Computer Graphics

Karin Kosina (vka kyrah)

Part 2 (reloaded)

Computer Graphics

Karin Kosina (vka kyrah)

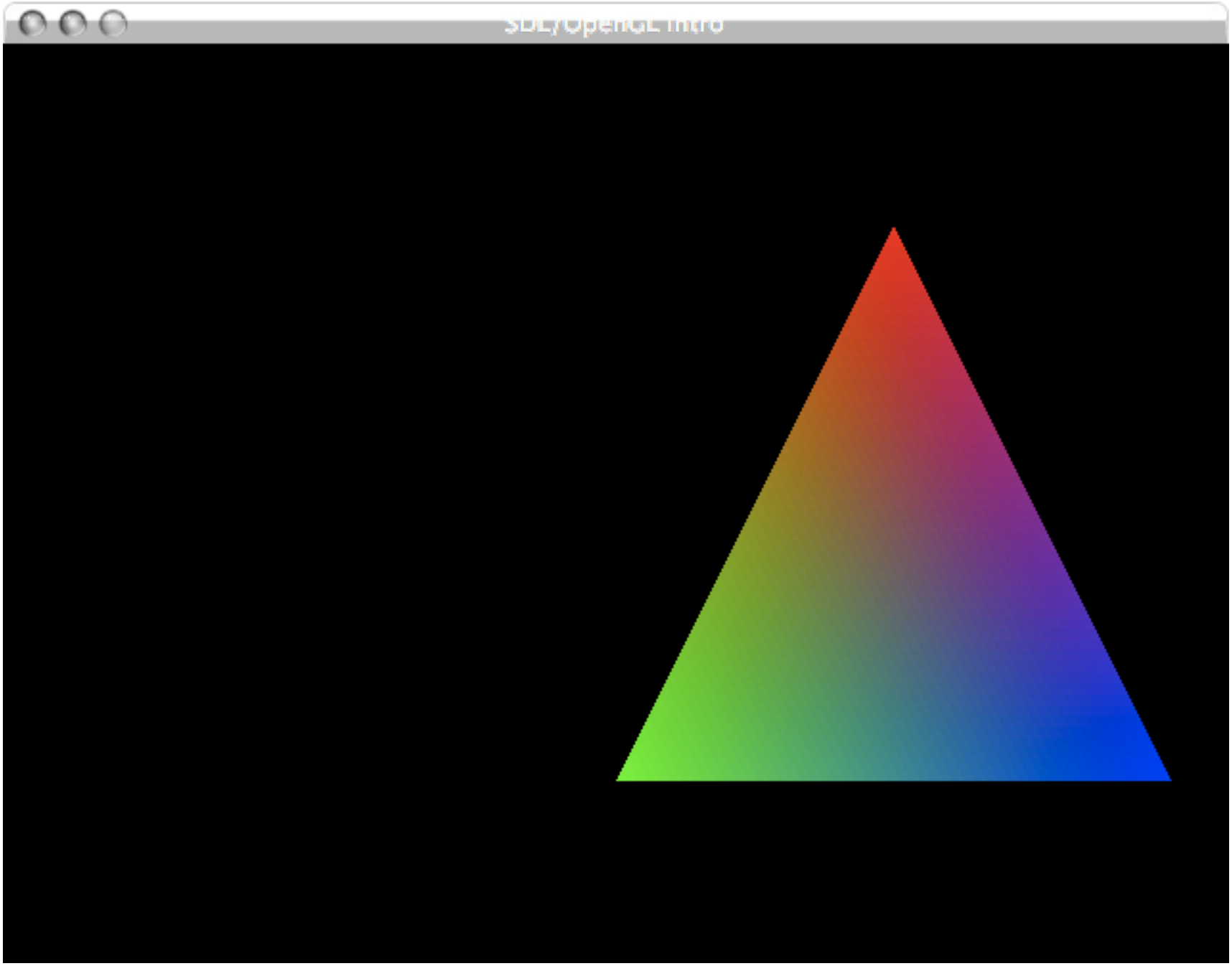
Review of the first workshop

OpenGL

- a platform-independent API for 2D and 3D graphics applications
- a standard, not a library
  - various implementions (e.g. by graphics card vendors) with varying degrees of optimisation
- Input: primitives (polygons, lines, points)
- Output: pixels
- low-level
- state-machine
- only does rendering
  - need additional framework for OS integration, image loading,...

