,0)$



f = 10 20 50 100

Shininess

$L_a = (0.2, 0.2, 0.2)$
$L_d = (1.0, 1.0, 1.0)$
$L_s = (1.0, 1.0, 1.0)$

$M_a = (0.0, 1.0, 0.0)$
$M_d = (0.0, 1.0, 0.0)$

OpenGL Lighting: Example

```
void initialize()
{
  float ambient[4] = {0.2, 0.2, 0.2, 1.0};
  float white[4]   = {1.0, 1.0, 1.0, 1.0};

  float mat_col[4] = {1.0, 1.0, 0.0, 1.0};     //Yellow

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
  glLightfv(GL_LIGHT0, GL_SPECULAR, white);

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_col);
  glMaterialfv(GL_FRONT, GL_SPECULAR, white);
  glMaterialf(GL_FRONT, GL_SHININESS, 50);
}
```

L_a L_d

L_s

M_a, M_d

M_s

f

OpenGL Lighting: Example

```
void display()
{
  float lgt_pos[4]={0., 10., 10., 1.};
  glClear(GL_COLOR_BUFFER_BIT |  GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(5., 3., 2., 0., 0., 0., 0., 1., 0.);

  glLightfv(GL_LIGHT0, GL_POSITION, lgt_pos);

  glPushMatrix();
    glTranslatef(0.0, 1.2, 0.0);
    glRotatef(angle, 0.0, 1.0, 0.0);
    glutSolidTeapot(1.0);
  glPopMatrix();

  glFlush();
}
```

Specifying Material Properties

- 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()`

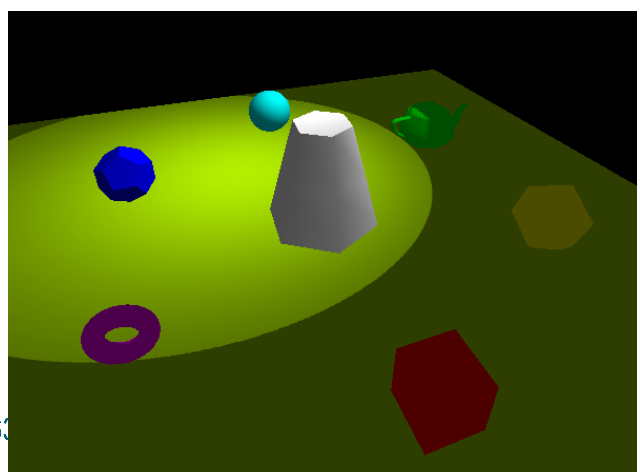
```
glColor3f (0, 1, 0);
glutSolidTeapot(1);
```

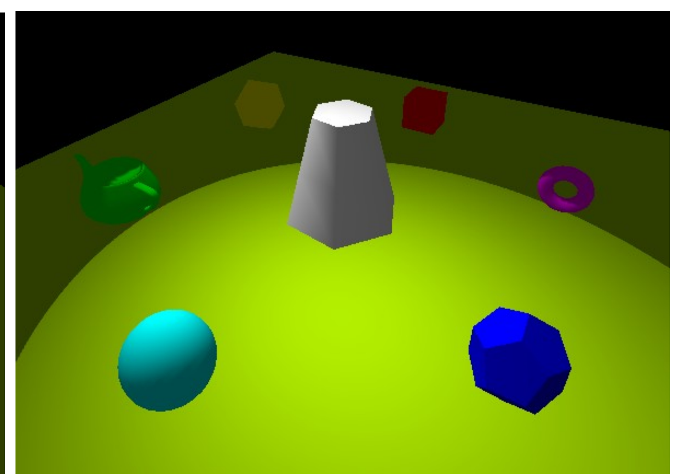
Does not require separate array initializations

Placement of Light Sources

Light source fixed in the scene

```
void display()
{
  float lgt_pos[4]={0., 10., 10., 1.};
  glClear(GL_COLOR_BUFFER_BIT |  GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(5., 3., 2., 0., 0., 0., 0., 1., 0.);
  glLightfv(GL_LIGHT0, GL_POSITION, lgt_pos);

  glTranslatef(0.0, 1.2, 0.0);
  glRotatef(angle, 0.0, 1.0, 0.0);
  glutSolidTeapot(1.0);

  glFlush();
}
```

