

Computer Graphics Lighting



Oregon State
University



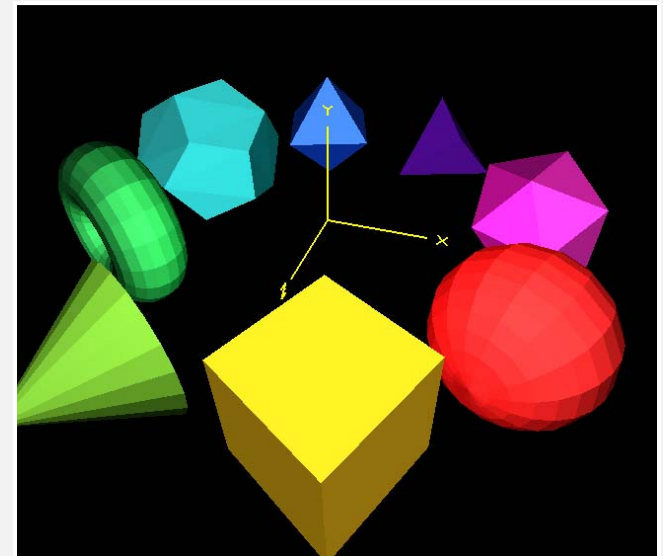
This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State
University
Computer Graphics

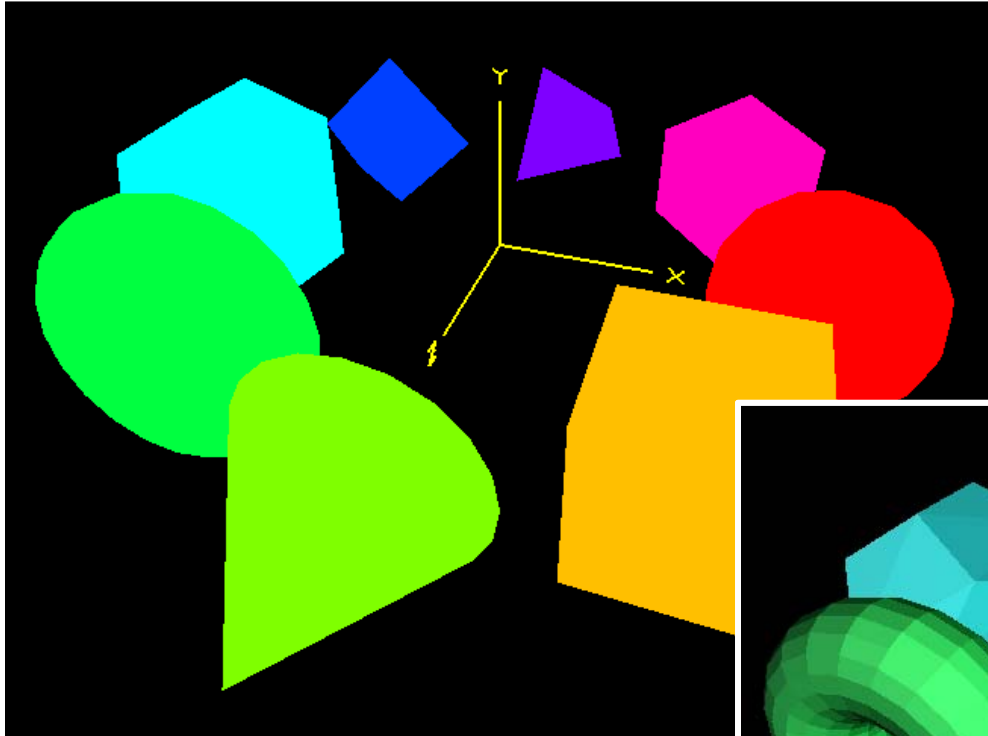
Mike Bailey

mjb@cs.oregonstate.edu

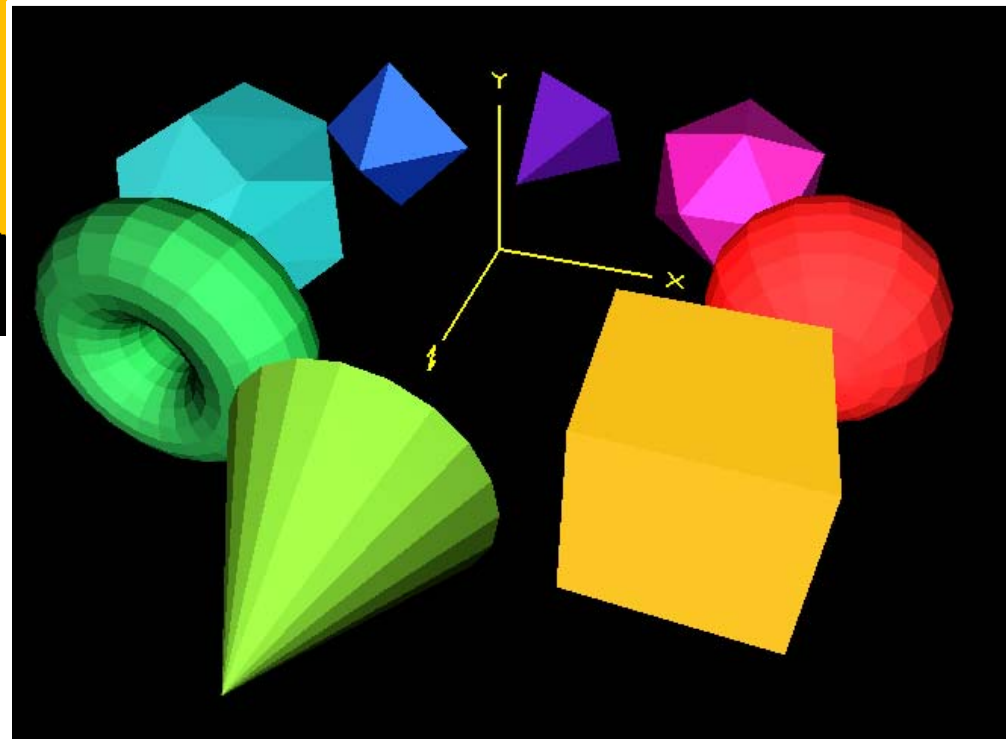


Why Do We Care About Lighting?

Lighting “dis-ambiguates” 3D scenes



Without lighting



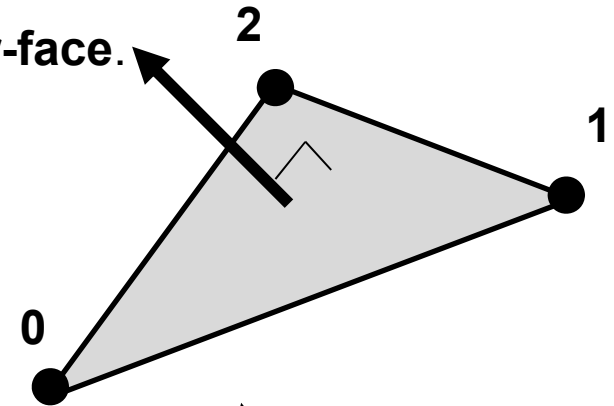
With lighting

The Surface Normal

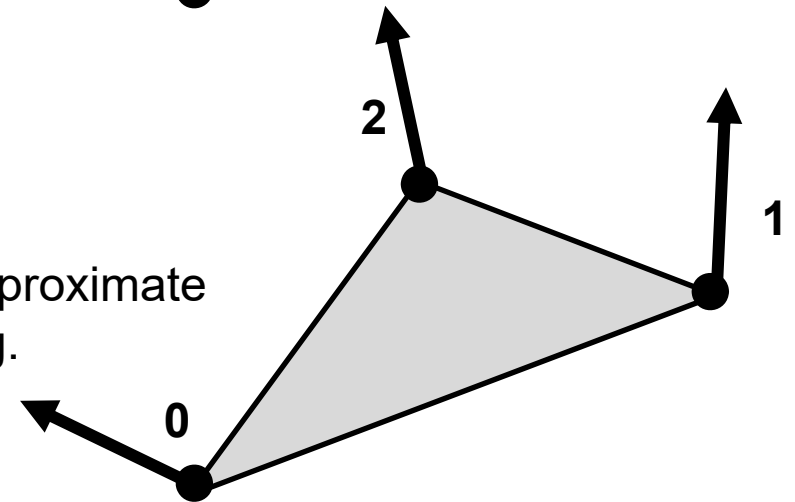
A surface normal is a vector perpendicular to the surface.

Sometimes surface normals are defined or computed **per-face**.

$$\mathbf{n} = (\mathbf{P1} - \mathbf{P0}) \times (\mathbf{P2} - \mathbf{P0})$$



Sometimes they are defined **per-vertex** to best approximate the underlying surface that the face is representing.



