Part 2 (now the fun really starts)

3D computer graphics with OpenGL

Karin Kosina (vka kyrah)

<review>

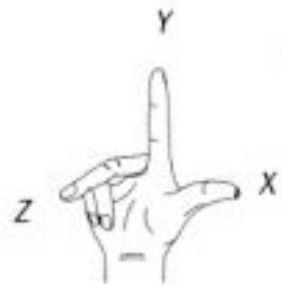
review #1:

the depth buffer

```
glEnable(GL_DEPTH_TEST);
```

review #2:

right-hand coordinate system?



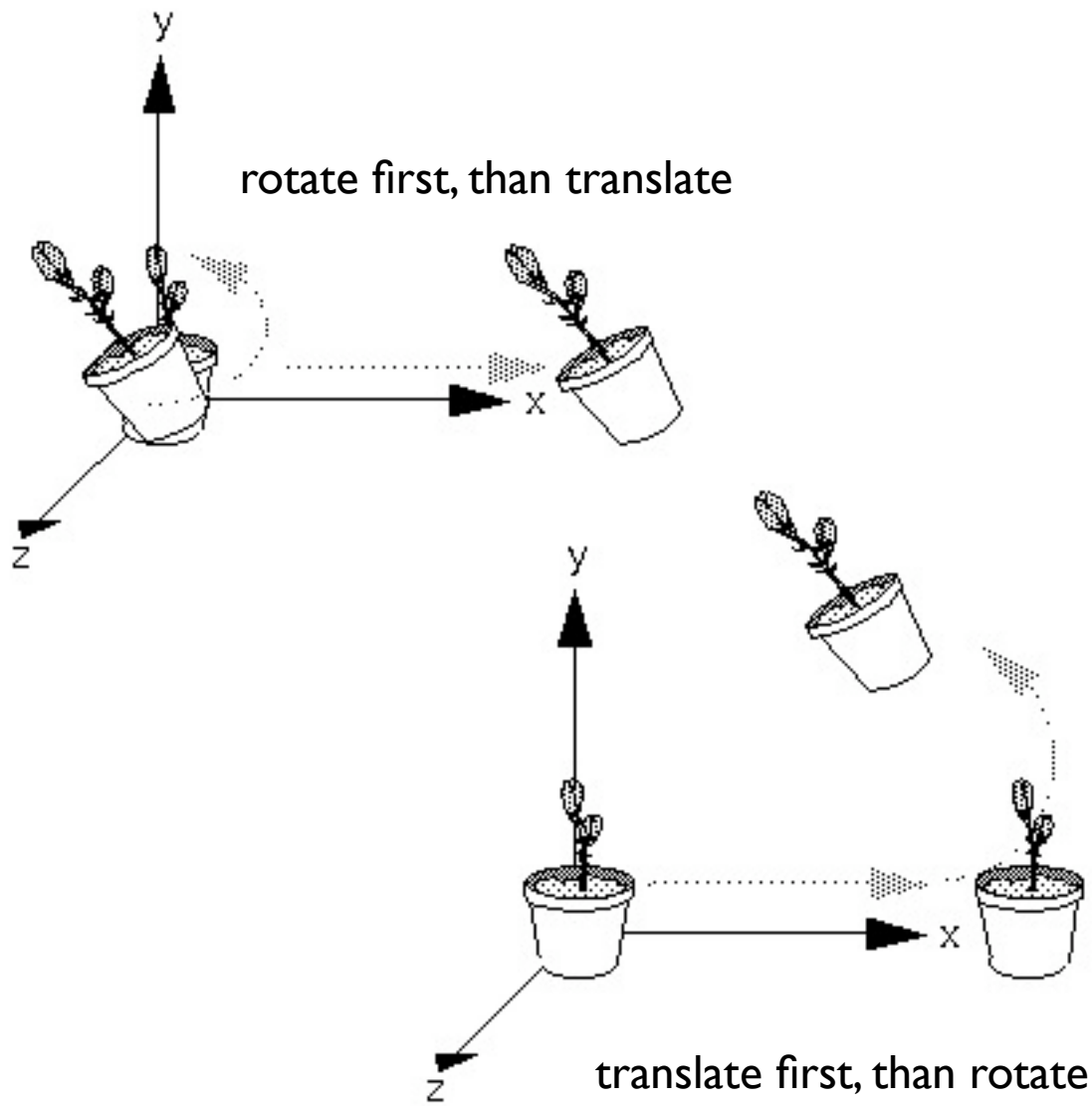
right-hand coordinate system

review #3:

transformations

order of transformations

- Matrix multiplication is not commutative.
  - The order of operations is important!
  - Example: Rotation and translation



</review>

lighting

what is light?

- particles or waves or neither or both...?
- it's complicated...

light in OpenGL consists of

- ambient light
  - scattered light (seemingly coming from all directions)
- diffuse light
  - light coming from one direction
  - scattered evenly when bouncing off a surface
- specular light ("shininess")
  - light coming from one direction
  - bounces off the surface in a preferred direction
- emitted light
  - originates from object – unaffected by light sources

it's all a fake

a lighting equation +
a shading model

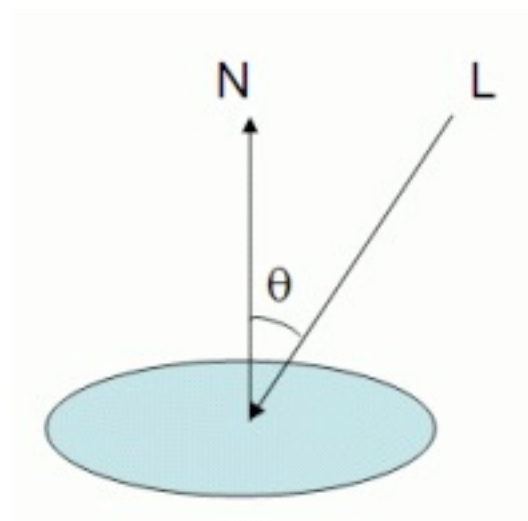
lambertian reflection

- **Lambert's cosine law:**

  The brightness of a diffusely radiating plane surface is proportional to the cosine of the angle formed by the line of sight and the normal to the surface.

- Same intensity regardless if the viewers position.

- Used for diffuse lighting component in OpenGL.



$$I_o = L_d * M_d * \cos(\theta)$$

I ... reflected intensity
L_d ... the light's diffuse intensity
M_d ... the material's diffuse coefficient

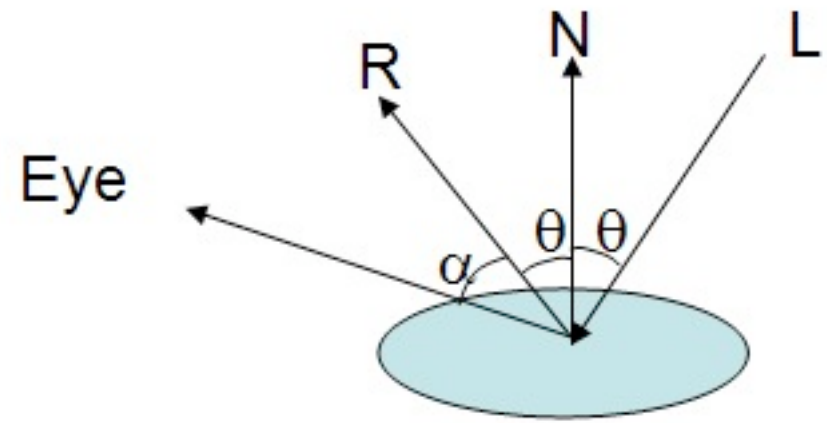
the phong model

- **The Phong model:**

  The specular component is proportional to the cosine between the light reflection vector and the eye vector.