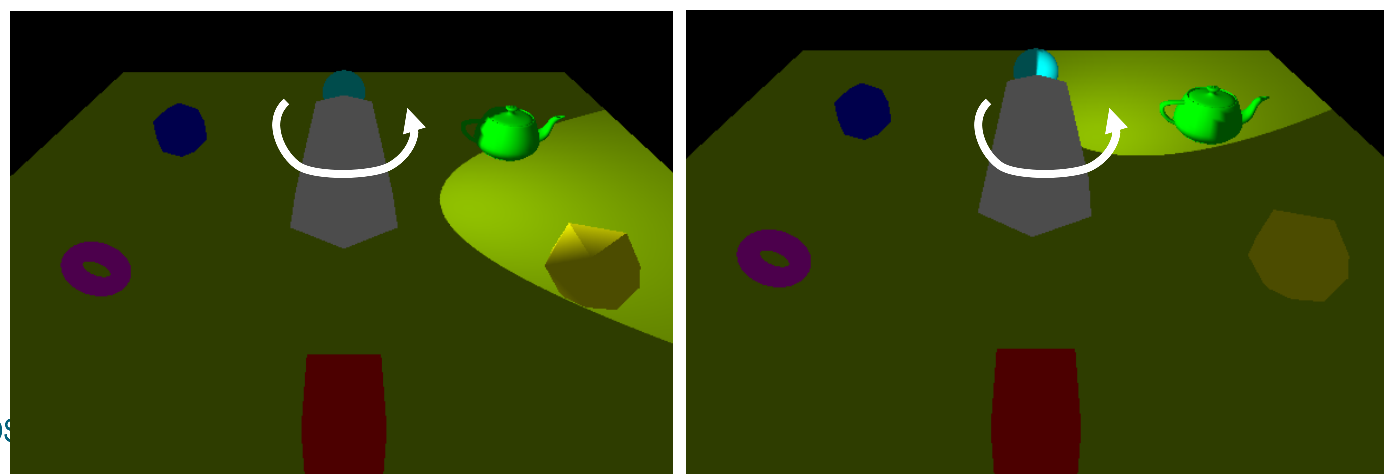




Placement of Light Sources

Light source fixed on an object

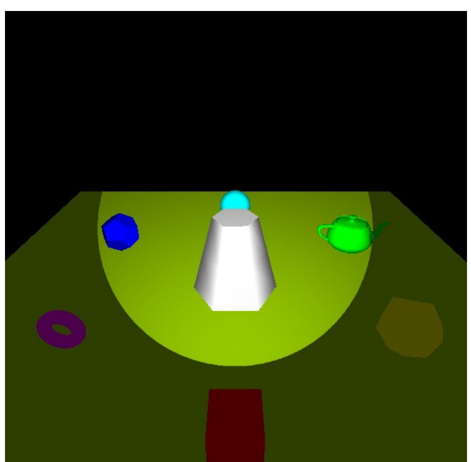
```
void display()
{
  float lgt_pos[4]={0., 10., 10., 1.};
  glClear(GL_COLOR_BUFFER_BIT |  GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(5., 3., 2., 0., 0., 0., 0., 1., 0.);

  glTranslatef(0.0, 1.2, 0.0);
  glRotatef(angle, 0.0, 1.0, 0.0);
  glLightfv(GL_LIGHT0, GL_POSITION, lgt_pos);
  glutSolidTeapot(1.0);

  glFlush();
}
```



Placement of Light Sources

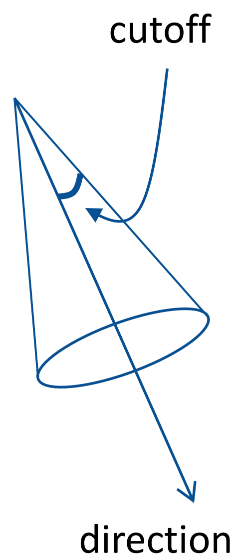
Light source fixed on the camera

```
void display()
{
  float lgt_pos[4]={0., 10., 10., 1.};
  glClear(GL_COLOR_BUFFER_BIT |  GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glLightfv(GL_LIGHT0, GL_POSITION, lgt_pos);
  gluLookAt(5., 3., 2., 0., 0., 0., 0., 1., 0.);

  glTranslatef(0.0, 1.2, 0.0);
  glRotatef(angle, 0.0, 1.0, 0.0);
  glutSolidTeapot(1.0);

  glFlush();
}
```



