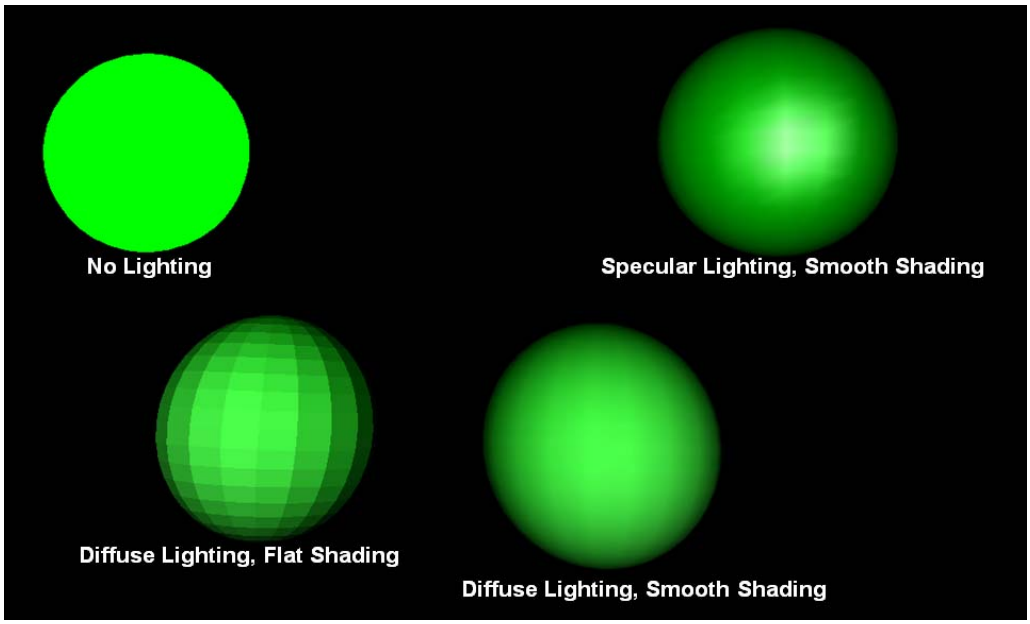
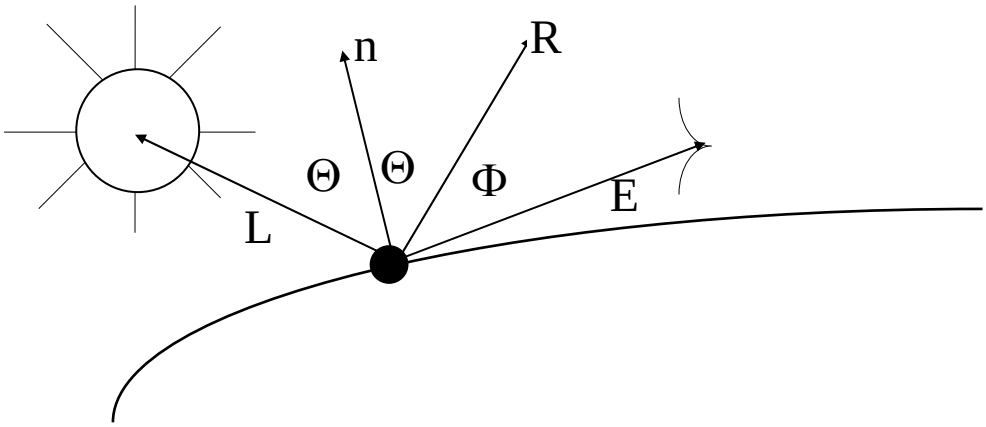
OpenGL Lighting Made (Somewhat) Easier