- If the eye vector coincides with the reflection vector then we get the maximum specular intensity.

- OpenGL uses a simplification of the Phong model:
  the Blinn-Phong model



L is the vector from the light to the vertex being shaded.

R is the vector L mirror reflected on the surface.

N is the normal vector, and Eye is the vector from the vertex to the eye, or camera.

The specular component is proportional to the cosine of alpha.

the blinn-phong model

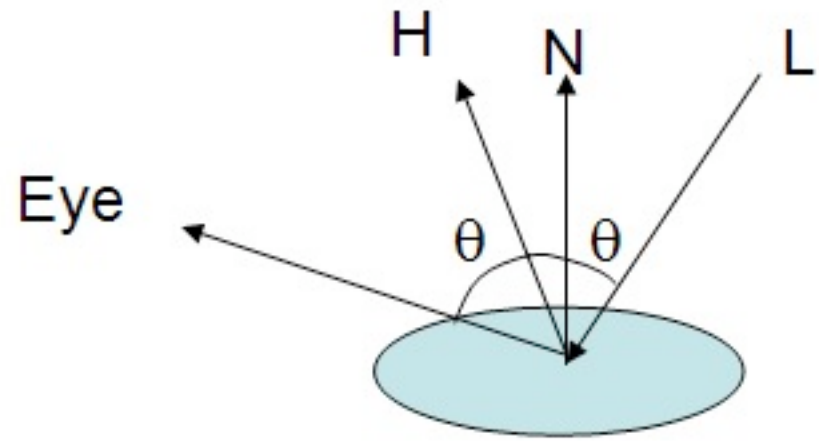
- **The Blinn-Phong model:**

  The specular component is based on the cosine of the angle between the half vector and the normal.

- Used for specular lighting component in OpenGL (the "bright spot").

$$Spec = (N \cdot H)^s * L_s * M_s$$

s ... shininess value
L_s ... the light's specular intensity
M_s ... the material's specular coefficient



L is the vector from the light to the vertex being shaded.

H is the half vector, a vector with a direction half-way between the eye vector and the light vector.

N is the normal vector, and Eye is the vector from the vertex to the eye, or camera.

The specular component is proportional to the cosine of alpha.

shading models

in OpenGL:

GL_FLAT

- flat shading
  - face normals (one color per polygon)

GL_SMOOTH

- gouraud shading
  - vertex normals (one colour per vertex, interpolated over the polygon along edges and scanlines)

not implemented!

- phong shading
  - interpolate vertex normals at each pixels (not just the colour values)

Flat shading vs. Gouraud shading

```
glShadeModel(GL_FLAT);
```

```
glShadeModel(GL_SMOOTH);
```

lighting step by step

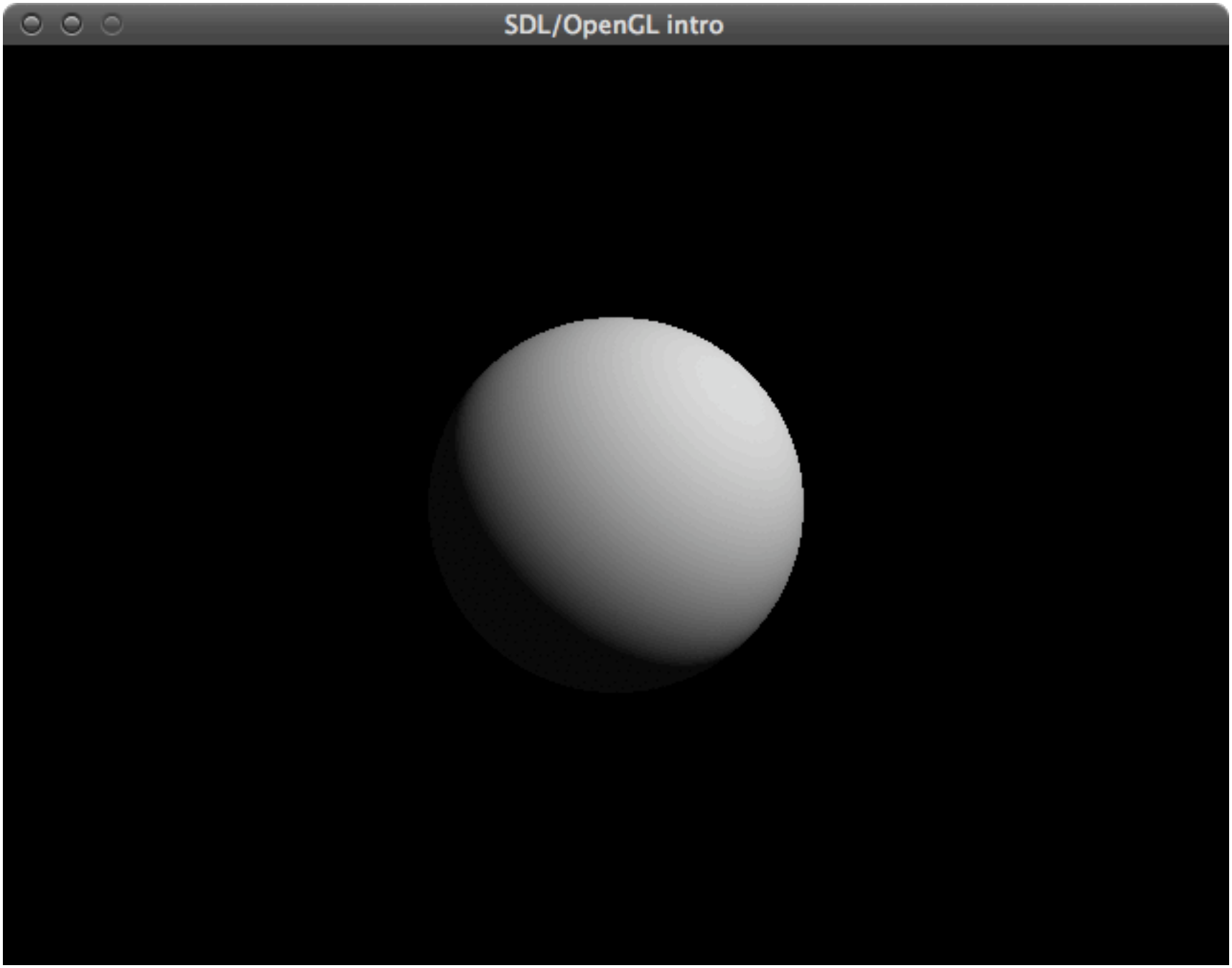
- Define normal vectors for each vertex
  - Normals determine the orientation of the object relative to the light source
- Create, select, and position one or more light sources.
- Select a lighting model.
- Define material properties for the objects in the scene.