SDL

- SDL is a free cross-platform multi-media development API
- abstraction for OS-dependent tasks
  - create window and rendering context
  - handle keyboard, mouse, and joystick events
  - audio
  - thread abstraction
  - ...
- see http://libsdl.org


SDL/OpenGL Intro

SDL framework

```
int main(int argc, char ** argv)
{
  int width = 640, height = 480;

  // Initialize SDL
  if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    fprintf(stderr, "Unable to init SDL: %s\n", SDL_GetError());
    return -1;
  }

  if (!SDL_SetVideoMode(width, height, 32, SDL_OPENGL)) {
    fprintf(stderr, "Unable set video mode: %s\n", SDL_GetError());
    SDL_Quit();
    return -1;
  }

  SDL_WM_SetCaption("SDL/OpenGL intro", NULL);  // window title
  myinit(width, height);  // initialize OpenGL

  // ... continued on next page
```

SDL framework

```
  // main application loop
  bool done = false;
  while (!done) {
    mydisplay();
    SDL_Event event;
    while (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) done = true;
      if (event.type == SDL_KEYDOWN) {
        switch(event.key.keysym.sym) {
        case SDLK_ESCAPE:
          done = true;
        }
      }
    }
  }

  SDL_Quit();
  return 0;
}
```

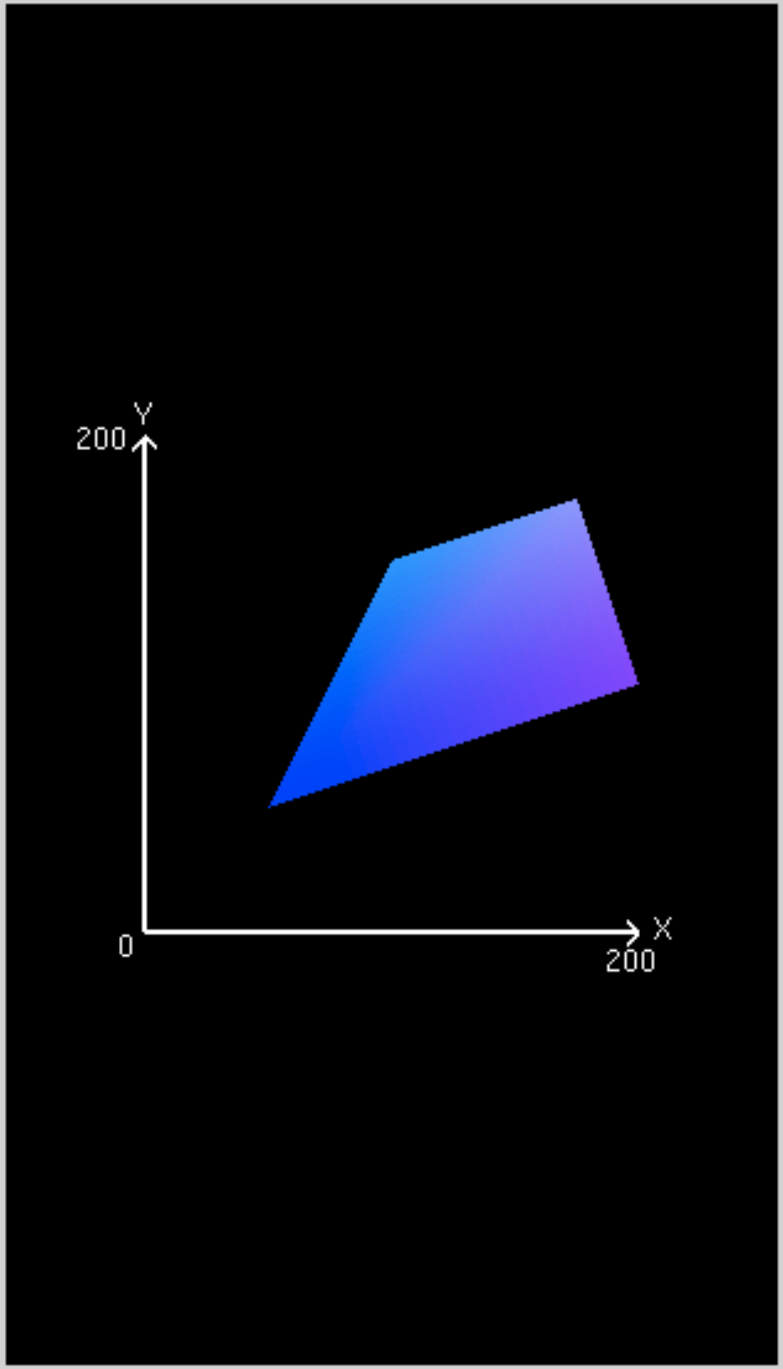
OpenGL initialisation

```
void myinit(int width, int height)
{
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0, (float)width/(float)height, 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0.0, 0.0, 4.0,  // eye
            0.0, 0.0, -1.0, // center
            0.0, 1.0, 0.0); // up

}
```

drawing

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

  glTranslatef(1.0f, 0.0f, 0.0f);

  glBegin(GL_TRIANGLES);
  glColor3f(1.0f, 0.0f, 0.0f);
  glVertex3f( 0.0f, 1.0f, 0.0f);
  glColor3f(0.0f, 0.0f, 1.0f);
  glVertex3f( 1.0f,-1.0f, 0.0f);
  glColor3f(0.0f, 1.0f, 0.0f);
  glVertex3f(-1.0f,-1.0f, 0.0f);
  glEnd();

  glPopMatrix();
  SDL_GL_SwapBuffers();
}
```

Screen-space view



Command manipulation window