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

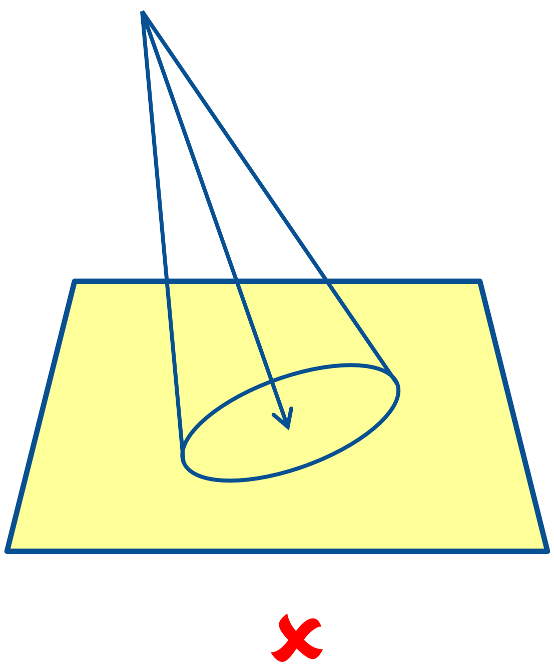
Cutoff = 15 degs.
Exponent = 2

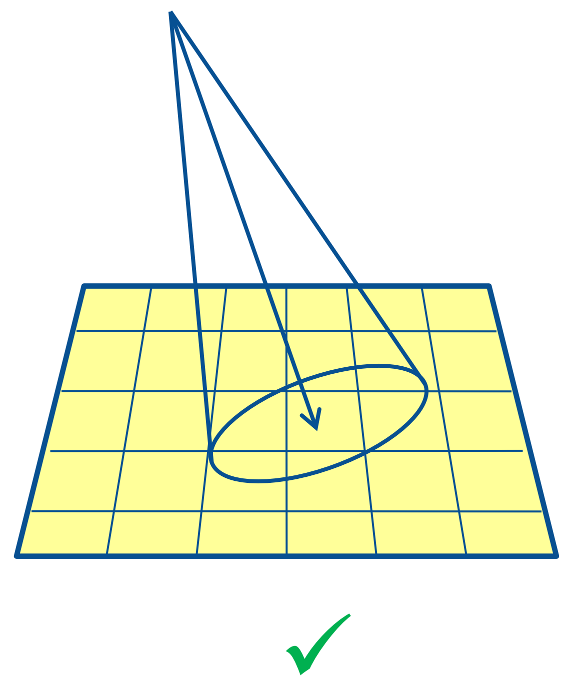
Cutoff = 15 degs.
Exponent = 50

Cutoff = 15 degs.
Exponent = 100

OpenGL Lighting

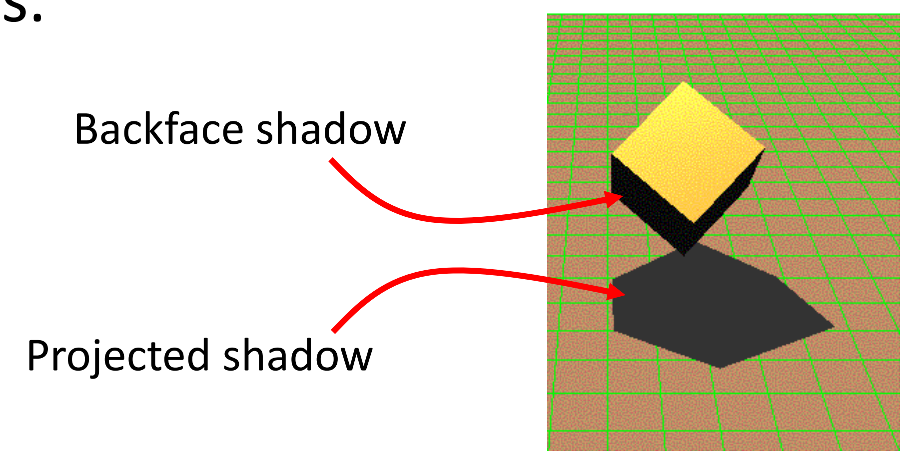
A spot light in the middle of a large quad (eg. floor plane) will not be visible unless the plane is subdivided into smaller quads. This is because lighting calculations are done only at the vertices of every polygon.