lighting example

```
void myinit(int width, int height)
{
  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glShadeModel(GL_SMOOTH);

  // continue with initialisation code as before
  // ....
```

lighting example

```
void mydisplay()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

  GLUquadricObj* q = gluNewQuadric();
  gluQuadricDrawStyle (q, GLU_FILL);
  gluQuadricNormals   (q, GLU_SMOOTH);
  gluSphere (q, 1, 200, 200);
  gluDeleteQuadric (q);

  SDL_GL_SwapBuffers();
}
```



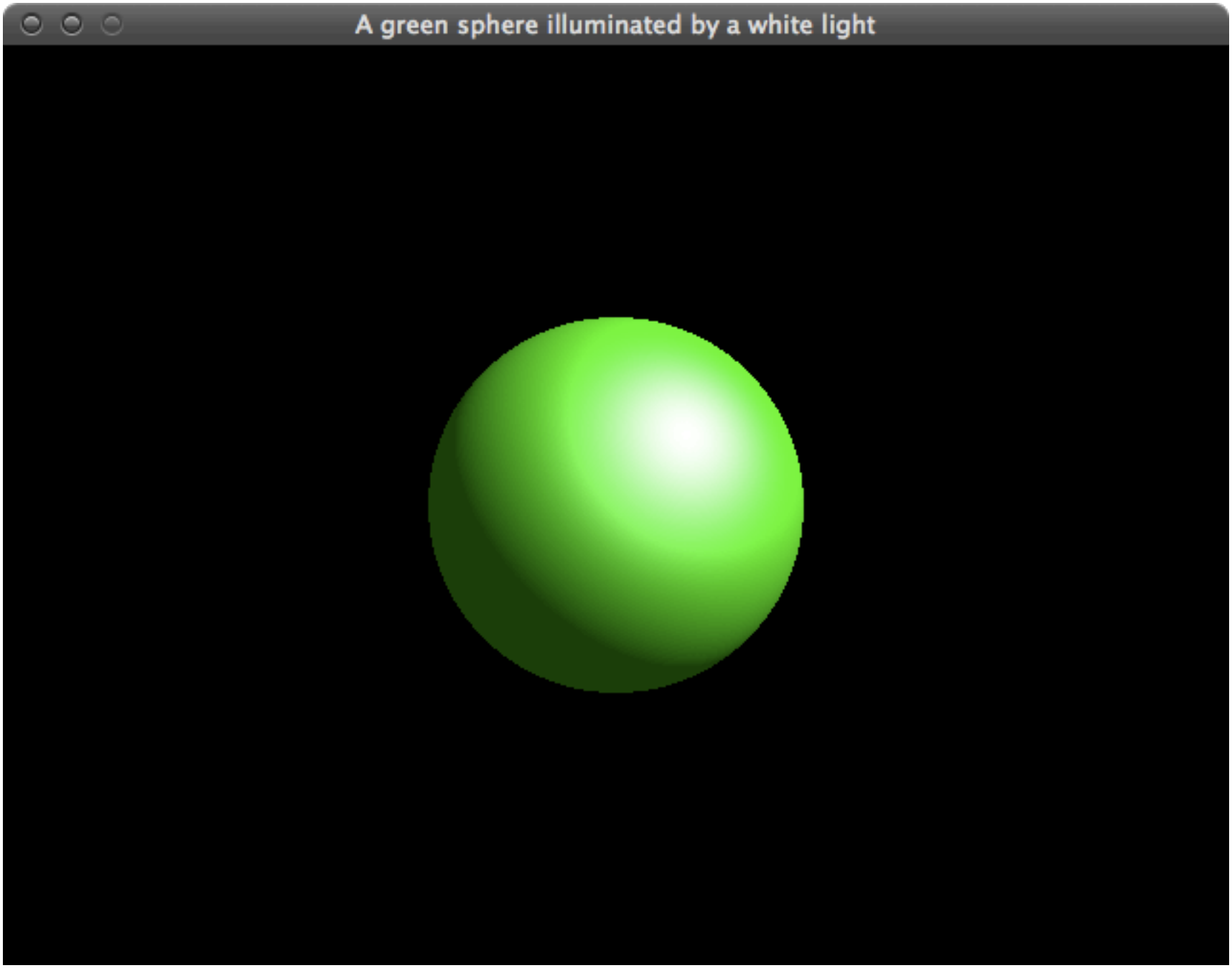
firstlight.cpp

material properties

- The color of a material depends on the percentage of incoming red, green, and blue light it reflects.
- Like lights, materials have different ambient, diffuse, and specular colors.
  - Material colors determine reflectance of the light component
  - Ambient and diffuse reflectances define the color of the material (typically similar or identical)
  - Specular reflectance is usually white or gray

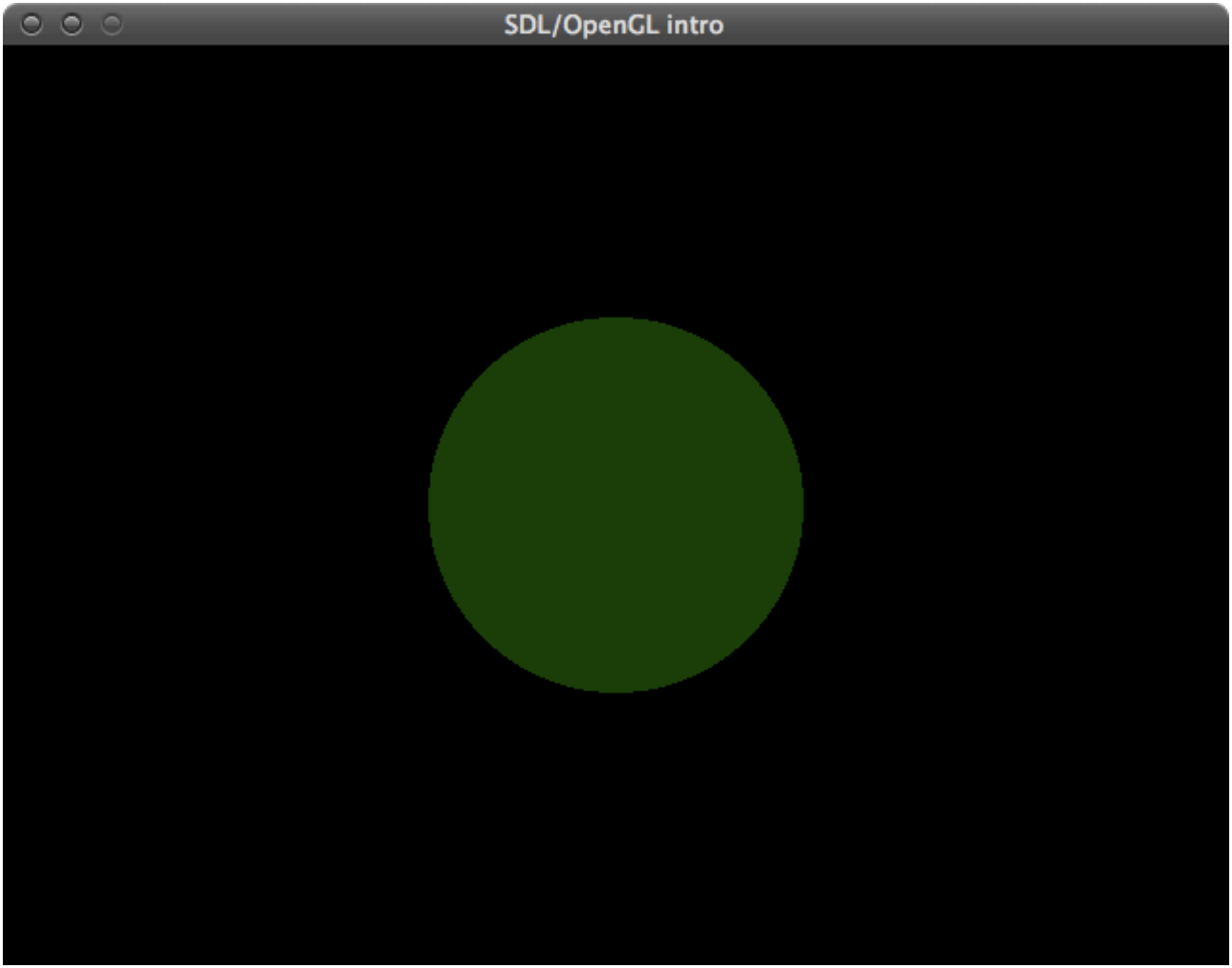
lighting example

```
void myinit(int width, int height)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 10.0 };
  GLfloat mat_ambient_and_diffuse[] = { 0.0, 1.0, 0.0, 1.0 };

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_and_diffuse);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_ambient_and_diffuse);

  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glShadeModel(GL_SMOOTH);

  // continue with initialisation code as before
  // ....
```