```
glBegin (GL_TRIANGLE_FAN);
glColor3f   (0.00   , 0.00   , 1.00   );
glVertex2f (50.0   , 50.0   );
glColor3f   (0.00   , 0.50   , 1.00   );
glVertex2f (100.0 , 150.0 );
glColor3f   (0.50   , 0.50   , 1.00   );
glVertex2f (175.0 , 175.0 );
glColor3f   (0.50   , 0.00   , 1.00   );
glVertex2f (200.0 , 100.0 );
glEnd();
```

Click on the arguments and
move the mouse to modify values.

manipulating the matrix stack

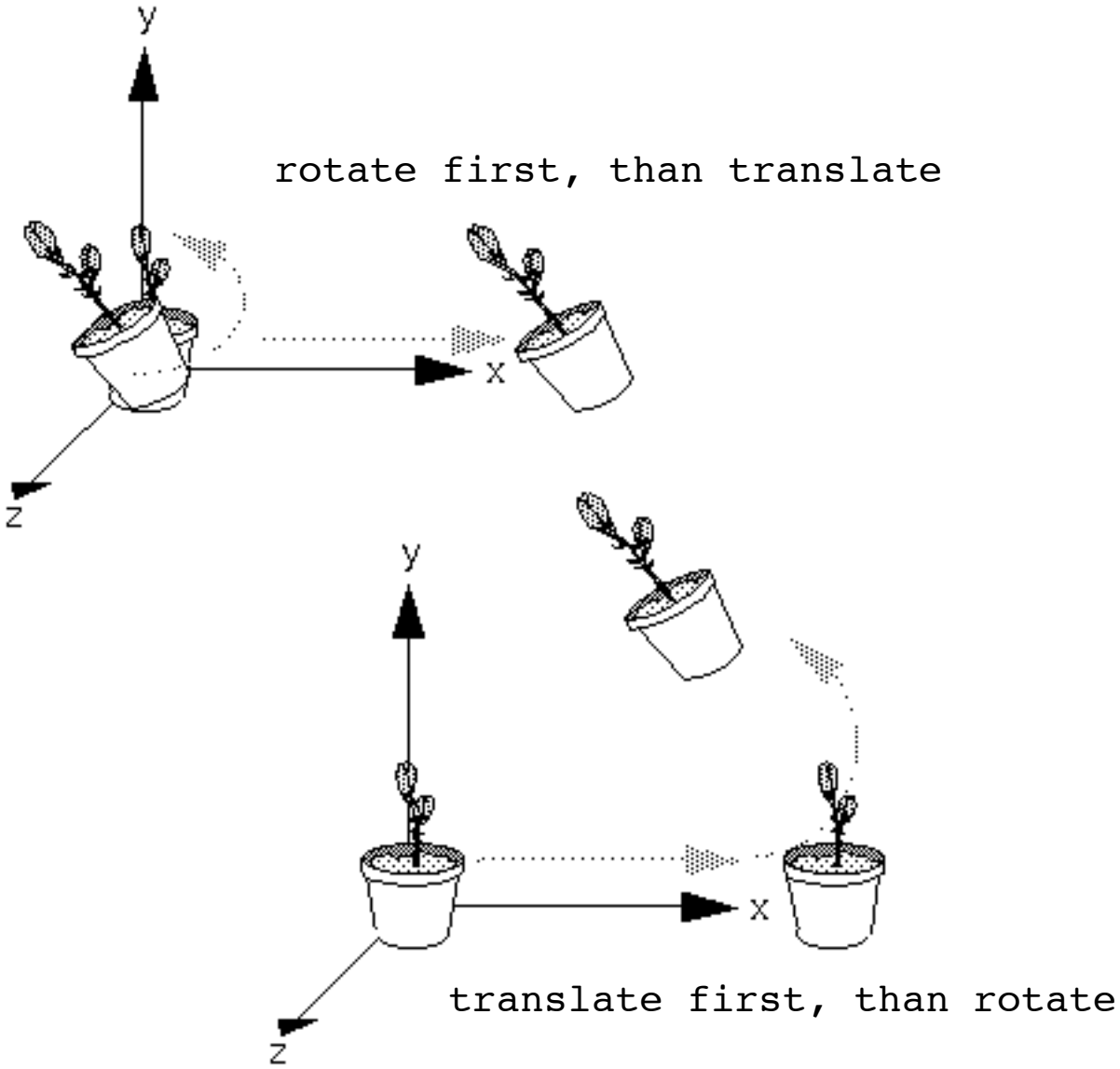
- glPushMatrix()
  - push all matrices in the current stack (determined by glMatrixMode()) down one level (the topmost matrix is duplicated)
- glPopMatrix()