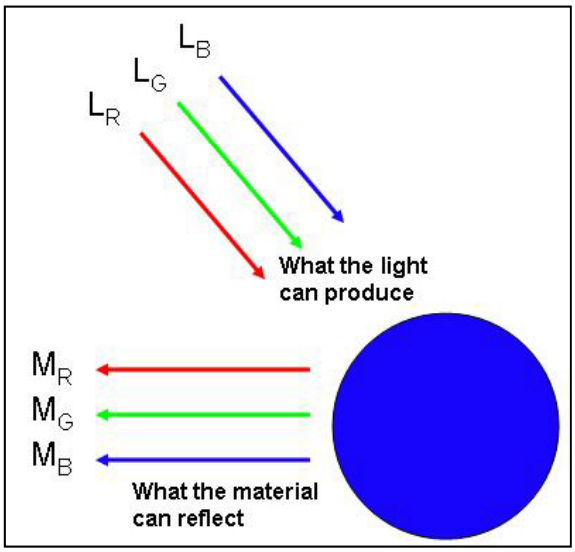
Mike Bailey
Oregon State University



If you've tried using the GLUT solid objects so far, you have noticed that they turn out to be big colored blobs on the screen, like the left-most image shown here. This is because you have given them no knowledge of lighting behaviors. What we need is a *lighting model*, that is, some mathematics that tells us how bright or dim different parts of a surface will be given some scene lights. The typical computer graphics lighting model is based on the diagram below:



The light has 3 color components: red, green, and blue. The object material's ability to reflect light has 3 components: red, green, and blue. These are pair-wise multiplied for ambient, diffuse, and specular to see how much of each component is re-transmitted by the object.



Red = $L_R * M_R$
Green = $L_G * M_G$
Blue = $L_B * M_B$

These three equations are performed separately for ambient, diffuse, and specular. The diffuse product is then multiplied by $\cos(\theta)$ and the specular component is multiplied by $\cos^s(\phi)$.

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.

Ways to Simplify Too Many Options

1. Set the ambient light globally using glLightModelfv(GL_LIGHT_MODEL_AMBIENT, *). Set it to some low intensity of white.

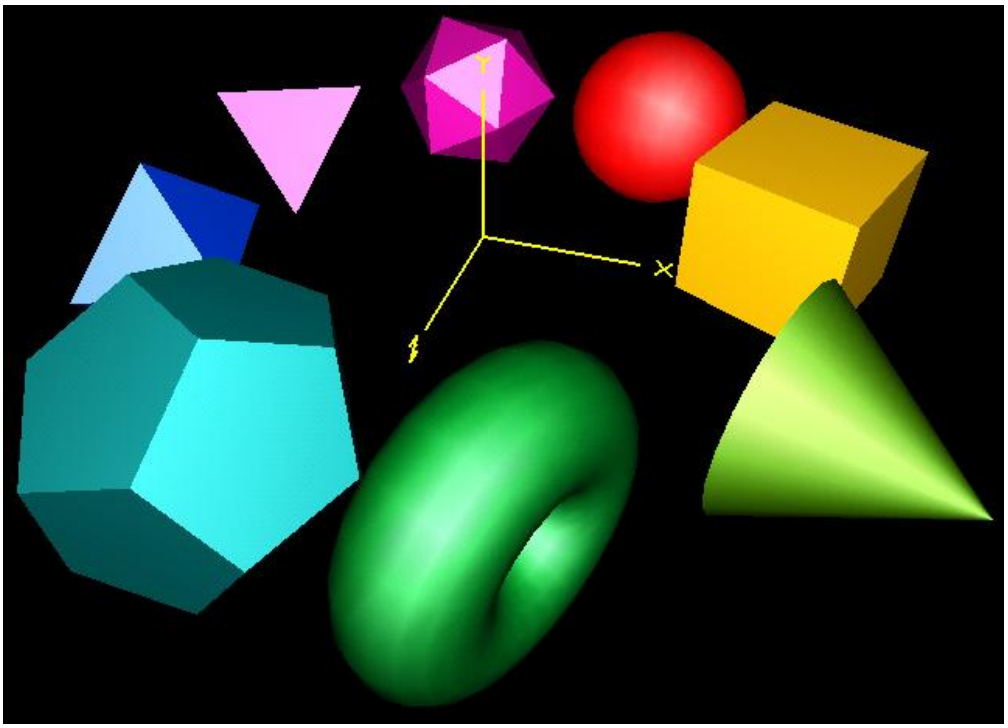
2. Set the light's ambient component to zero.