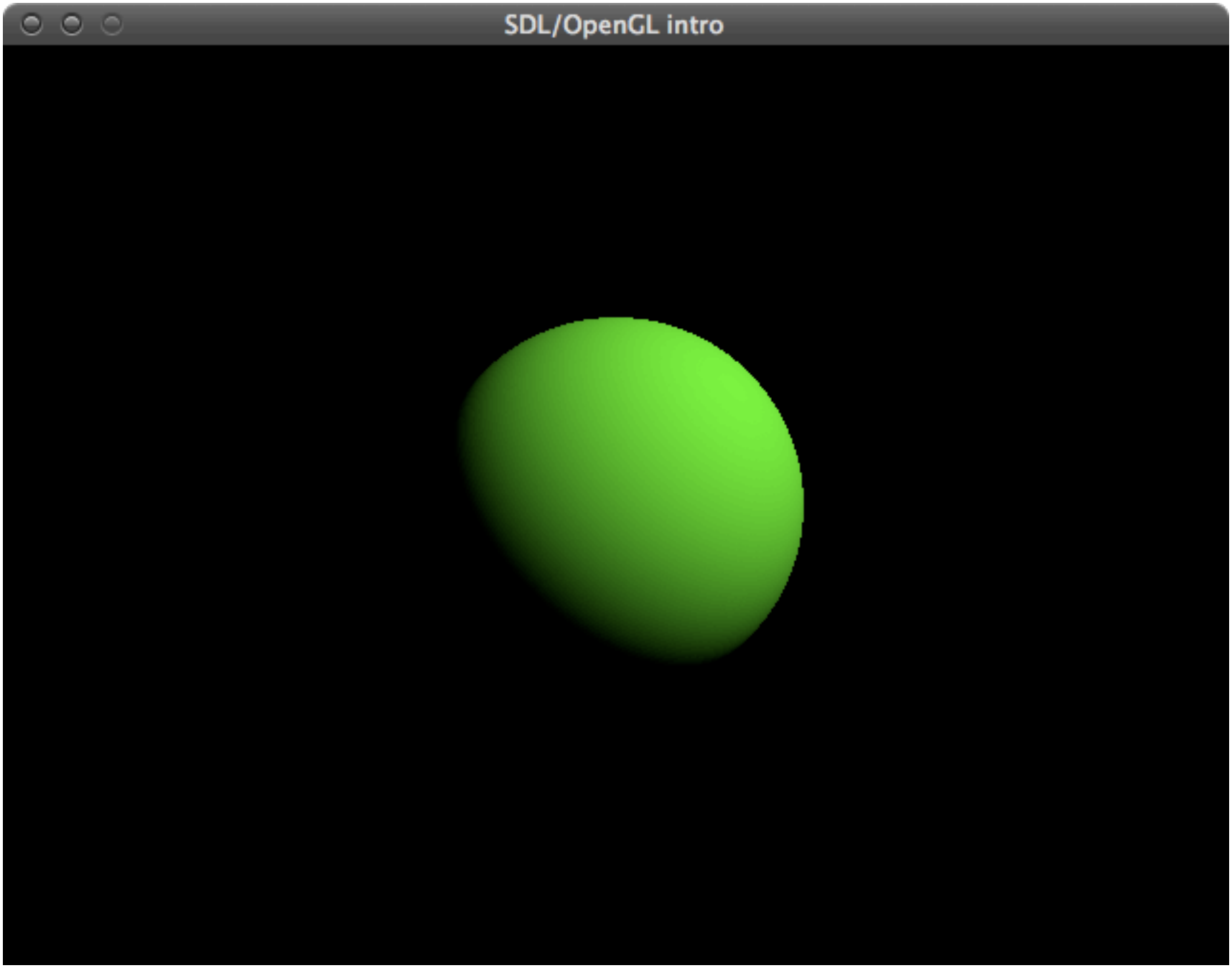


materialcolour.cpp

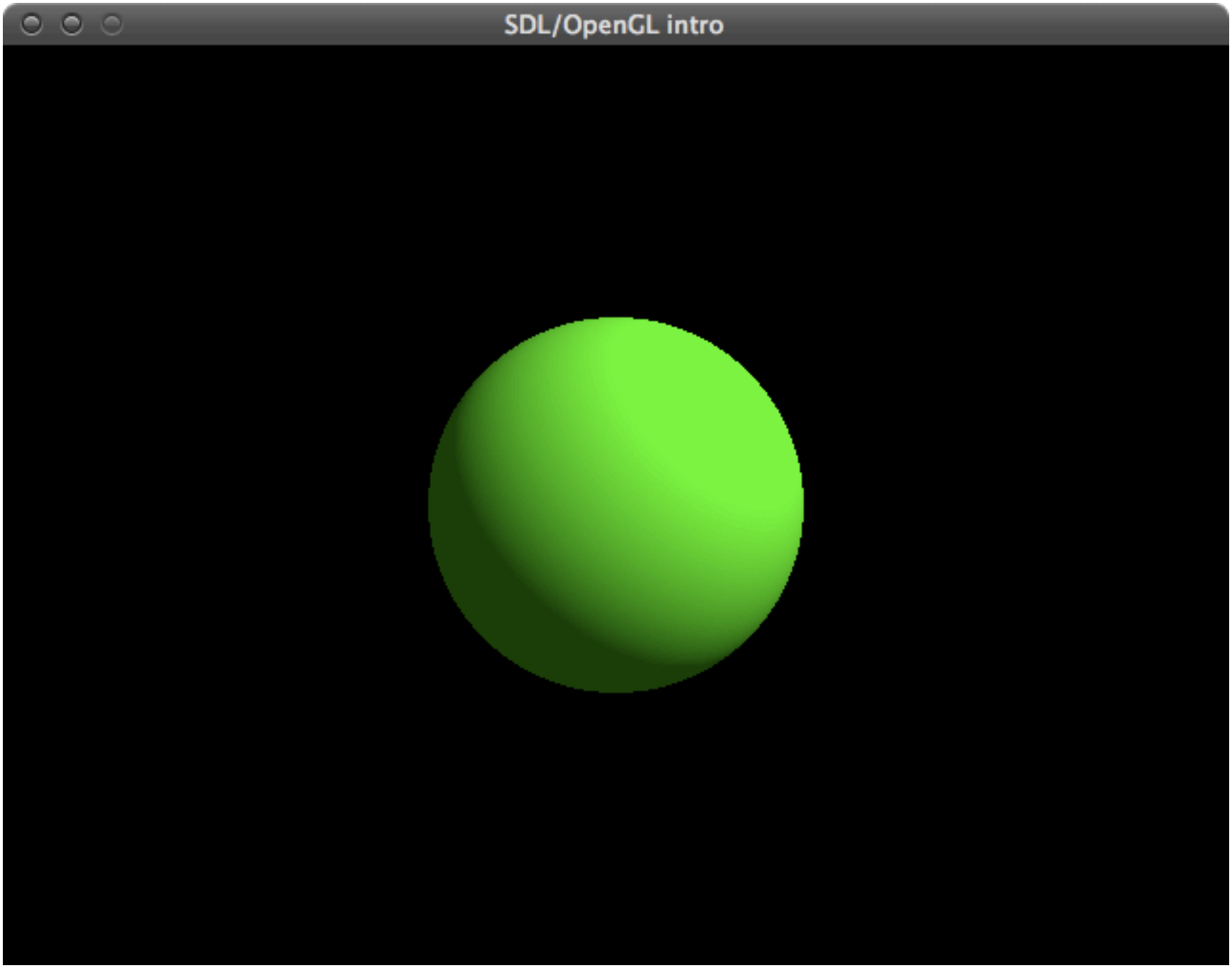
WTF are these strange
light components?