Setting a Surface Normal in OpenGL

```
glMatrixMode( GL_MODELVIEW );
```

```
glTranslatef( tx, ty, tz );
```

```
glRotatef( degrees, ax, ay, az );
```

```
glScalef( sx, sy, sz );
```

```
glNormal3f( nx, ny, nz ); ← Per-face
```

```
glColor3f( r, g, b );
```

```
glBegin( GL_TRIANGLES );
```

```
    glVertex3f( x0, y0, z0 );
```

```
    glVertex3f( x1, y1, z1 );
```

```
    glVertex3f( x2, y2, z2 );
```

```
glEnd( );
```



Oregon State
University

Computer Graphics

Setting a Surface Normal in OpenGL

```
glMatrixMode( GL_MODELVIEW );
```

```
glTranslatef( tx, ty, tz );
```

```
glRotatef( degrees, ax, ay, az );
```

```
glScalef( sx, sy, sz );
```

```
glColor3f( r, g, b );
```

```
glBegin(GL_TRIANGLES );
```

```
    glNormal3f( nx0, ny0, nz0 );
```

```
    glVertex3f( x0, y0, z0 );
```

```
    glNormal3f( nx1, ny1, nz1 );
```

```
    glVertex3f( x1, y1, z1 );
```

```
    glNormal3f( nx2, ny2, nz2 );
```

```
    glVertex3f( x2, y2, z2 );
```

```
glEnd( );
```

Per-vertex



Oregon State
University

Computer Graphics

Flat Shading (Per-face)

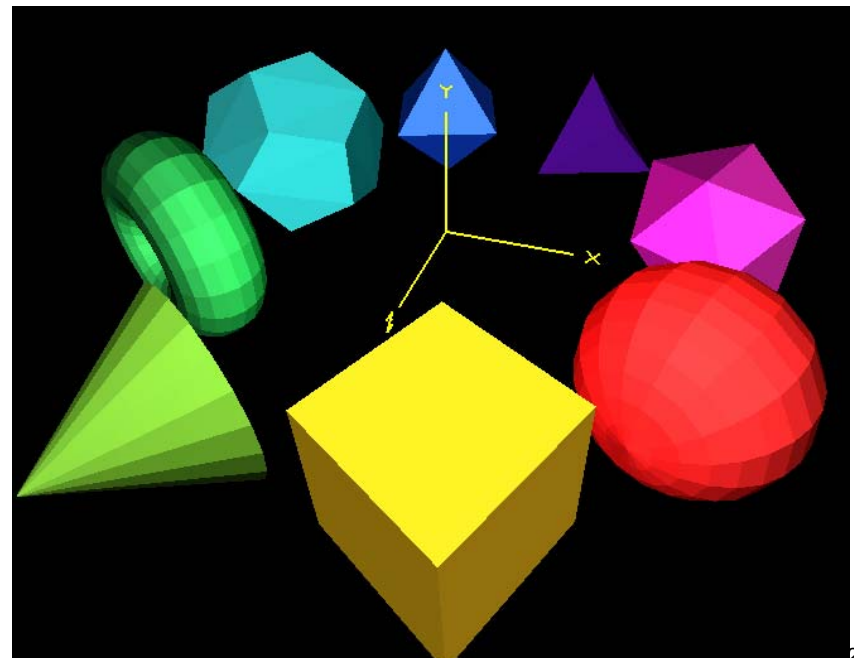
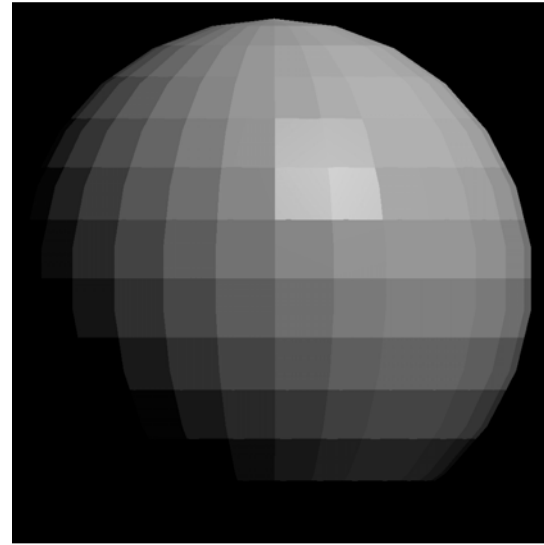
6

```
glMatrixMode( GL_MODELVIEW );
```

```
glTranslatef( tx, ty, tz );  
glRotatef( degrees, ax, ay, az );  
glScalef( sx, sy, sz );
```

```
glShadeModel( GL_FLAT );  
glNormal3f( nx, ny, nz );
```

```
glColor3f( r, g, b );  
glBegin( GL_TRIANGLES );  
    glVertex3f( x0, y0, z0 );  
    glVertex3f( x1, y1, z1 );  
    glVertex3f( x2, y2, z2 );  
glEnd( );
```



Oregon State
University

Computer Graphics

Smooth Shading (Per-vertex)

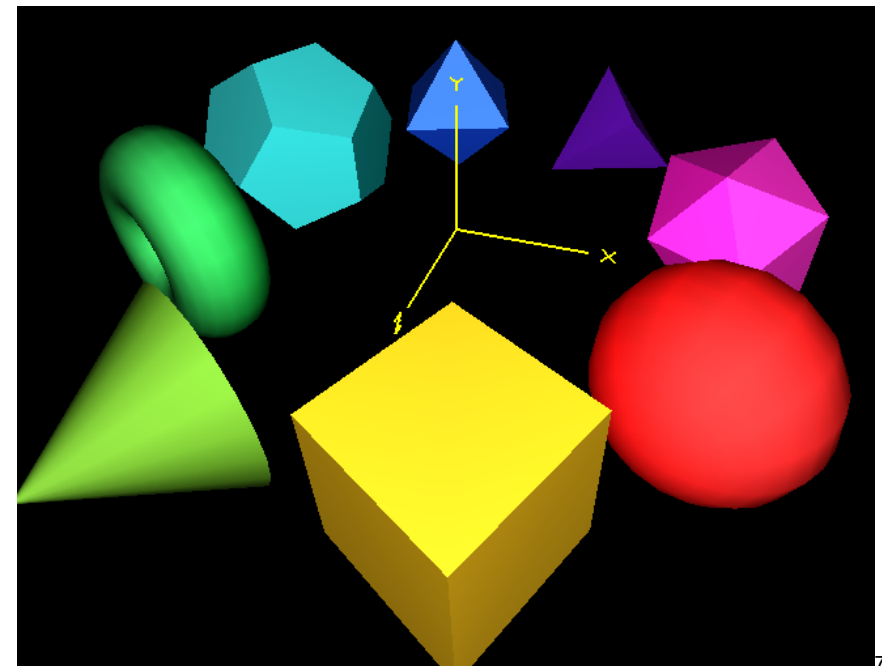
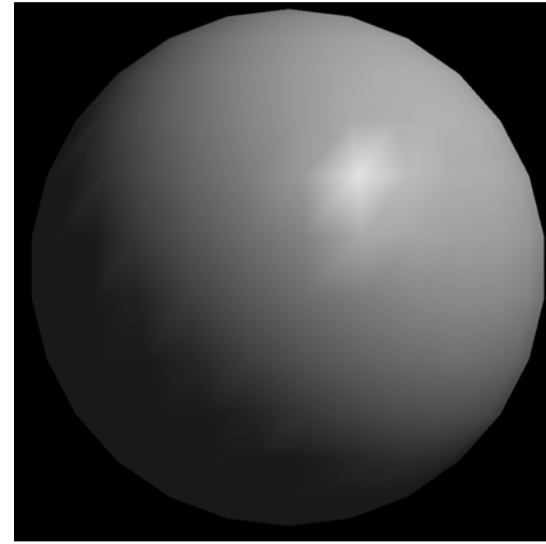
7