  - pop the top matrix off the stack. The second matrix from the top of the stack becomes top, the contents of the popped matrix are destroyed.

model transformations in OpenGL

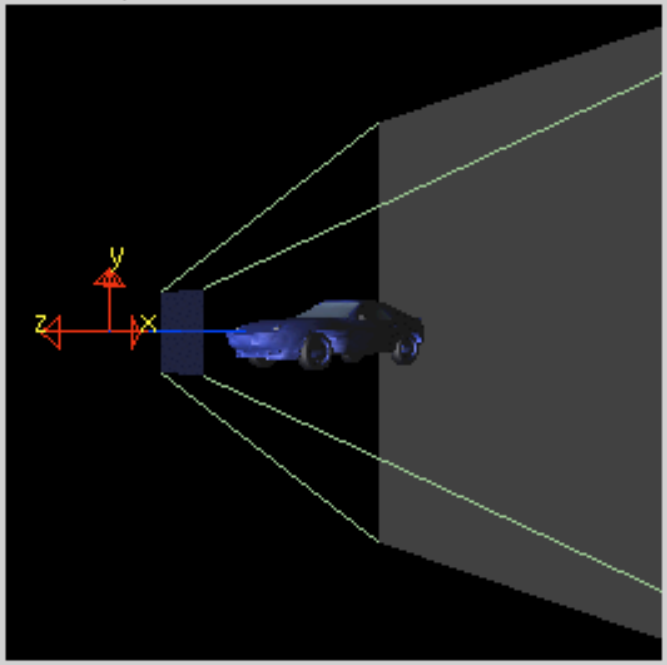
- 3 modeling transformations
  - glTranslate*()
  - glRotate*()
  - glScale*()
- Multiply a proper matrix for transform/rotate/scale to the current matrix and load the resulting matrix as current matrix.

order of transformations

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



World-space view

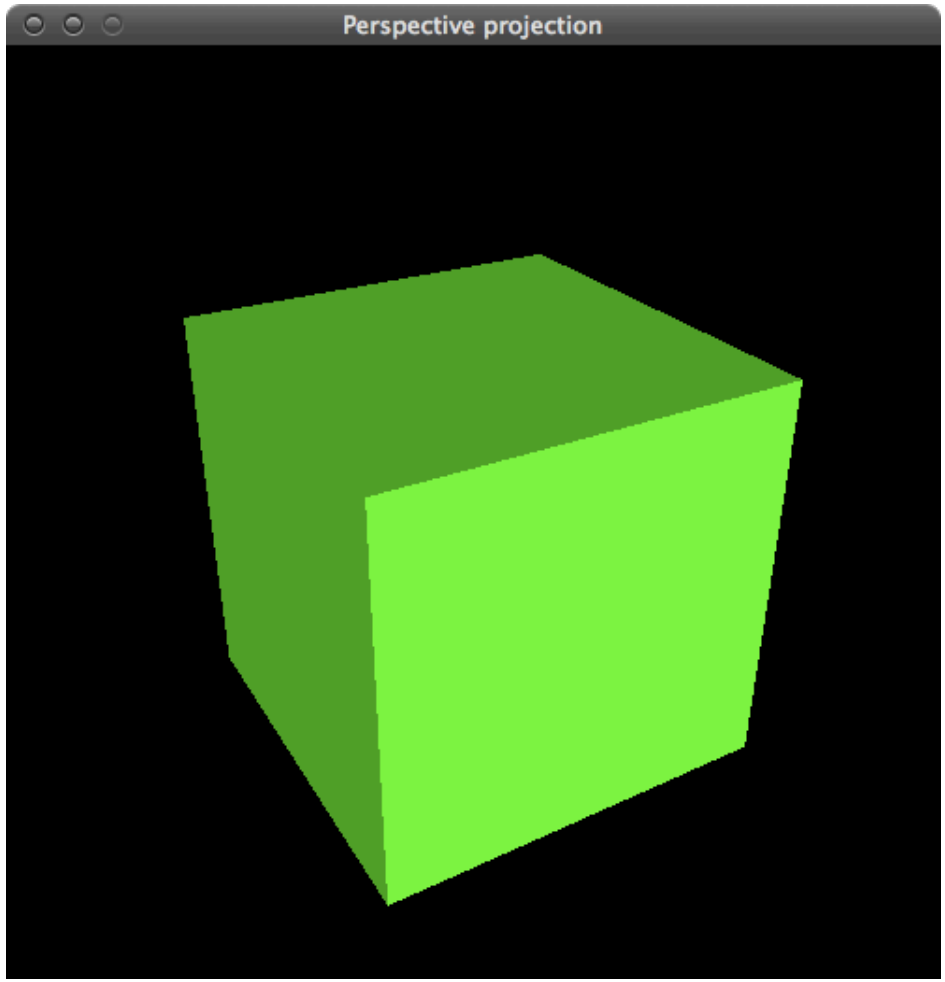


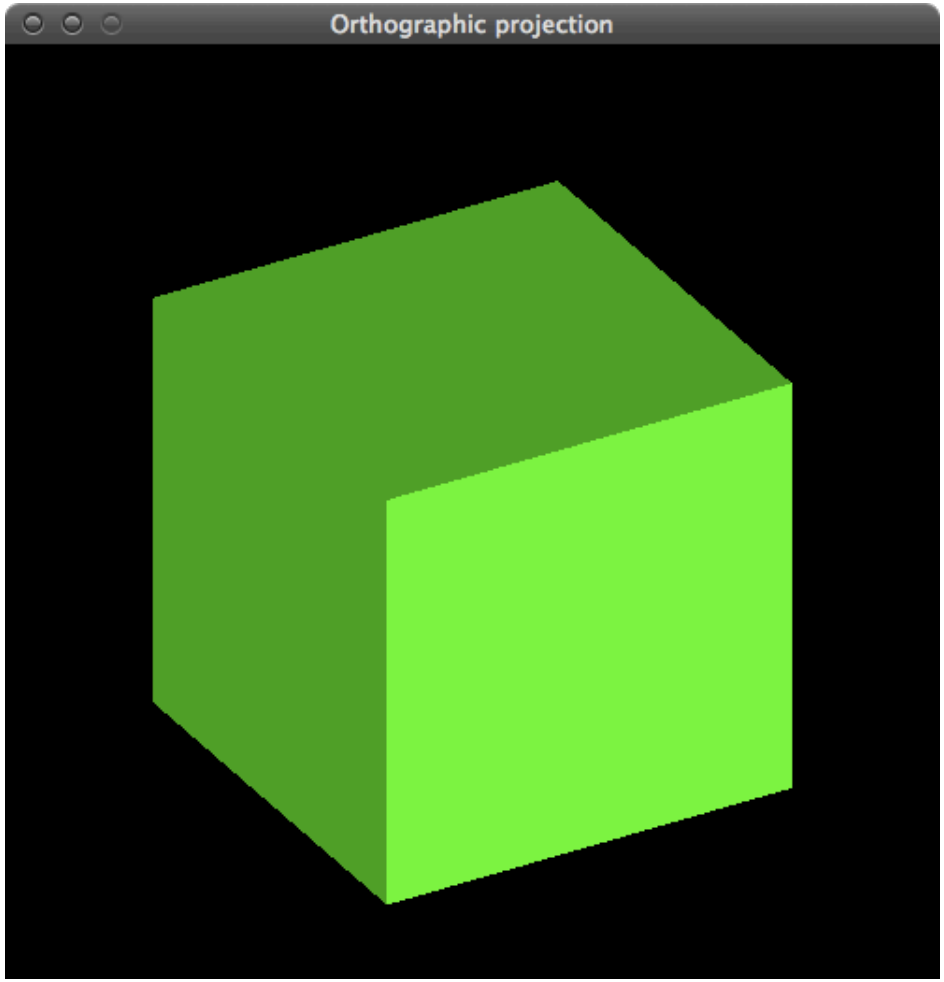
Screen-space view

Command manipulation window