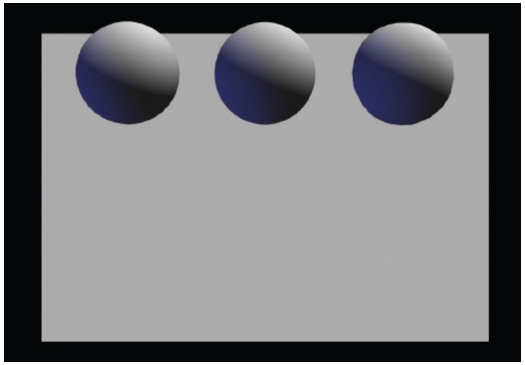
Shadows

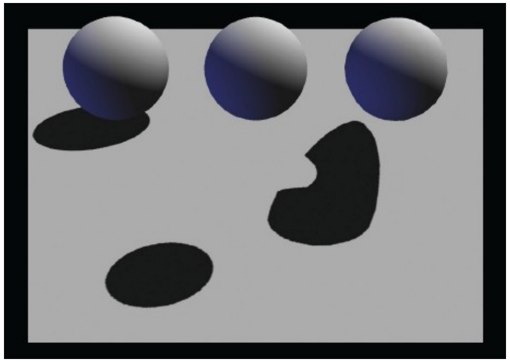
- Two types of shadows:
  - Backface shadows: A shadow on an object's surface that is oriented away from light. This type of shadows are automatically generated by the illumination model ($\theta \geq 90°$; see slide 4)
  - Projected shadows or cast shadows: Shadows cast by a part of an object's surface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

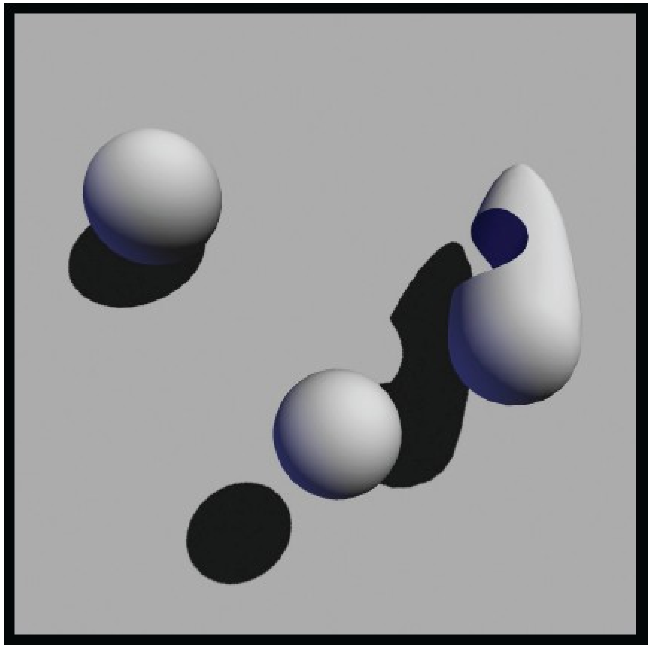


Projected Shadows

- 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.



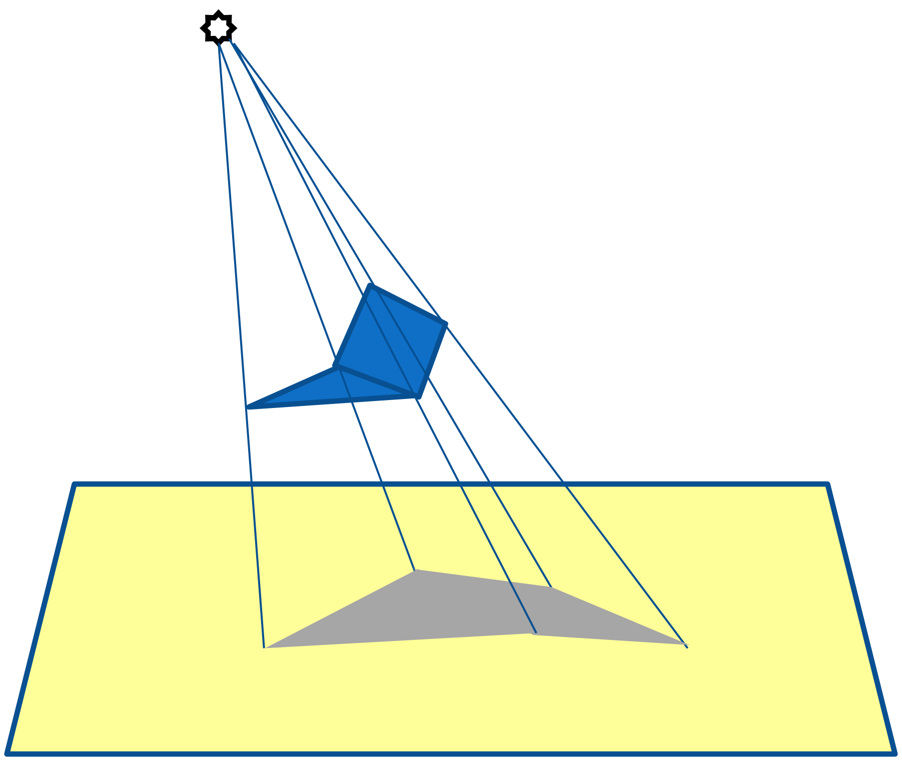




Planar Shadows on Floor Plane (*y*=0)

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();
```