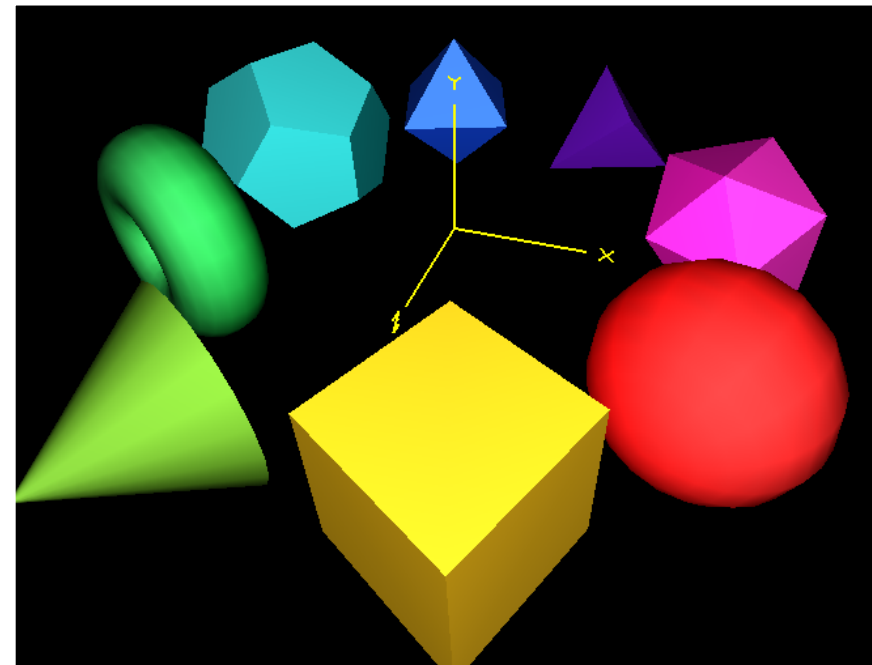
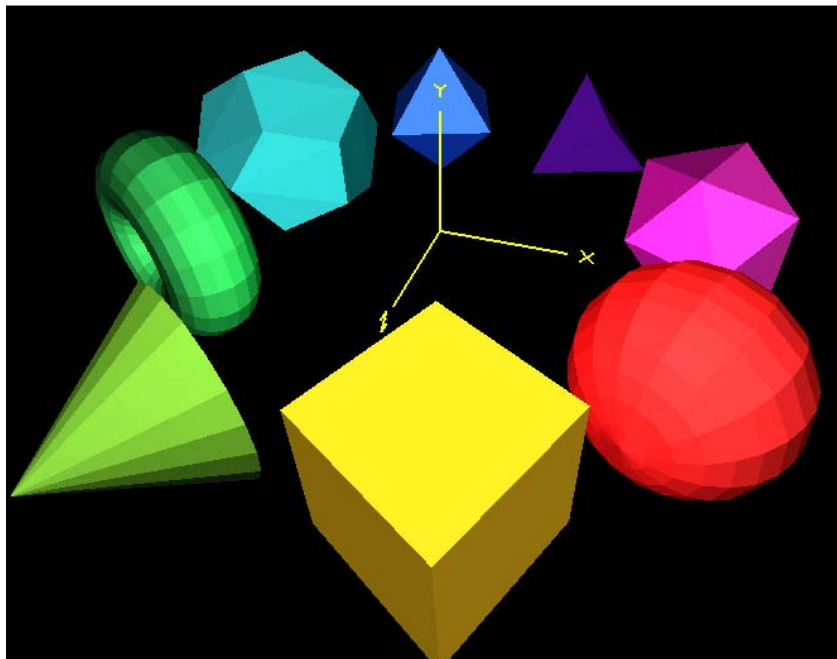
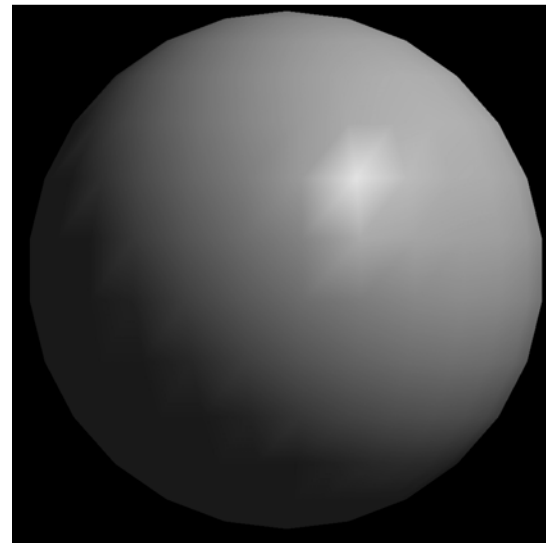
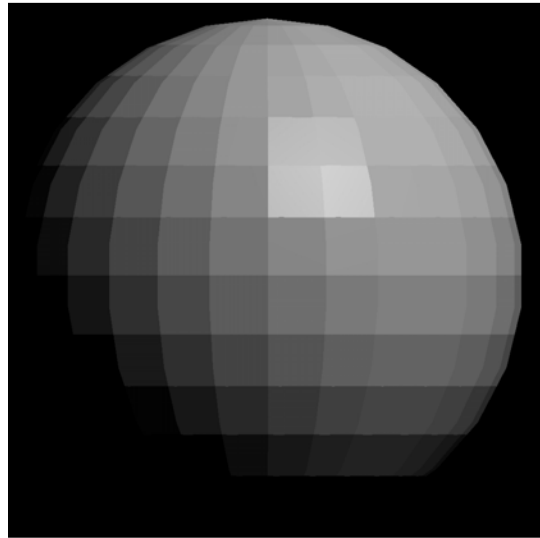
```
glMatrixMode( GL_MODELVIEW );
```

```
glTranslatef( tx, ty, tz );  
glRotatef( degrees, ax, ay, az );  
glScalef( sx, sy, sz );
```

```
glShadeModel( GL_SMOOTH );
```

```
glColor3f( r, g, b );  
glBegin( GL_TRIANGLES );  
    glNormal3f( nx0, ny0, nz0 );  
    glVertex3f( x0, y0, z0 );  
    glNormal3f( nx1, ny1, nz1 );  
    glVertex3f( x1, y1, z1 );  
    glNormal3f( nx2, ny2, nz2 );  
    glVertex3f( x2, y2, z2 );  
glEnd( );
```





OpenGL Surface Normals Need to be Unitized by Someone

```
glTranslatef( tx, ty, tz );  
glRotatef( degrees, ax, ay, az );  
glScalef( sx, sy, sz );  
  
glNormal3f( nx, ny, nz );
```

OpenGL expects the normal vector to be a unit vector, that is:
 $nx^2 + ny^2 + nz^2 = 1$

If it is not, or if you are using scaling transformations, you can force OpenGL to do the unitizing for you with:

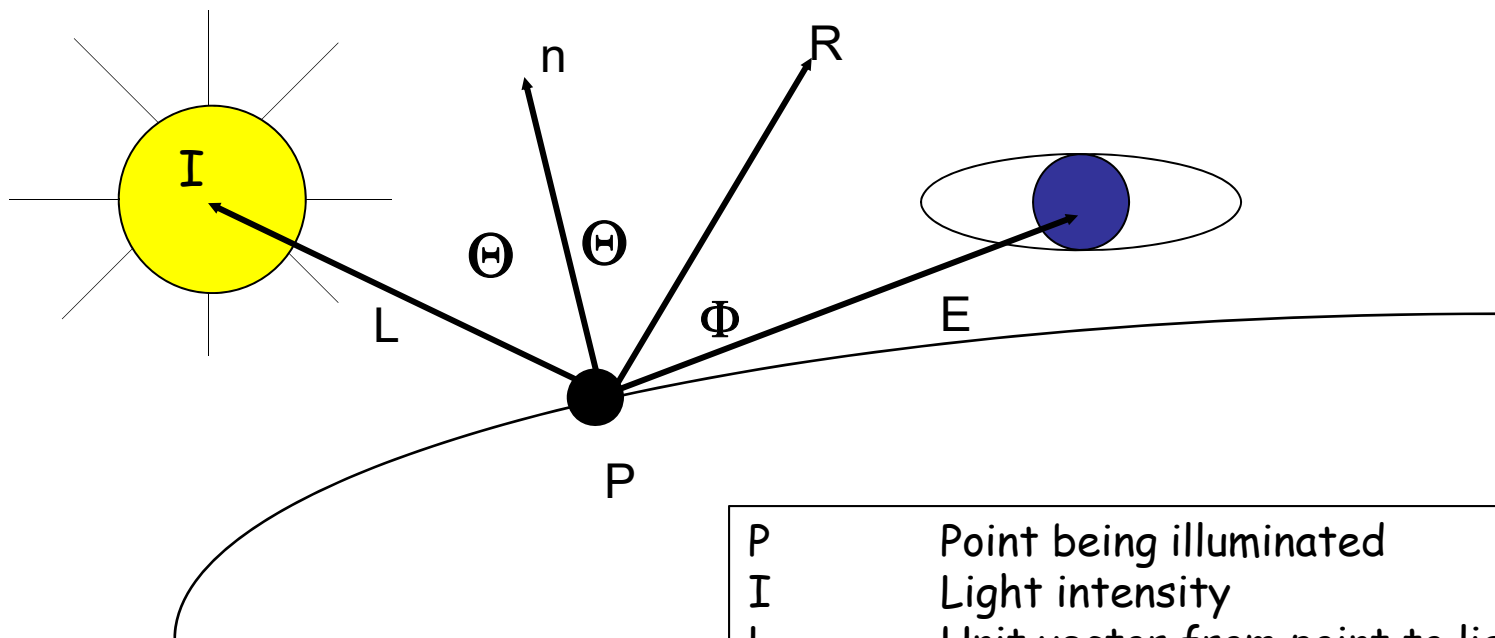
```
glEnable( GL_NORMALIZE );
```



Oregon State
University

Computer Graphics

The OpenGL “built-in” Lighting Model



P	Point being illuminated
I	Light intensity
L	Unit vector from point to light
n	Unit vector surface normal
R	Perfect reflection unit vector
E	Unit vector to eye position



The OpenGL “built-in” Lighting Model

1. **Ambient** = a constant Accounts for light bouncing “everywhere”
2. **Diffuse** = $I \cdot \cos\Theta$ Accounts for the angle between the incoming light and the surface normal
3. **Specular** = $I \cdot \cos^S\phi$ Accounts for the angle between the “perfect reflector” and the eye. The exponent, S , accounts for surface shininess