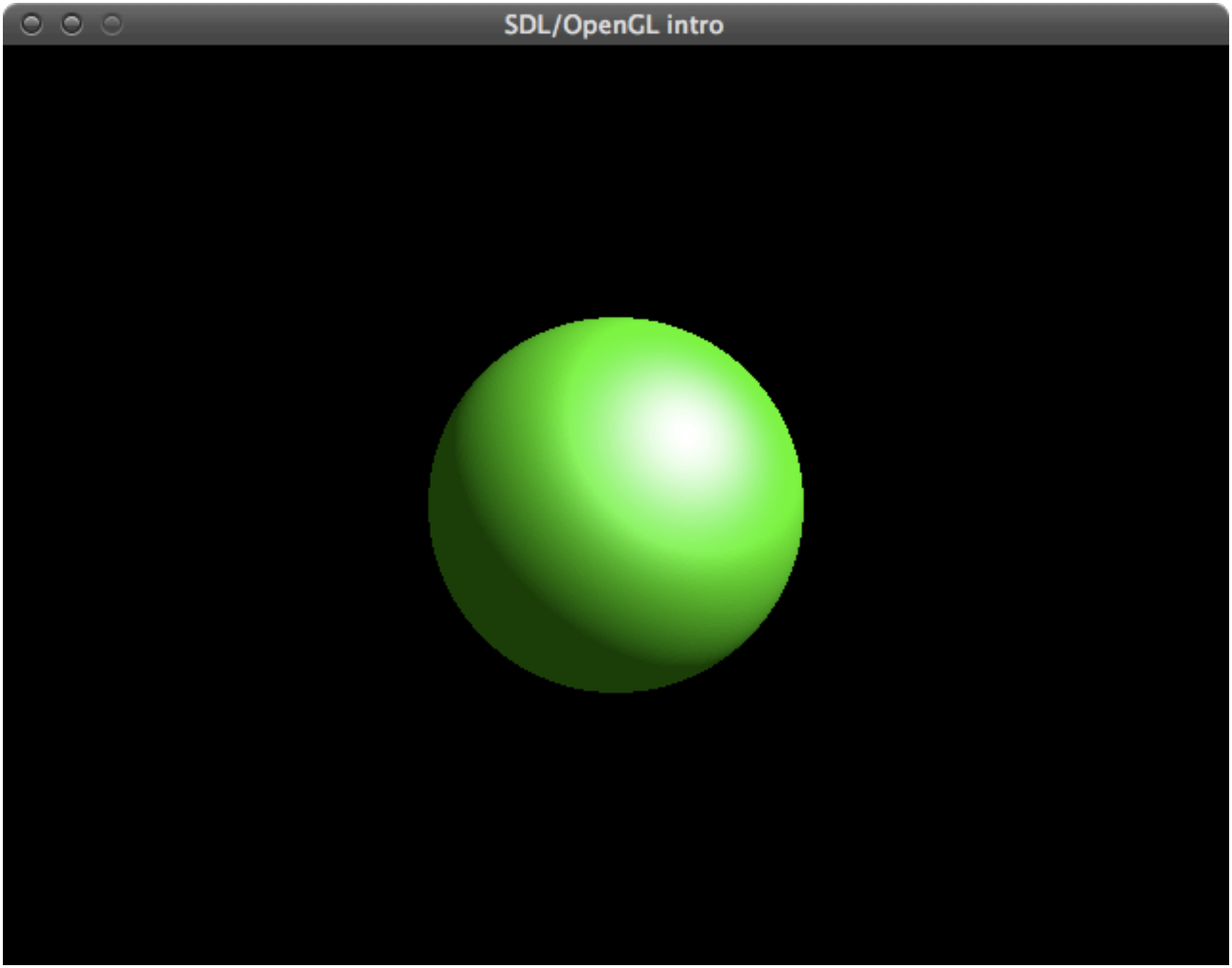
components.cpp *(ambient light only)*



components.cpp *(diffuse light only)*



components.cpp *(ambient and diffuse light)*



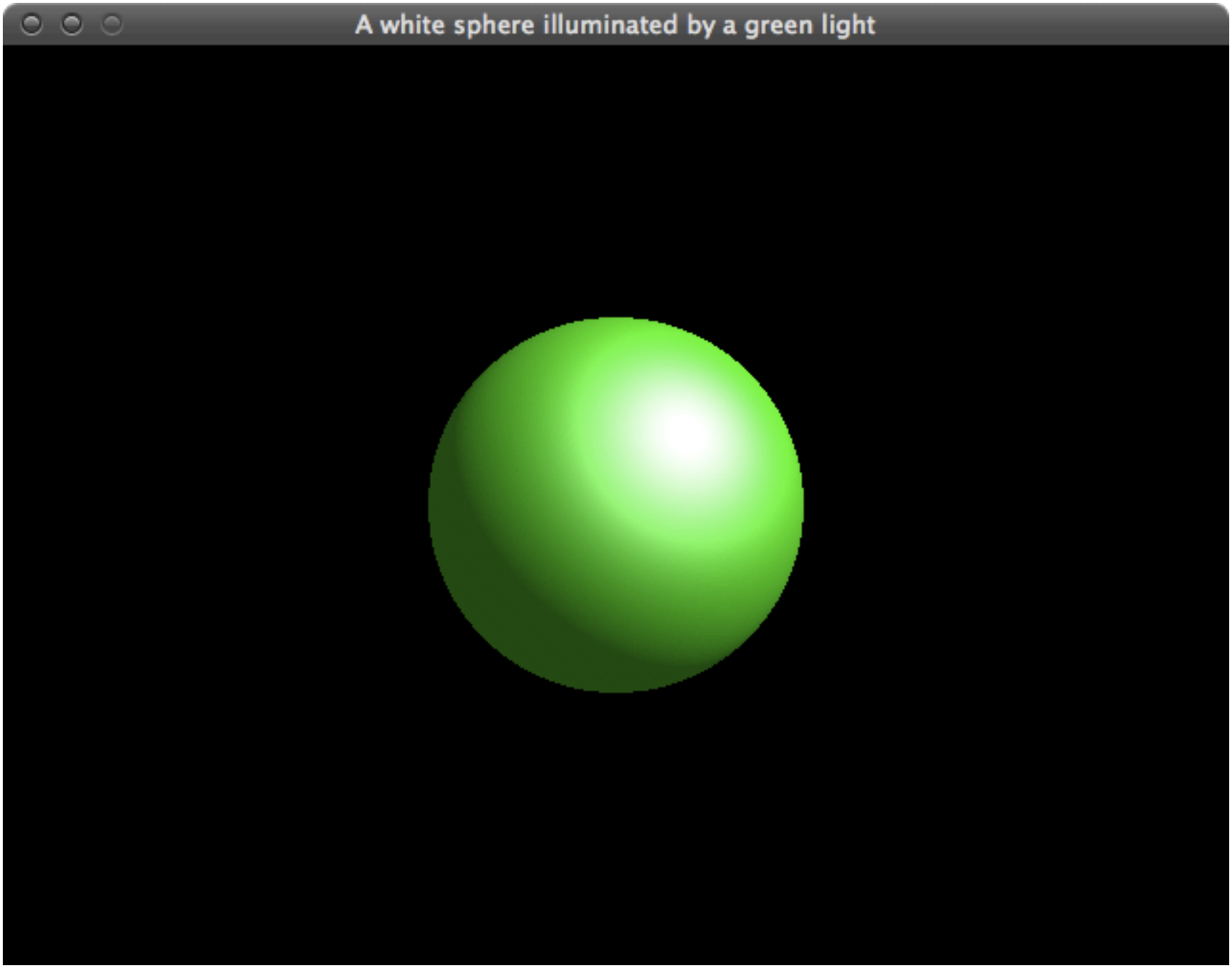
components.cpp *(ambient, diffuse and specular light)*

light source properties

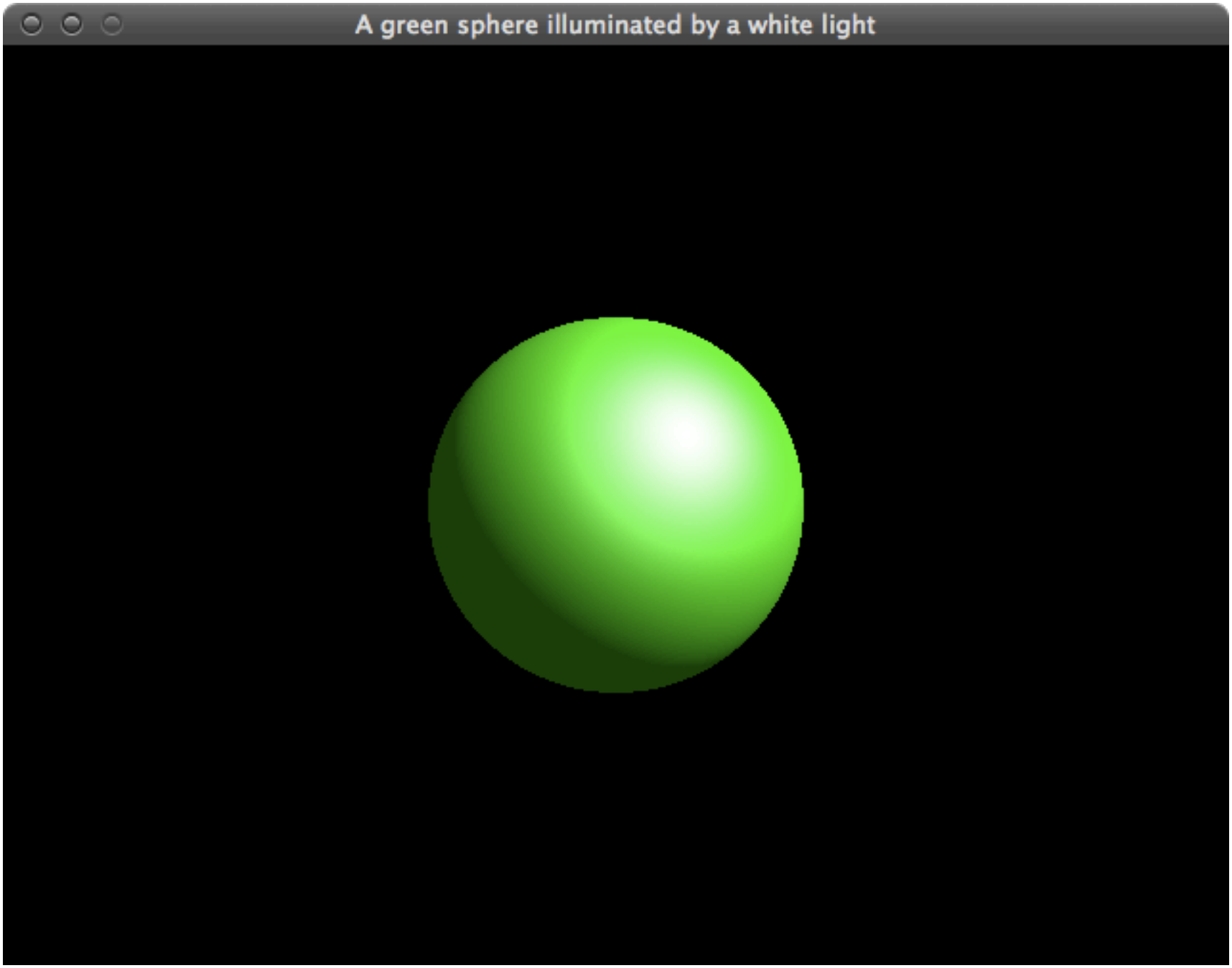
- Properties of light sources can be changed using **`glLight*()`** calls
- Available properties:
  - **`GL_AMBIENT`** (r, g, b, a – default: 0 0 0 1)
  - **`GL_DIFFUSE`** (r, g, b, a – default: 1 1 1 1)
  - **`GL_SPECULAR`** (r, g, b, a – default: 1 1 1 1)
  - **`GL_POSITION`** (x, y, z, w position – default: 0 0 1 0)

