# Lights, Camera, Action!

- Interestingly, this cliché is actually a very good match for the "next step" in learning OpenGL:
  - Setting up lighting effects in OpenGL
  - Controlling and positioning the "camera" for your 3D scene
  - Intercepting user activity and reacting to it
- Relevant Nate Robins tutors for these topics are *projection*, *transformation*, *lightmaterial*, and *lightposition*

# *glColor*()* is "Absolute" Color

- By "absolute" color we mean: independent of lighting
- In the real world, perceived color is highly dependent on the lighting environment
  - red object under white light looks red
  - cyan object under green light looks green
  - yellow object under red light looks — gasp! — red
  - blue object under cyan light looks blue
  - red object under blue light looks black…etc.
- In OpenGL, we'll need *glMaterial*()* and *glLight*()*

# The OpenGL Light Model

- Based on, but not the same as real world lighting
  - Food for thought: why not?
- Light is broken up into three components:
  - *Ambient:* Light that is so scattered as to appear to be coming from all directions and going in all directions
  - *Diffuse:* Light coming from a specific direction
  - *Specular:* Light that is reflected back in a focused direction; affects the perception of "shininess"

- A *light source* emits light, defined in terms of these three component colors
  - A minimum of 8 light sources (`GL_LIGHT0` to `GL_LIGHT7`), and they can be turned on or off individually at any time
- A *material* absorbs or reflects light, again defined in terms of these three component colors
- When doing lighting in OpenGL, objects/vertices no longer use plain color; they are given a material
- Lighting (and therefore shading) in OpenGL is based on the interaction of light sources on materials, according to combinations of their respective ambient, diffuse, and specular components

# Setting Up a Lit Scene

- Define your model so that it captures the data that influences the 3D environment
  - light sources: colors, positions, directions
  - material settings: colors, other properties
- Translate your internal settings into OpenGL with:
  - `glEnable(GL_LIGHTING)` — activate lighting
  - `glEnable(GL_LIGHT0)` — turn on/off light sources
  - `glLight*()` — configure light sources

- Prepare your geometric model to interact properly with lighting
  - Normal vectors using `glNormal*()`
    - For now, suffice it to say that these control how light reflects off a polygon; we'll tackle these in more detail later in the course
    - The GLUT quickie shapes do this for you already; if you build your own objects, you'll need to do this yourself
  - Ambient, diffuse, and specular material properties using `glMaterial*()`

# Material Details

```
void glMaterialf (GLenum face, GLenum pname, GLfloat param);
void glMaterialfv (GLenum face, GLenum pname, const GLfloat *params);
void glMateriali (GLenum face, GLenum pname, GLint param);
void glMaterialiv (GLenum face, GLenum pname, const GLint *params);
```

*Which side of the current face?*
GL_FRONT
GL_BACK
GL_FRONT_AND_BACK

"combo" properties

*Which material property?*
GL_AMBIENT
GL_DIFFUSE
GL_AMBIENT_AND_DIFFUSE
GL_SPECULAR
GL_SHININESS
GL_EMISSION

*Set the property to what value?*
RGBA most of the time; for GL_SHININESS, a single scalar value from 0 to 128 (128 being "shiniest")

# Light Details

```
void glLightf (GLenum light, GLenum pname, GLfloat param);
void glLightfv (GLenum light, GLenum pname, const GLfloat *params);
void glLighti (GLenum light, GLenum pname, GLint param);
void glLightiv (GLenum light, GLenum pname, const GLint *params);
```

*Which light?*
GL_LIGHT0 to GL_LIGHT7
…some implementations of OpenGL may have more

*Which light property?*
GL_AMBIENT
GL_DIFFUSE
GL_SPECULAR
GL_POSITION
GL_*_ATTENUATION
GL_SPOT_*

*Set the property to what value?*
RGBA most of the time; for GL_POSITION, an $(x, y, z, w)$ tuple, $w = 0$ implies direction instead of position; attenuation are scalars, and spotlight values are scalars except for GL_SPOT_DIRECTION

# Even More Details

- While *glLight*()* and *glMaterial*()* specify the parameters for OpenGL's lighting/shading calculations, there are also configurable options on *how* to do these calculations
- This tweaking can be done with *glLightModel*()* — check the red book for details
- In general, the defaults for the light model will suffice
- OpenGL also supports *custom shaders*, to *really* control how light interacts with your materials

# The OpenGL Camera

- Positioning the camera is pretty much a single function:

```
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY,
centerZ, upX, upY, upZ);
```

- *eye* is the camera's location; *center* is where the camera is looking; *up* is the camera's orientation
- That's all — call it before drawing and you're done
- Point to ponder: note the *glu* prefix — the "camera" is not a base OpenGL entity!

# Intercepting User Activity

There is pretty much a single consistent pattern for reading and responding to user activity with GLUT:

1. Register your handler functions by event type (mouse, keyboard, etc.)
2. Implement your handler functions to interpret the activity the way you wish
3. Once interpreted, call the "model" functions that change the state of your world
4. Request a repaint — `glutPostRedisplay()`

- Mouse functions:
  - `glutMouseFunc()` — mouse button activity
  - `glutMotionFunc()` — motion with button(s) down
  - `glutPassiveMotionFunc()` — motion without buttons pressed
- Keyboard functions:
  - `glutKeyboardFunc()` — conventional keys
  - `glutSpecialFunc()` — "special" keys (arrows, etc.)
- And of course, the all-important glutIdleFunc()
- And many more — check *glut.h*