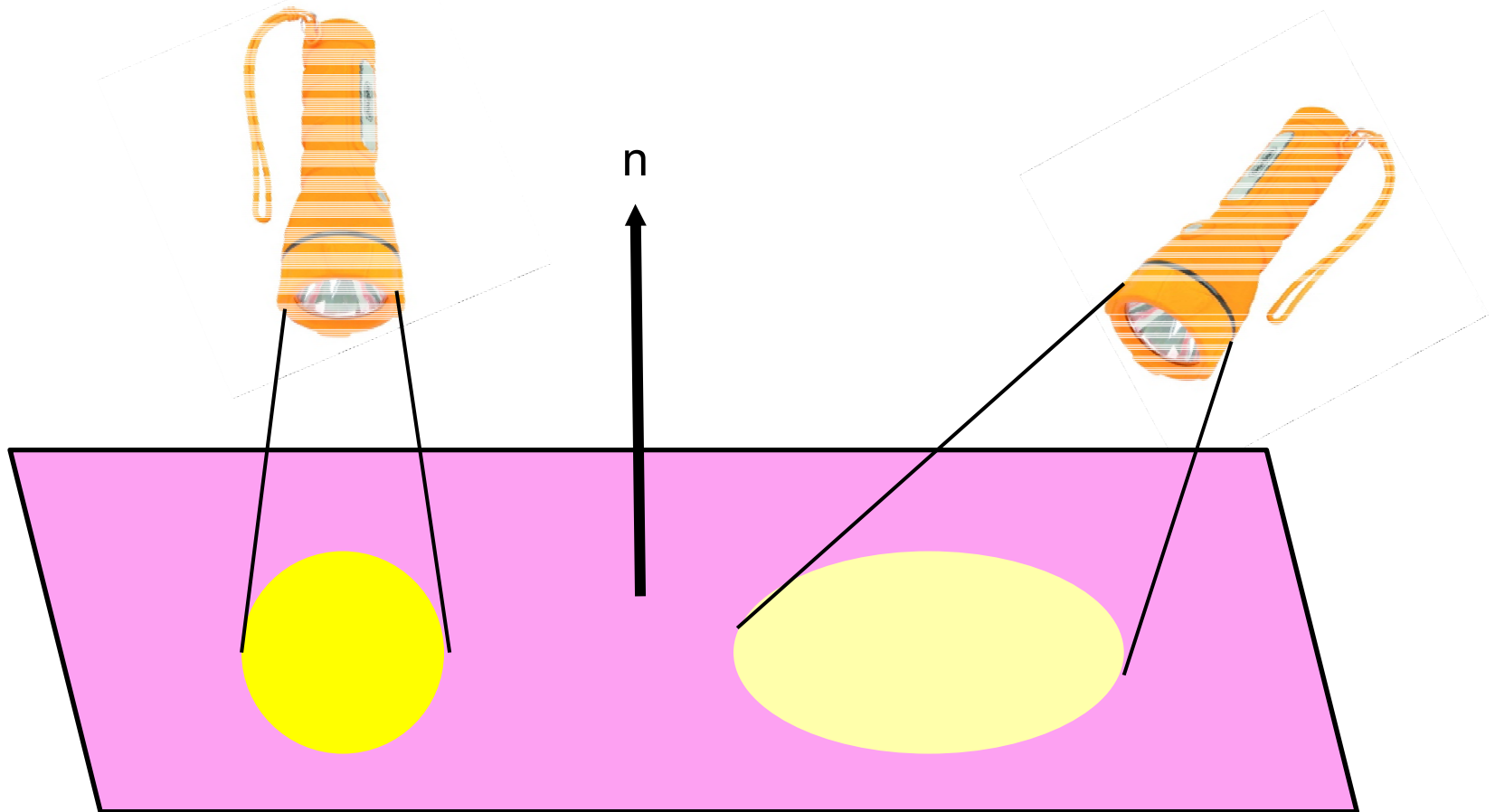
Note that $\cos\Theta$ is just the dot product between unit vectors L and n

Note that $\cos\phi$ is just the dot product between unit vectors R and E



Diffuse Lighting works because of spreading out the same amount of light energy across more surface area

$$\text{Diffuse} = I * \cos\Theta$$

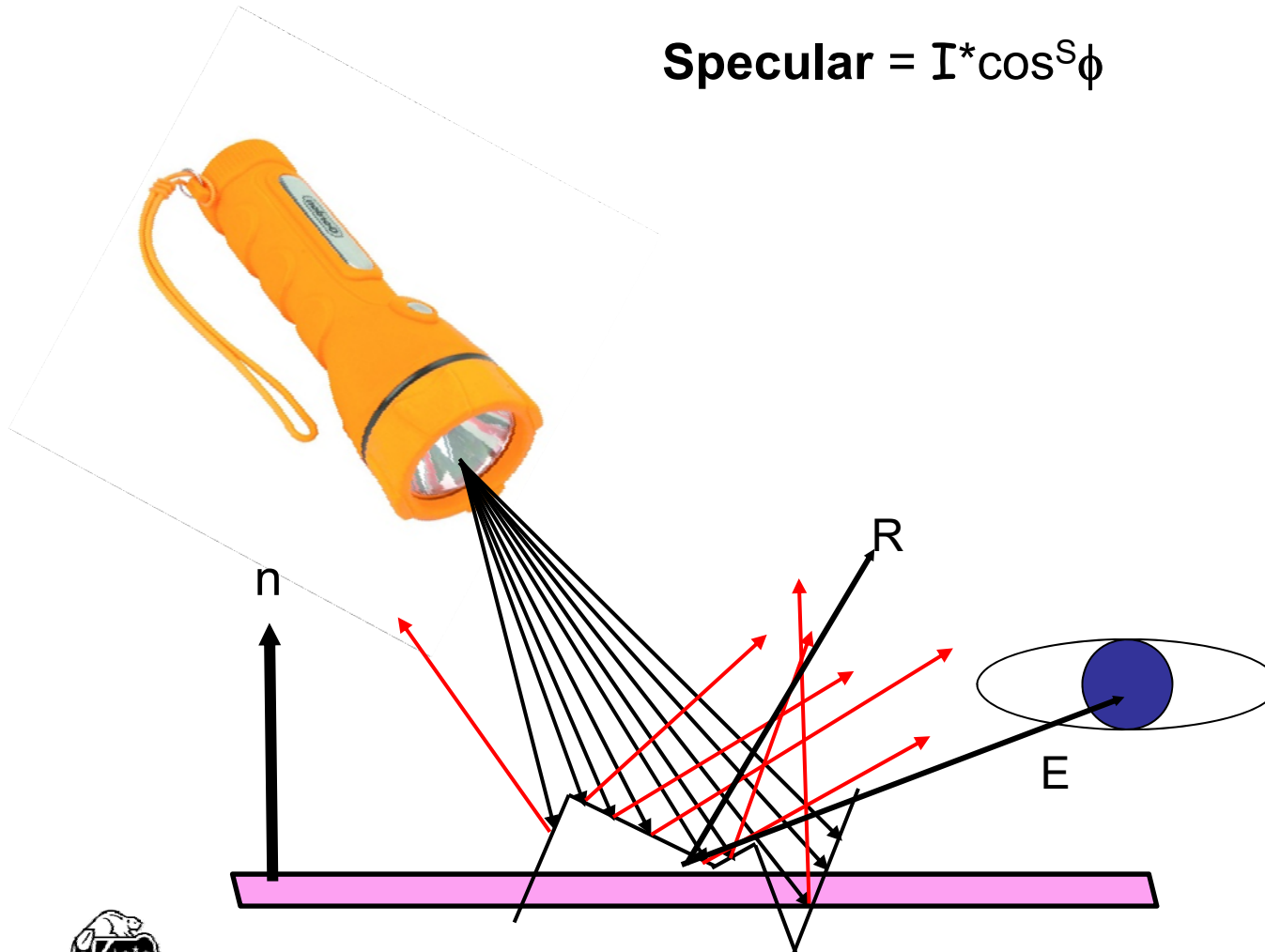


The Specular Lighting equation is a heuristic that approximates reflection from a rough surface

$$\text{Specular} = I^* \cos^S \phi$$

$S \approx$ “shininess”

$1/S \approx$ “roughness”