coloured light example

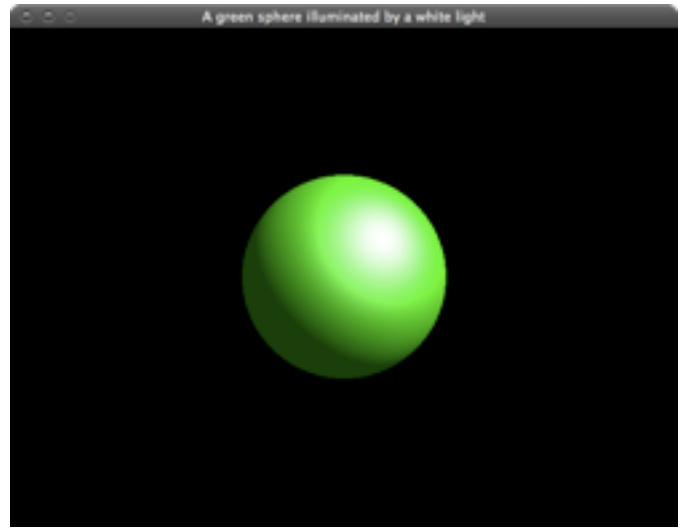
```
void myinit(int width, int height)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 10.0 };
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

  GLfloat light_ambient[] = { 0.0, 1.0, 0.0, 1.0 };
  GLfloat light_diffuse[] = { 0.0, 1.0, 0.0, 1.0 };
  GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  // ...
```



lightcolour.cpp



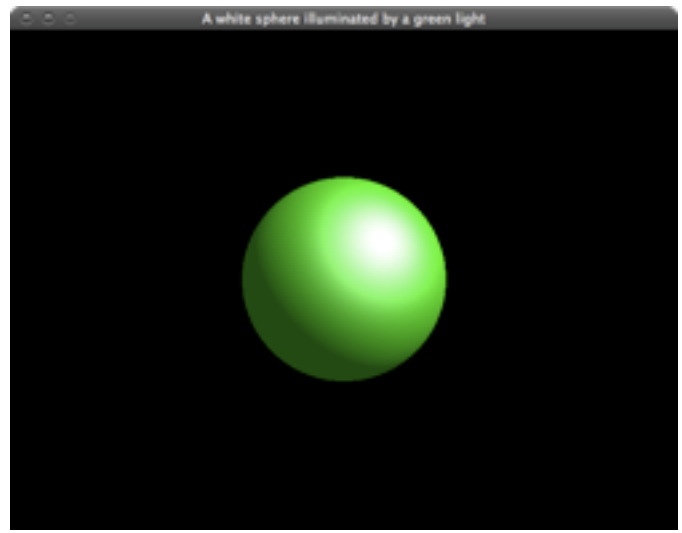
materialcolour.cpp



A green sphere illuminated by a white light

ALMOST same RESULT

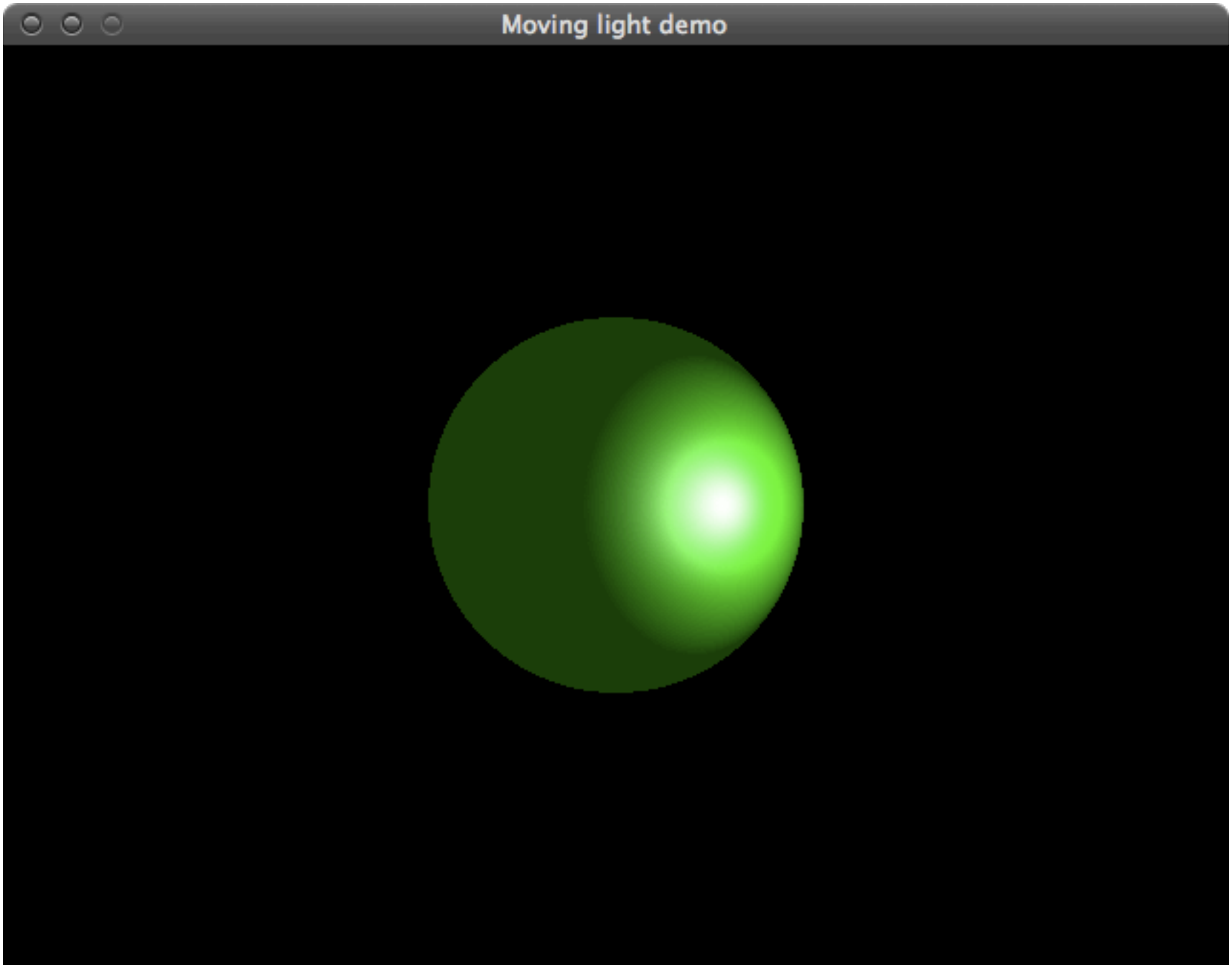
A white sphere illuminated by a green light