```
glTranslatef( 0.00  , 0.00  , 0.00  );
   glRotatef( 0.0   , 0.00  , 1.00  , 0.00  );
    glScalef( 1.00  , 1.00  , 1.00  );
     glBegin( . . . );
      . . .
```

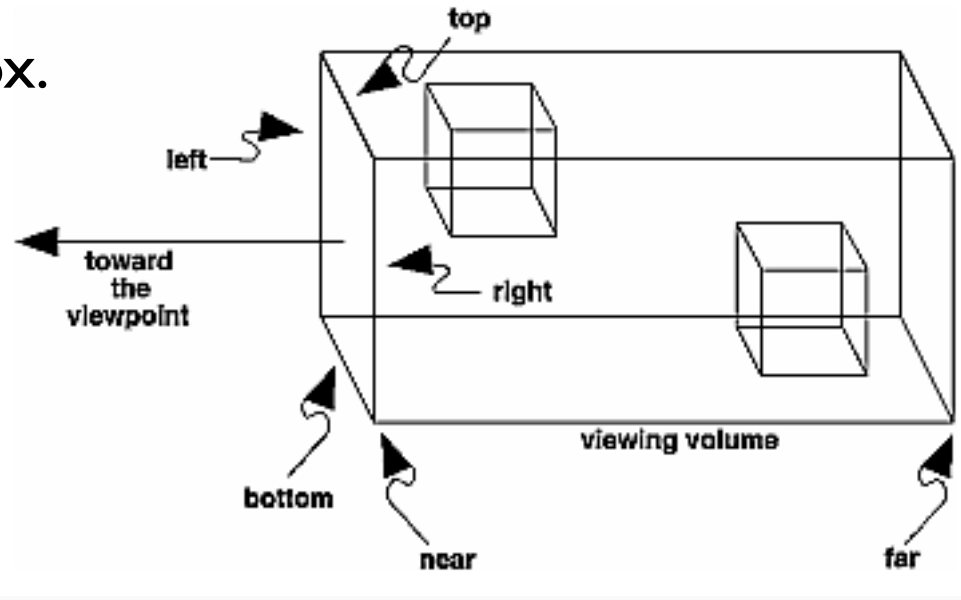
Click on the arguments and move the mouse to modify values.


Perspective projection


Orthographic projection

projection

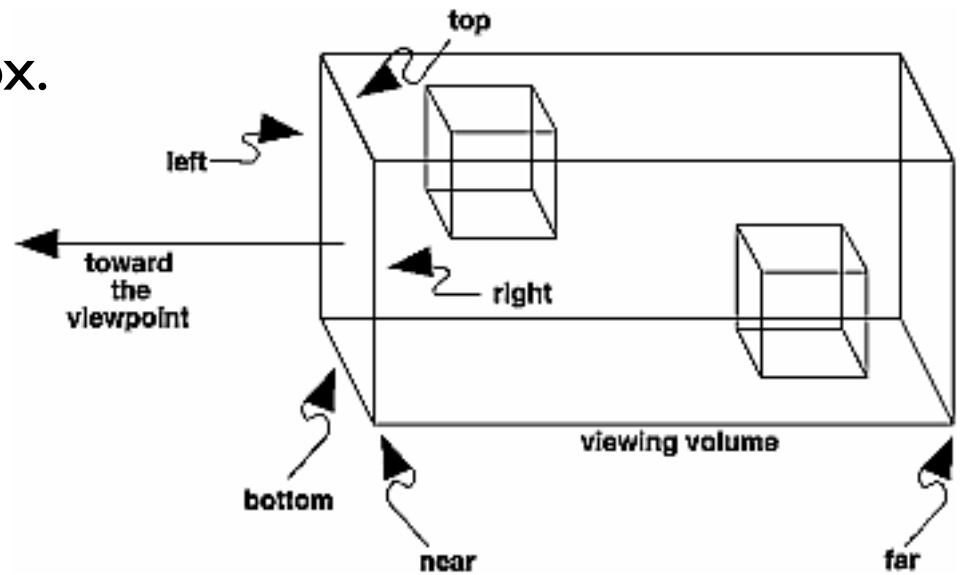