The Three Elements of Computer Graphics Lighting



+



+

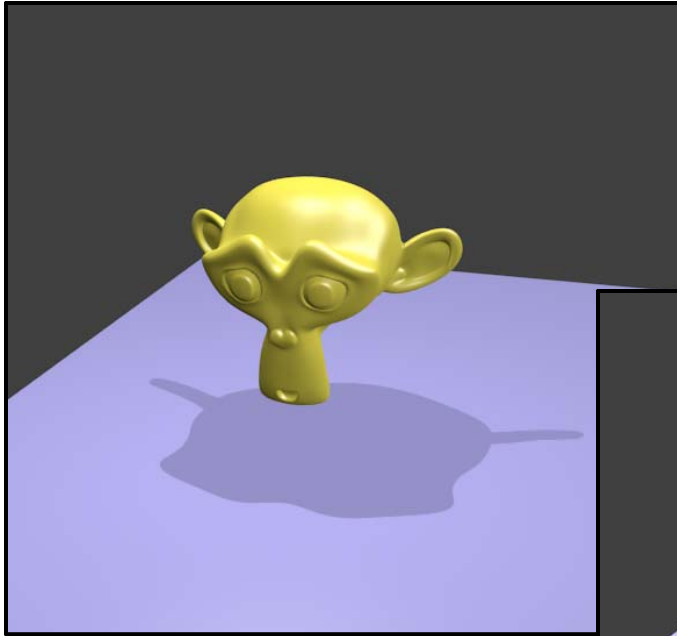


=

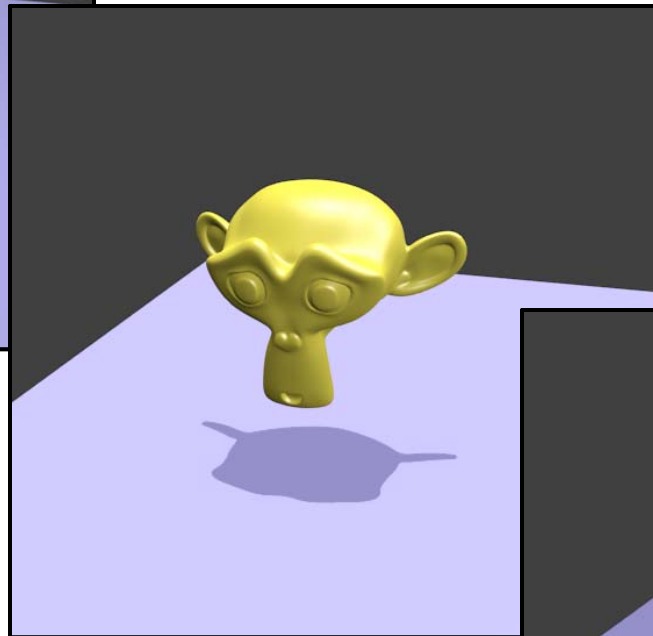


Types of Light Sources

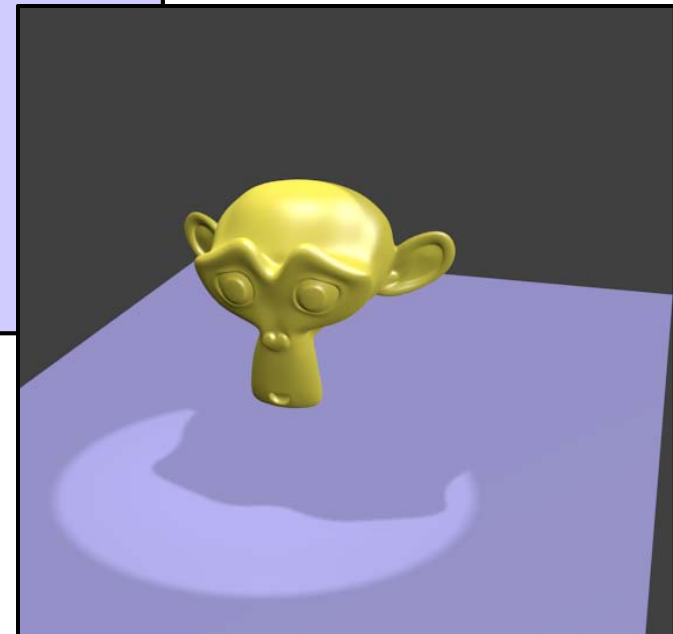
Point



Directional (Parallel, Sun)