moving the light

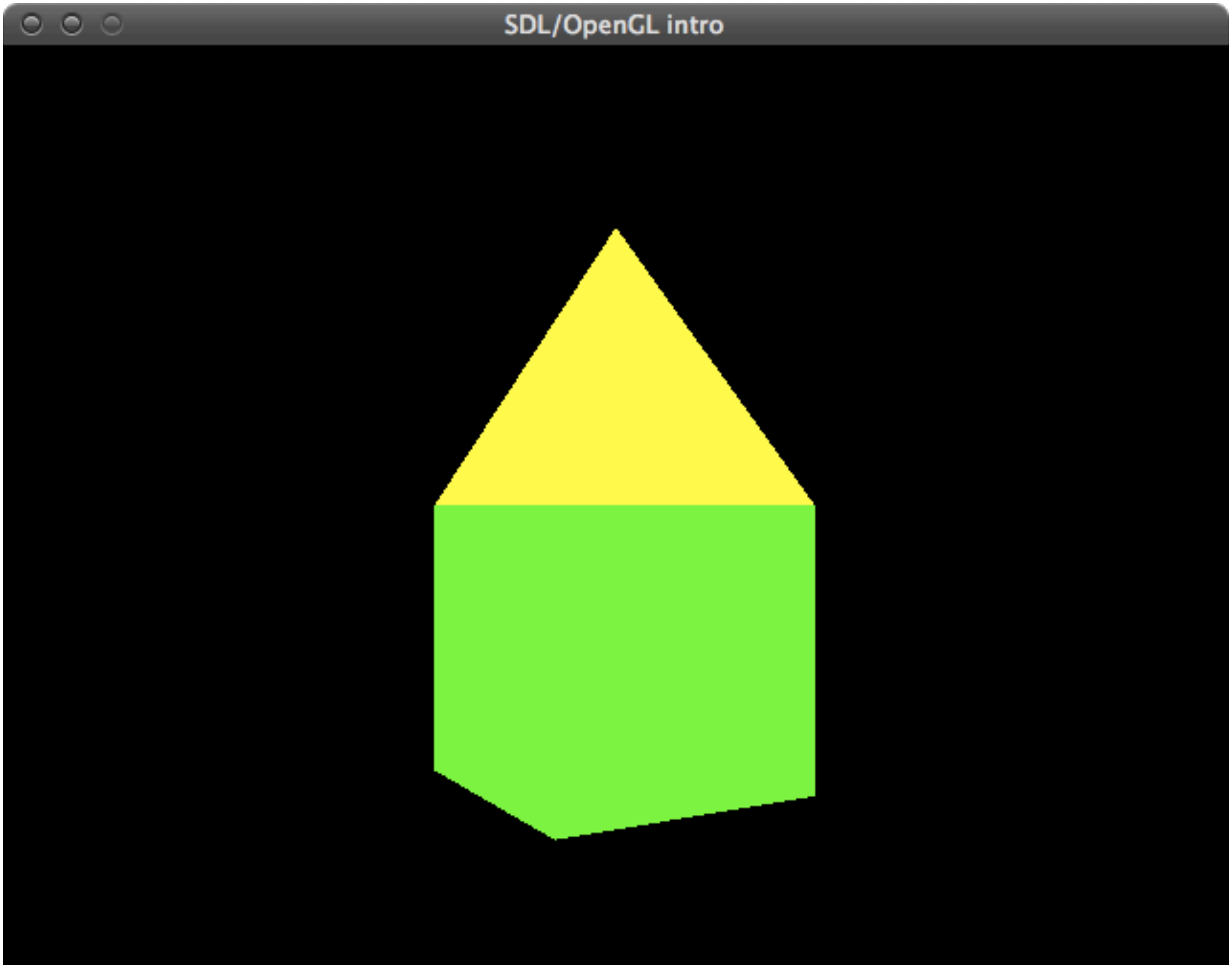
- Lights are influenced by the modelview matrix like any other object
- Translating the light relative to a stationary object?
  - Change model transform to specify the light position
  - Set light position after this
- Something like this:

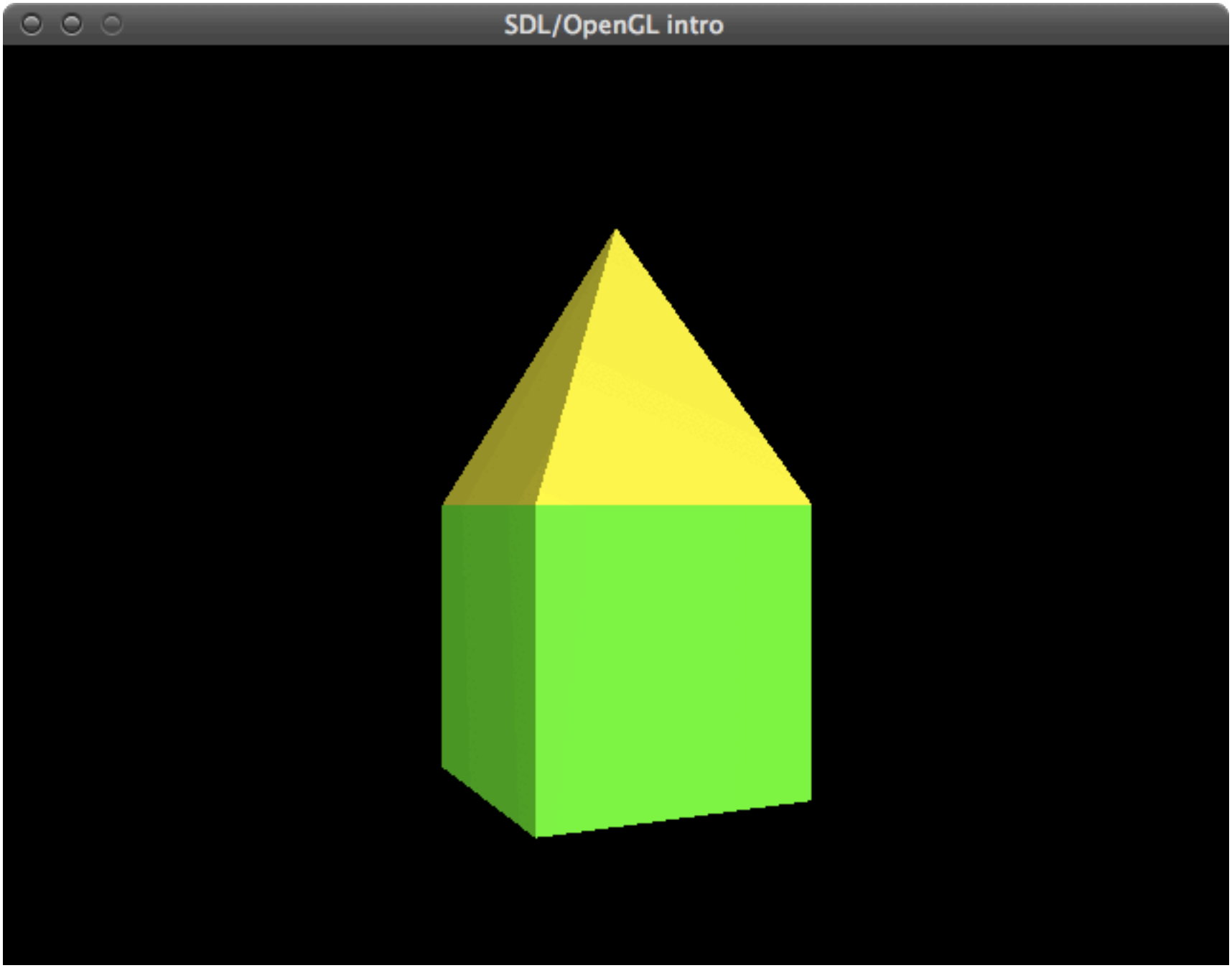
```
glPushMatrix ();
 glRotatef ((float) spin, 0.0, 1.0, 0.0);
 glLightfv (GL_LIGHT0, GL_POSITION, light_position);
glPopMatrix ();
drawScene();
```



movinglight.cpp

now add lighting to our
3D example


SDL/OpenGL intro


SDL/OpenGL intro

what you need

- a light source
- glMaterial instead of glColor
- normal vectors
  - faces must be defined in counter-clockwise order
  - to test: glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);
  - normals should be unit length
    - either do normalisation yourself (recommended)
    - or let OpenGL do it for you:
      glEnable(GL_NORMALIZE);