3. Set the light's diffuse and specular components to the full color of the light.

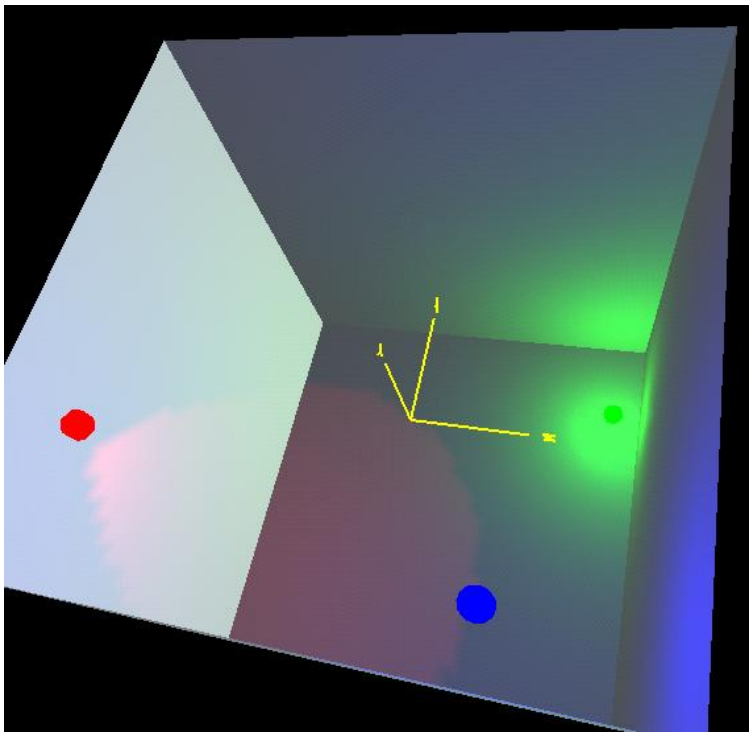
4. Set each material's ambient and diffuse to the full color of the object.

5. Set each material's specular component to some fraction of white.

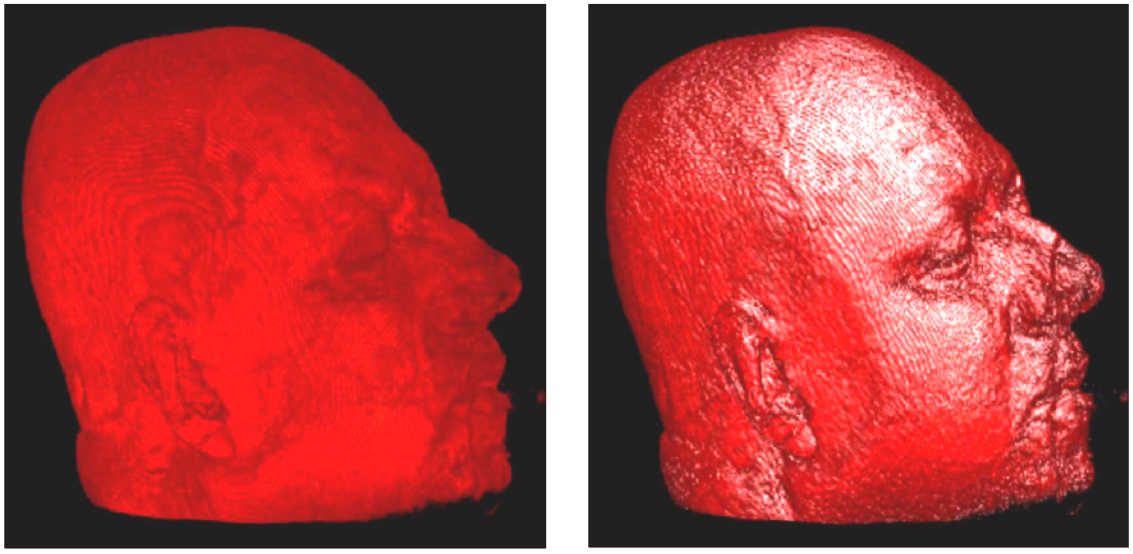
Examples



Point light at the eye, smooth shading, shiny surfaces



Spot light (red), point lights (green, blue) bouncing through the scene



Recall that lighting *really* enhances the appearance of volume data

```
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;
}


void
Display( void )
{
```