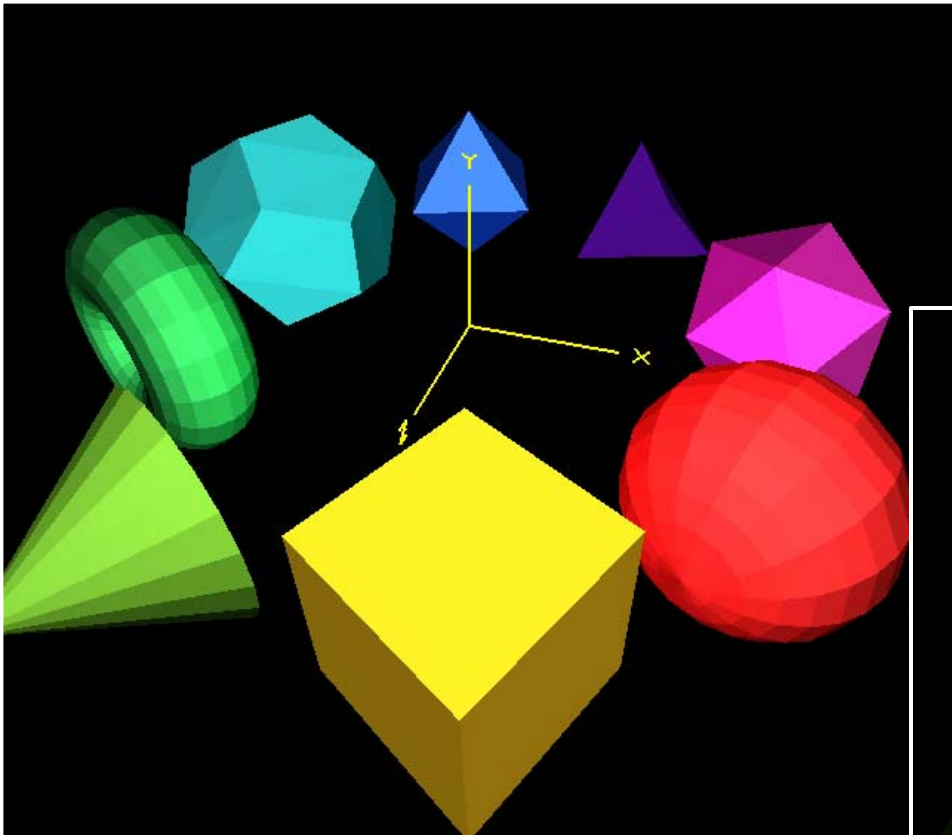


Spotlight

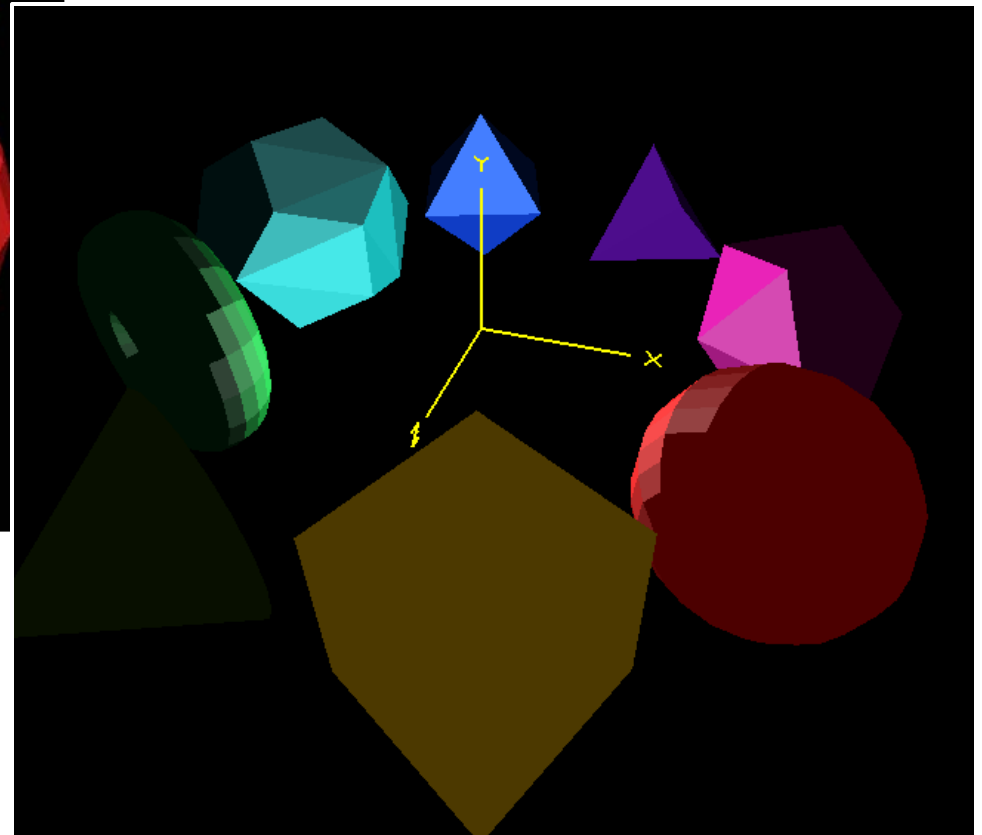


Lighting Examples

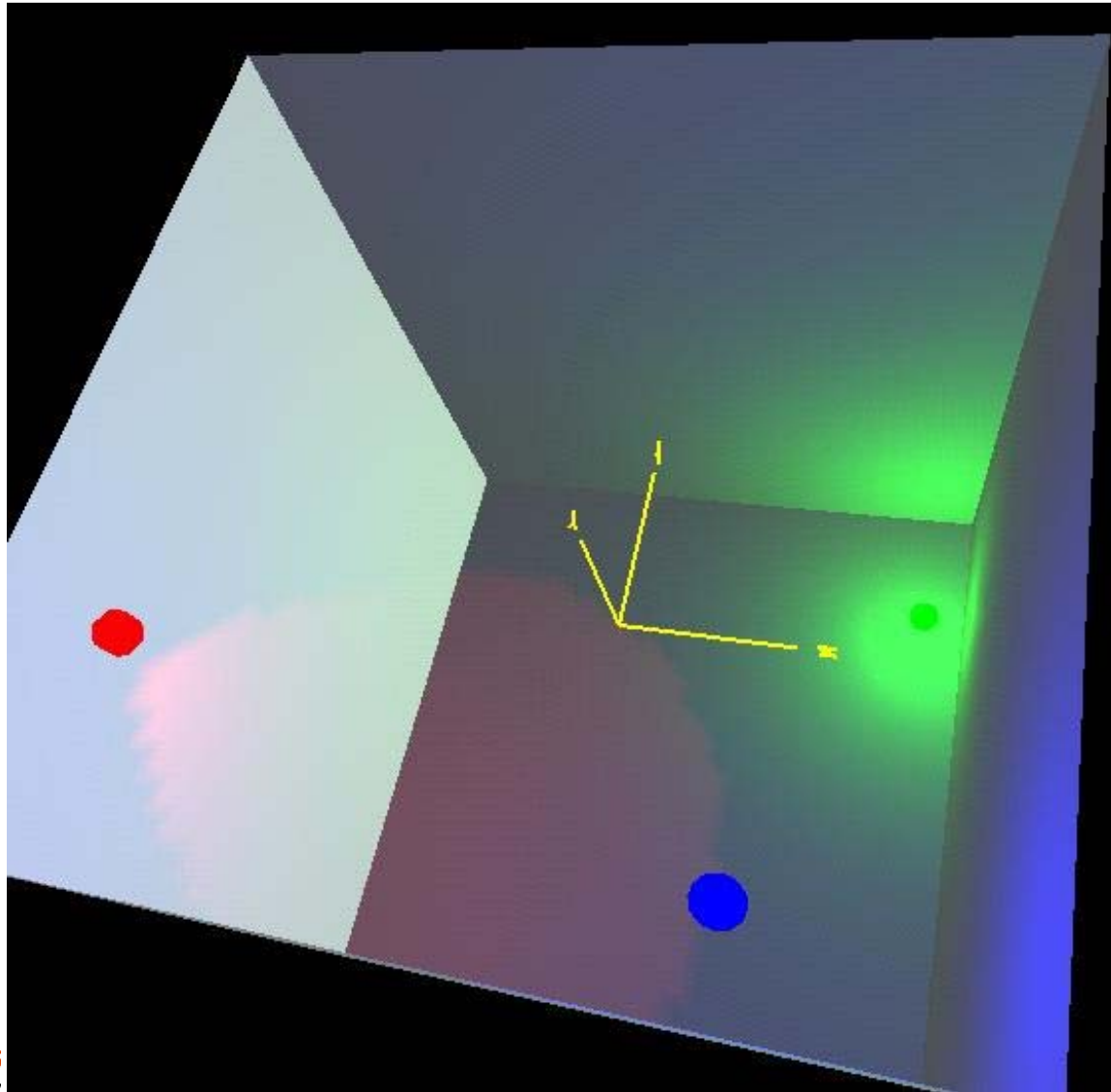
Point Light at the Eye



Point Light at the Origin



Lighting Examples



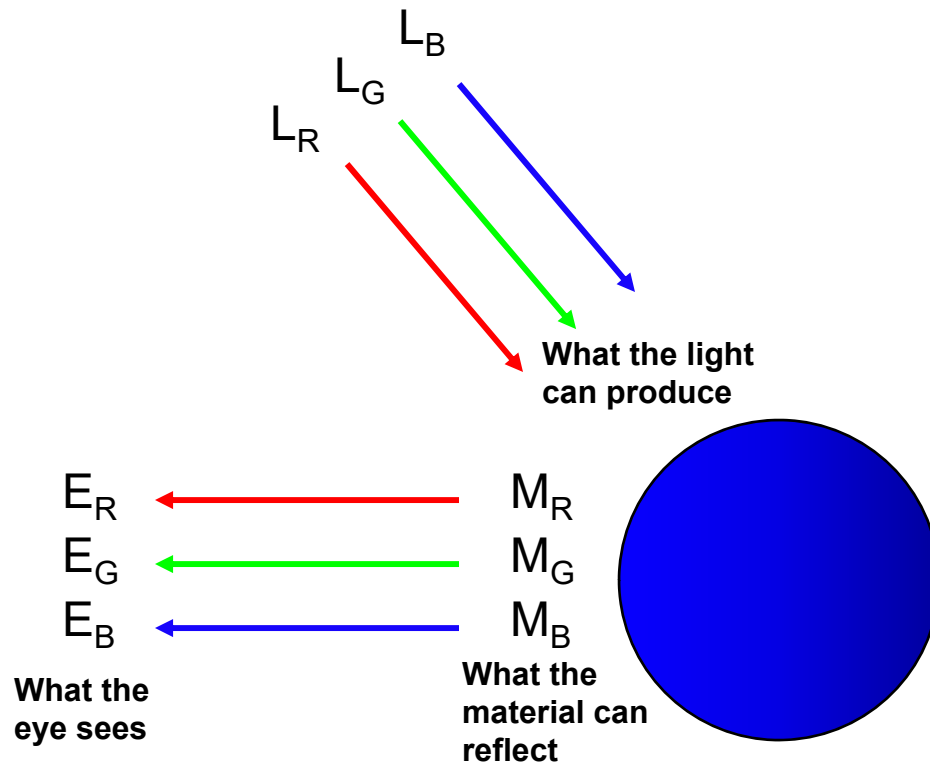
Spot Lights



Oregon St
University

Computer Graphics

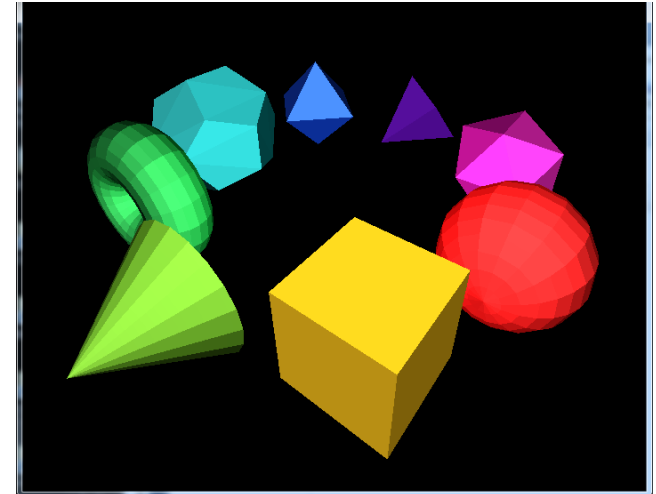
Colored Lights Shining on Colored Objects



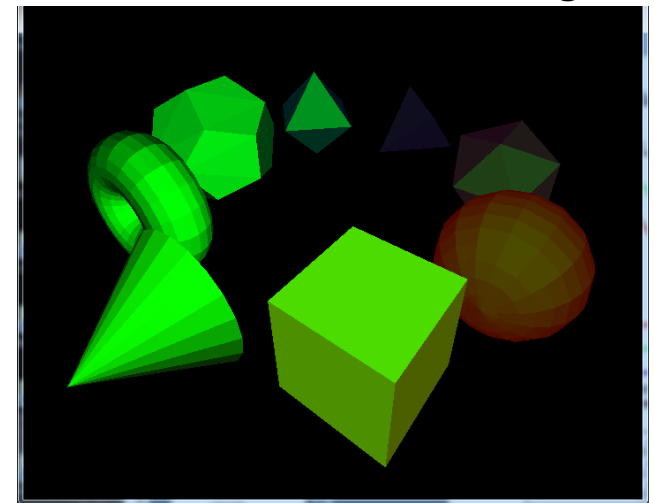
$$\begin{aligned} E_R &= L_R * M_R \\ E_G &= L_G * M_G \\ E_B &= L_B * M_B \end{aligned}$$



White Light



Green Light



Too Many Lighting Options

If there is one light and one material, the following things can be set independently:

- Global scene ambient red, green, blue
- Light position: x, y, z
- Light ambient red, green, blue
- Light diffuse red, green, blue
- Light specular red, green, blue
- Material reaction to ambient red, green, blue
- Material reaction to diffuse red, green, blue
- Material reaction to specular red, green, blue
- Material specular shininess

This makes for **25** things that can be set for just one light and one material! While many combinations are possible, some make more sense than others.



Oregon State
University

Computer Graphics

Ways to Simplify Too Many Lighting Options

1. Set the ambient light globally using, for example,
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, MulArray3(.3f, White))
I.e., set it to some low intensity of white.
2. Set the light's ambient component to zero.
3. Set the light's diffuse and specular components to the full color of the light.
4. Set each material's ambient and diffuse to the full color of the object.
5. Set each material's specular component to some fraction of white.



Oregon State
University

Computer Graphics

```
float    White[ ] = { 1.,1.,1.,1. };
```

```
// utility to create an array from 3 separate values:
```

```
float *
Array3( float a, float b, float c )
{
    static float array[4];

    array[0] = a;
    array[1] = b;
    array[2] = c;
    array[3] = 1.;
    return array;
}
```

```
// utility to create an array from a multiplier and an array:
```

```
float *
MulArray3( float factor, float array0[3] )
{
    static float array[4];

    array[0] = factor * array0[0];
    array[1] = factor * array0[1];
    array[2] = factor * array0[2];
    array[3] = 1.;
    return array;
}
```



Oregon State
University

Setting the Material Characteristics

```
glMaterialfv( GL_BACK, GL_AMBIENT,  MulArray3( .4, White ) );  
glMaterialfv( GL_BACK, GL_DIFFUSE,  MulArray3( 1., White ) );  
glMaterialfv( GL_BACK, GL_SPECULAR, Array3( 0., 0., 0. ) );  
glMaterialf ( GL_BACK, GL_SHININESS, 5. );  
glMaterialfv( GL_BACK, GL_EMISSION, Array3( 0., 0., 0. ) );
```

```
glMaterialfv( GL_FRONT, GL_AMBIENT,  MulArray3( 1., rgb ) );  
glMaterialfv( GL_FRONT, GL_DIFFUSE,  MulArray3( 1., rgb ) );  
glMaterialfv( GL_FRONT, GL_SPECULAR, MulArray3( .7, White ) );  
glMaterialf ( GL_FRONT, GL_SHININESS, 8. );  
glMaterialfv( GL_FRONT, GL_EMISSION, Array3( 0., 0., 0. ) );
```



Oregon State
University

Computer Graphics

Setting the Light Characteristics

```
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, MulArray3( .2, White ) );
glLightModeli ( GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE );
```

```
glLightfv( GL_LIGHT0, GL_AMBIENT, Array3( 0., 0., 0. ) );
glLightfv( GL_LIGHT0, GL_DIFFUSE, LightColor );
glLightfv( GL_LIGHT0, GL_SPECULAR, LightColor );
```

```
glLightf( GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1. );
glLightf( GL_LIGHT0, GL_LINEAR_ATTENUATION, 0. );
glLightf( GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0. );
```

You can have
multiple lights

// this is here because we are going to do object (and thus normal) scaling:

```
glEnable( GL_NORMALIZE );
```

$$\text{Attenuation} = \frac{1}{C + Ld + Qd^2} \quad \text{where } d \text{ is the distance from the light to the point being lit}$$