- Transformation of the view volume into a unit cube with extreme points at (-1, -1, -1) and (1, 1, 1).
- Two projection methods: orthographic vs. perpective projection
- Orthographic projection:
  - View volume is a rectangular box.
  - Parallel lines remain parallel after the transform.



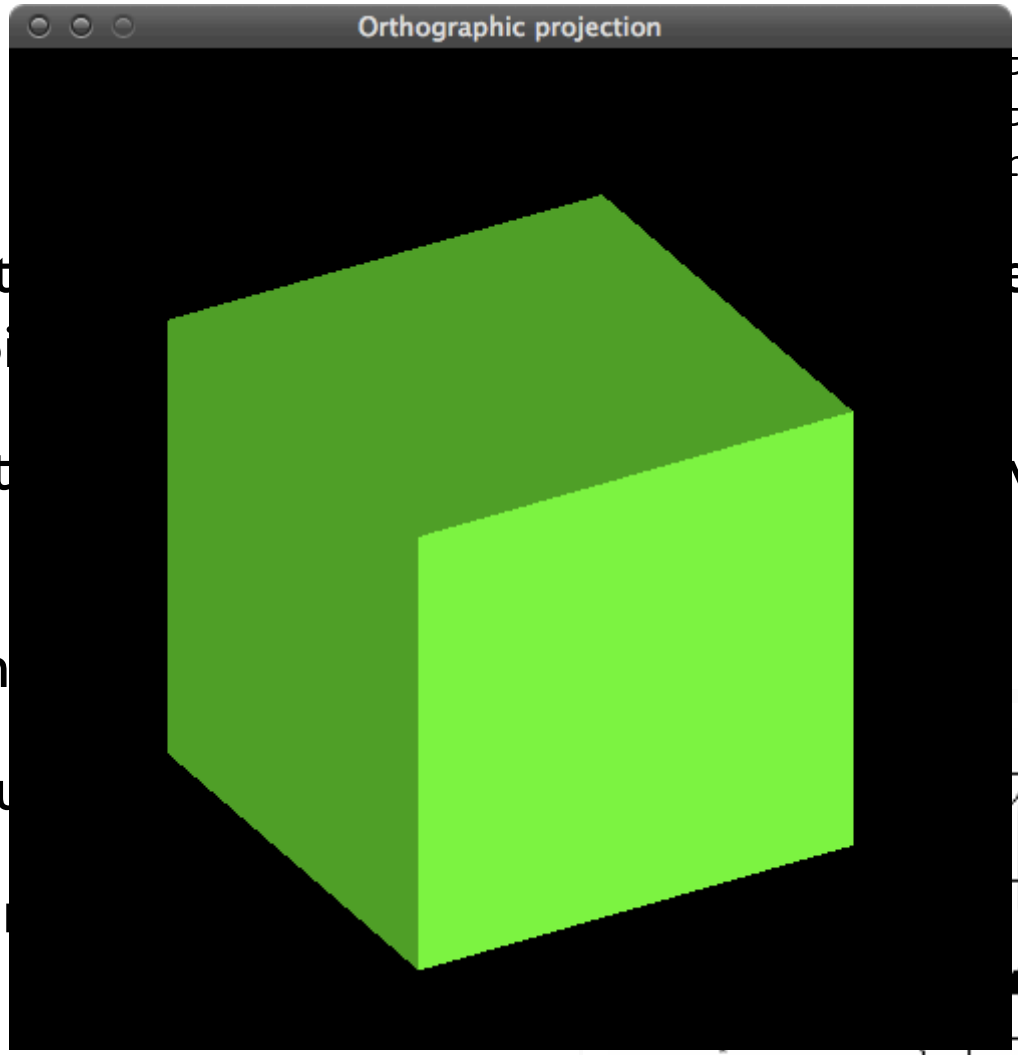
projection