<< set the viewport and the viewing volume as usual >>

```
      // define some of the lighting parameters:
      // (these could have gone in InitGraphics(), because they
      // do not change)
```

```
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_EMISSION, Array3( 0., 0., 0. ) );

glLightModelfv( GL_LIGHT_MODEL_AMBIENT, MulArray3( .2, White ) );

glLightfv( GL_LIGHT0, GL_AMBIENT,   Array3( 0., 0., 0. ) );
glLightf ( GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1. );
glLightf ( GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0. );


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

glEnable( GL_NORMALIZE );


// these lighting calls need to be here since they
// get changed via the pop-up menus:

glLightModeli ( GL_LIGHT_MODEL_TWO_SIDE, TwoSide );
if( Attenuate == NONE )
      glLightf ( GL_LIGHT0, GL_LINEAR_ATTENUATION, 0. );
else
      glLightf ( GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.20 );
glLightfv( GL_LIGHT0, GL_DIFFUSE,   LightColor );
glLightfv( GL_LIGHT0, GL_SPECULAR,  LightColor );
if( Shiny == DULL )
{
      glMaterialf ( GL_FRONT, GL_SHININESS, 2. );
      glMaterialfv( GL_FRONT, GL_SPECULAR, MulArray3( .5, White ) );
}
else
{
      glMaterialf ( GL_FRONT, GL_SHININESS, 8. );
      glMaterialfv( GL_FRONT, GL_SPECULAR, MulArray3( .7, White ) );
}


glMatrixMode( GL_MODELVIEW );
glLoadIdentity();


// 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 at the eye:

if( Light == AT_EYE )
{
      glLightfv( GL_LIGHT0, GL_POSITION,  Array3( XEYE, YEYE, ZEYE ) );
}


// if we do this, then the light will be stationary at the
// origin of the scene:

if( Light == AT_ORIGIN )
{
      glLightfv( GL_LIGHT0, GL_POSITION,  Array3( 0., 0., 0. ) );
}
```

```
	// 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 at the
	// origin of the scene and will move with the scene:

	if( Light == WITH_SCENE )
	{
		glLightfv( GL_LIGHT0, GL_POSITION,  Array3( 0., 0., 0. ) );
	}


	// possibly draw the axes:

	if( AxesOnOff == ON  )
	{
		glDisable( GL_LIGHTING );
		glShadeModel( GL_FLAT );
		glCallList( AxesList );
	}


	// specify shading mode:

	glShadeModel( GL_SMOOTH );



	// enable hardware lighting:

	glEnable( GL_LIGHTING );
	glEnable( GL_LIGHT0 );


	// draw the objects:

	for( int i = 0; i < 8; i++ )
	{
		glCallList( ObjectLists[i] );
	}



	// swap the double-buffered framebuffers:

	glutSwapBuffers();


	// be sure the graphics buffer has been sent:

	glFlush();
}
```

```
void
InitLists()
{
	float rgb[3];	// the object color

	if( Debug )
		fprintf( stderr, "InitLists\n" );



	// create the objects:

	ObjectLists[0] = glGenLists( 1 );
	glNewList( ObjectLists[0], GL_COMPILE );

	<< set the rgb[ ] array here >>

	glMaterialfv( GL_FRONT, GL_AMBIENT, MulArray3( 1., rgb ) );
	glMaterialfv( GL_FRONT, GL_DIFFUSE, MulArray3( 1., rgb ) );
	glPushMatrix();
		glTranslatef( x, y, z );
		glutSolidSphere( 1.0, 20, 20 );
	glPopMatrix();
	glEndList();


		. . .
```

Notes:

- The light positions are in homogeneous coordinates, that is, the positions are expressed as ***4-element*** arrays: x, y, z, w
- As homogeneous coordinates, the actual light position is $^{x}/_{w}$, $^{y}/_{w}$, $^{z}/_{w}$
- For a light position at infinity (that is, a directional, or parallel, light source), set its w = 0.
- The light position gets multiplied by the current ModelView matrix before it is used. This is so that lights can be attached to objects (e.g., headlights) and transformed in the same way.