Setting the Light Position

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
```

The light position gets transformed by the value of the **ModelView matrix** at the moment the light position is set.

// if we do this, then the light will be wrt the scene at XLIGHT, YLIGHT, ZLIGHT:

```
glLightfv( GL_LIGHT0, GL_POSITION, Array3(XLIGHT, YLIGHT, ZLIGHT) );
```

// translate the object into the viewing volume:

```
gluLookAt( XEYE, YEYE, ZEYE, 0., 0., 0., 0., 1., 0. );
```

// if we do this, then the light will be wrt the eye at XLIGHT, YLIGHT, ZLIGHT:

```
// glLightfv( GL_LIGHT0, GL_POSITION, Array3(XLIGHT, YLIGHT, ZLIGHT) );
```


// perform the rotations and scaling about the origin:

```
glRotatef( Xrot, 1., 0., 0. );
glRotatef( Yrot, 0., 1., 0. );
glScalef( Scale, Scale, Scale );
```

// if we do this, then the light will be wrt to the object at XLIGHT, YLIGHT, ZLIGHT:

```
// glLightfv( GL_LIGHT0, GL_POSITION, Array3(XLIGHT, YLIGHT, ZLIGHT) );
```

// specify the shading model:

```
glShadeModel( GL_SMOOTH );
```

// enable lighting:

```
glEnable( GL_LIGHTING );
```

You can enable and disable lighting “at all”.
(This toggles between using the lighting equations
say and what glColor3f() says

```
glEnable( GL_LIGHT0 );
```

You can enable and disable each light independently

// draw the objects:

...

Or

glDisable(GL_LIGHTING);

Computer Graphics

It is usually good form to disable the lighting after you
are done using it

Sidebar: Why are Light Positions 4-element arrays where the 4th element is 1.0? Homogeneous Coordinates!

```
float *  
Array3( float a, float b, float c )  
{  
    static float array[4];  
  
    array[0] = a;  
    array[1] = b;  
    array[2] = c;  
    array[3] = 1.;  
    return array;  
}
```

We usually think of a 3D point as being represented by a triple: (x,y,z).

Using homogeneous coordinates, we add a 4th number: (x,y,z,w)

Graphics systems take (x,y,z,w) and then divide x, y, and z by w before using them.

$$X = \frac{x}{w}, Y = \frac{y}{w}, Z = \frac{z}{w}$$



Oregon State
University

Computer Graphics

Thus (1,2,3,1) , (2,4,6,2) , (-1,-2,-3,-1) all represent the same 3D point.

This Lets us Represent Points at Infinity

27

This is useful to be able specify a parallel light source by placing the light source location at infinity.

The point $(1,2,3,1)$ represents the 3D point $(1,2,3)$

The point $(1,2,3,.5)$ represents the 3D point $(2,4,6)$

The point $(1,2,3,.01)$ represents the point $(100,200,300)$

So, $(1,2,3,0)$ represents a point at infinity, but along the ray from the origin through $(1,2,3)$.

Points-at-infinity are used for parallel light sources and some shadow algorithms



Oregon State
University

Computer Graphics

Additional Parameters for Spotlights

28

glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, Array3(xdir,ydir,zdir))

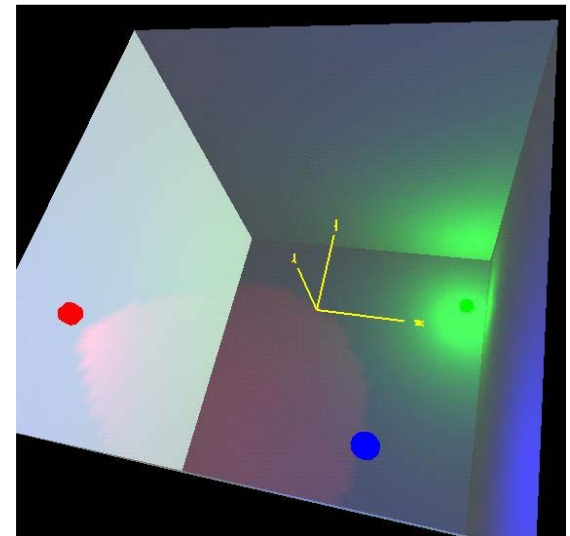
Specifies the spotlight-pointing direction. This gets transformed by the current value of the ModelView matrix.

glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e)

Specifies the spotlight directional intensity. This acts very much like the exponent in the specular lighting equation.

glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, deg)

Specifies the spotlight maximum spread angle.



**Oregon State
University**

Computer Graphics

Shortcuts I Like

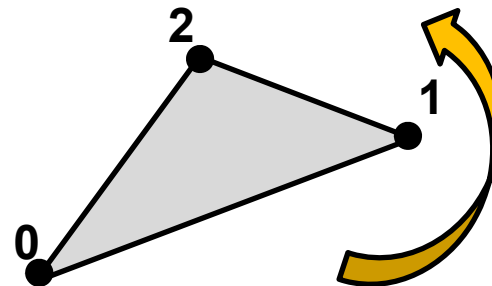
```
void
SetMaterial( float r, float g, float b, float shininess )
{
    glMaterialfv( GL_BACK, GL_EMISSION, Array3( 0., 0., 0. ) );
    glMaterialfv( GL_BACK, GL_AMBIENT, MulArray3( .4f, White ) );
    glMaterialfv( GL_BACK, GL_DIFFUSE, MulArray3( 1., White ) );
    glMaterialfv( GL_BACK, GL_SPECULAR, Array3( 0., 0., 0. ) );
    glMaterialf ( GL_BACK, GL_SHININESS, 2.f );

    glMaterialfv( GL_FRONT, GL_EMISSION, Array3( 0., 0., 0. ) );
    glMaterialfv( GL_FRONT, GL_AMBIENT, Array3( r, g, b ) );
    glMaterialfv( GL_FRONT, GL_DIFFUSE, Array3( r, g, b ) );
    glMaterialfv( GL_FRONT, GL_SPECULAR, MulArray3( .8f, White ) );
    glMaterialf ( GL_FRONT, GL_SHININESS, shininess );
}
```

Definition of GL_FRONT and GL_BACK:

GL_FRONT side

Vertices are CCW when viewed from the outside



GL_BACK side

Vertices are CW when viewed from the outside



**Oregon State
University**

Computer Graphics

Shortcuts I Like

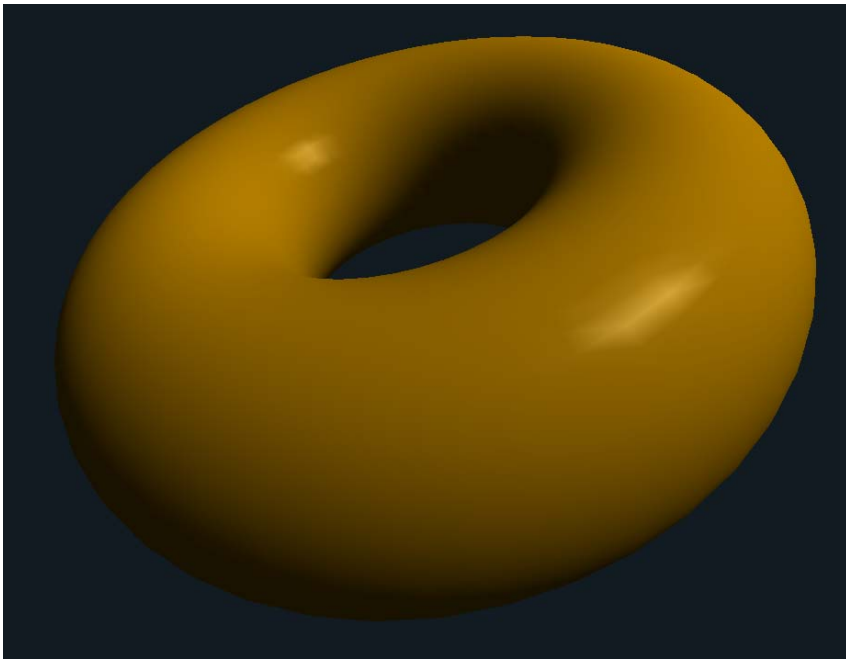
```
void
SetPointLight( int ilight, float x, float y, float z, float r, float g, float b )
{
    glLightfv( ilight, GL_POSITION, Array3( x, y, z ) );
    glLightfv( ilight, GL_AMBIENT, Array3( 0., 0., 0. ) );
    glLightfv( ilight, GL_DIFFUSE, Array3( r, g, b ) );
    glLightfv( ilight, GL_SPECULAR, Array3( r, g, b ) );
    glLightf ( ilight, GL_CONSTANT_ATTENUATION, 1. );
    glLightf ( ilight, GL_LINEAR_ATTENUATION, 0. );
    glLightf ( ilight, GL_QUADRATIC_ATTENUATION, 0. );
    glEnable( ilight );
}
```