```
glOrtho(float left, float right,
        float bottom, float top,
        float near, float far);
```

- Transformation of the view volume into a unit cube with extreme points at (-1, -1, -1) and (1, 1, 1).
- Two projection methods: orthographic vs. perpective projection
- Orthographic projection:
  - View volume is a rectangular box.
  - Parallel lines remain parallel after the transform.

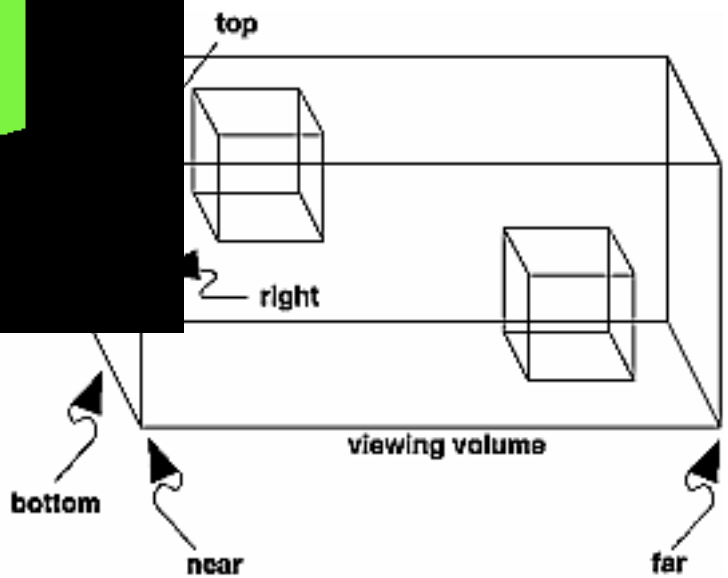


projection



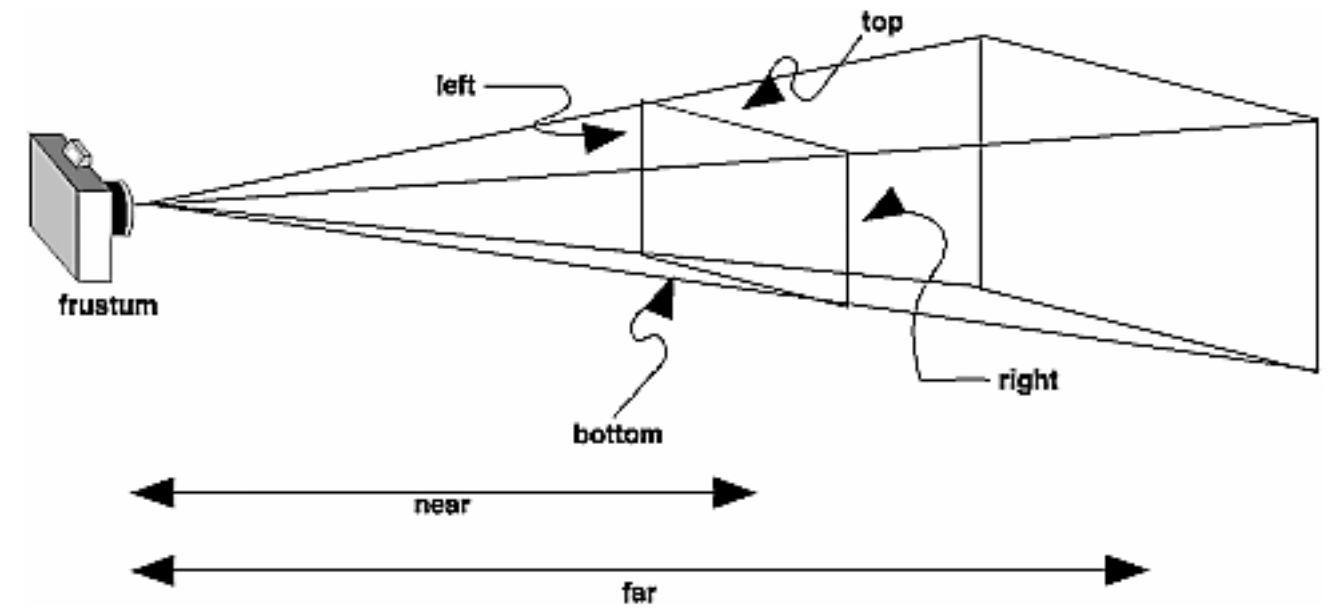