```
void
SetSpotLight( int ilight, float x, float y, float z, float xdir, float ydir, float zdir, float r, float g, float b )
{
    glLightfv( ilight, GL_POSITION, Array3( x, y, z ) );
    glLightfv( ilight, GL_SPOT_DIRECTION, Array3(xdir,ydir,zdir) );
    glLightf( ilight, GL_SPOT_EXPONENT, 1. );
    glLightf( ilight, GL_SPOT_CUTOFF, 45. );
    glLightfv( ilight, GL_AMBIENT, Array3( 0., 0., 0. ) );
    glLightfv( ilight, GL_DIFFUSE, Array3( r, g, b ) );
    glLightfv( ilight, GL_SPECULAR, Array3( r, g, b ) );
    glLightf ( ilight, GL_CONSTANT_ATTENUATION, 1. );
    glLightf ( ilight, GL_LINEAR_ATTENUATION, 0. );
    glLightf ( ilight, GL_QUADRATIC_ATTENUATION, 0. );
    glEnable( ilight );
}
```

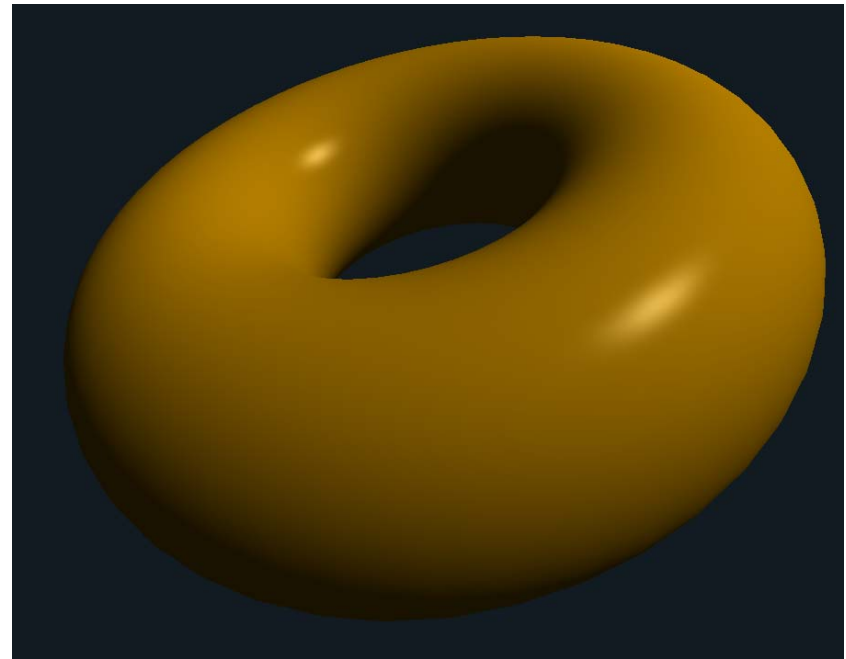
Sidebar: Note that we are computing the light intensity at each vertex³¹ first, and then interpolating that intensity across the polygon second

That is, you are only using the lighting model *at each vertex*.

You can do an even better job if you interpolate the normal across the polygon first, and then compute the light intensity with the lighting model at each fragment second:



Per-vertex



Per-fragment



**Oregon State
University**

Computer Graphics

But, for that you will need the Shaders course (CS 457/557) ³²

Per-vertex



Per-fragment



Sidebar: Smooth Shading can also interpolate vertex colors,³³ not just the results of the lighting model

```
glMatrixMode( GL_MODELVIEW );
```

```
glTranslatef( tx, ty, tz );  
glRotatef( degrees, ax, ay, az );  
glScalef( sx, sy, sz );
```

```
glShadeModel( GL_SMOOTH );
```

```
glBegin( GL_TRIANGLES );  
    glColor3f( r0, g0, b0 );  
    glVertex3f( x0, y0, z0 );  
    glColor3f( r1, g1, b1 );  
    glVertex3f( x1, y1, z1 );  
    glColor3f( r2, g2, b2 );
```

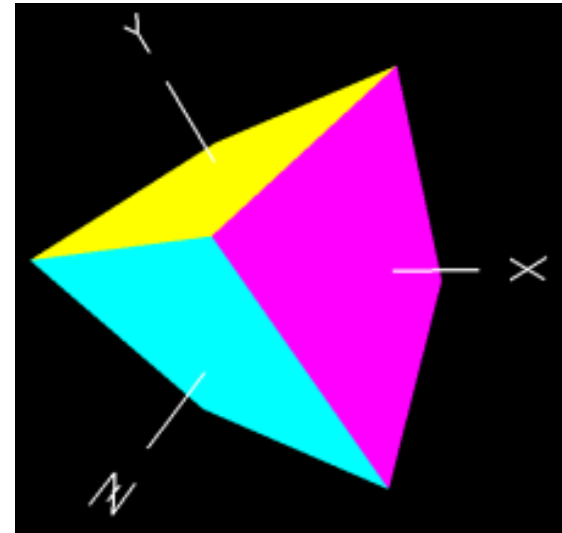
```
glEnd( );
```



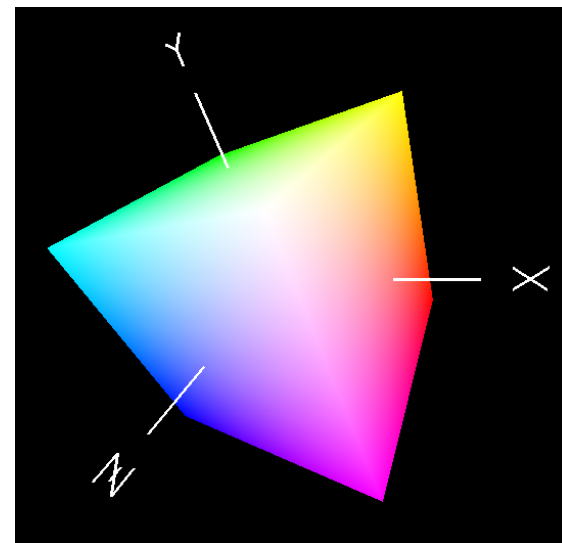
**Oregon State
University**

Computer Graphics

Flat

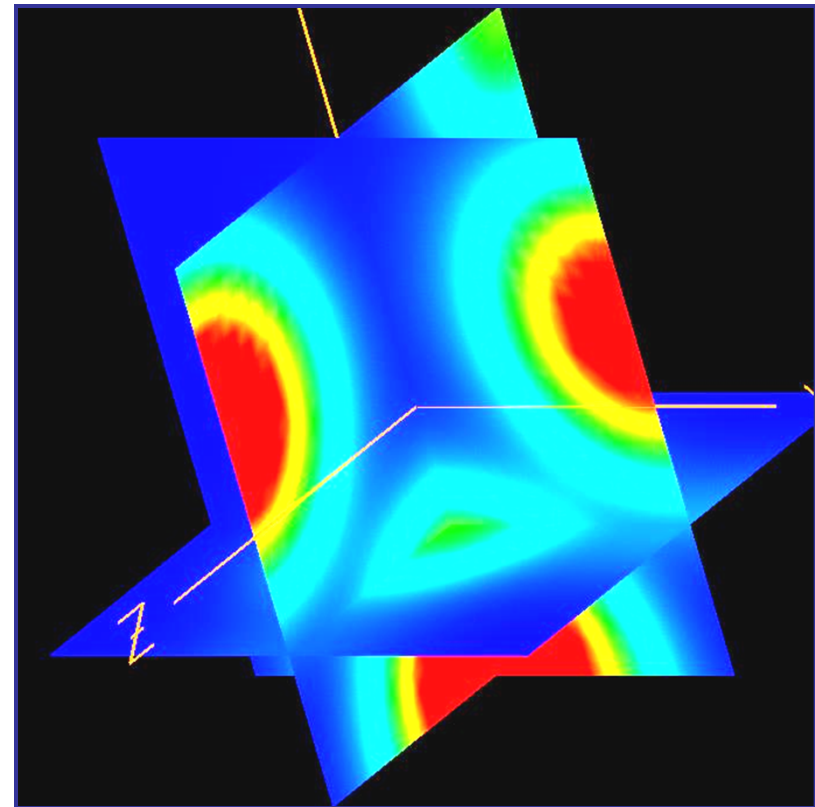


Smooth



Smooth Shading can also interpolate vertex colors, not just the results of the lighting model

This is especially useful when using colors for scientific visualization:



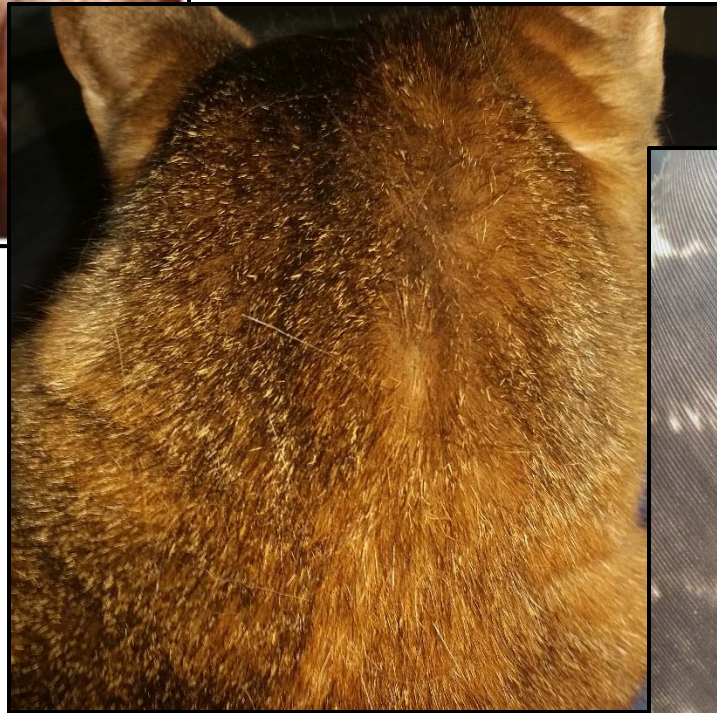
Tricky Lighting Situations

35

Watch for these in movies!



Hair



Fur

Feathers



Tricky Lighting Situations

36

Notice the lighting in the fur!