```
t, float right,
tom, float top,
r, float far);
```

- Transformat … e with extreme poi …
- Two project … ve projection
- Orthograph …
  - View volu …
  - Parallel li … after the …



projection

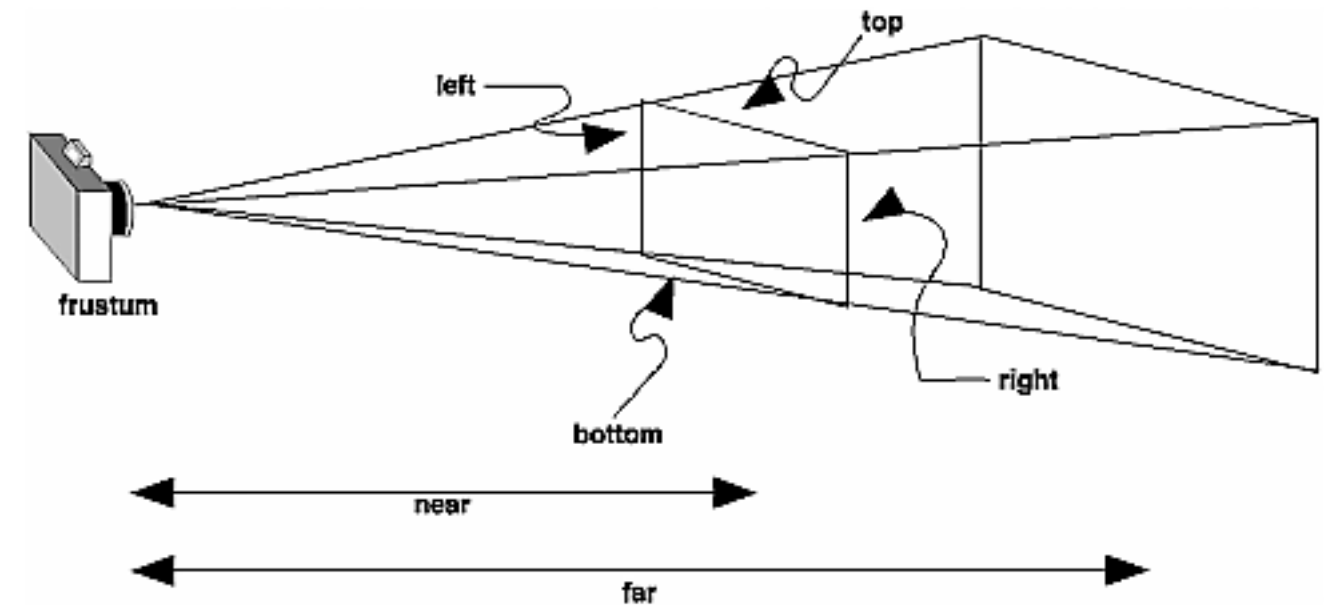
- Perpective projection:
  - The farther away an object lies from the camera, the smaller it appears after projection.
  - Parallel lines converge at the horizon.
  - View volume (called frustum) is a truncated pyramid with a rectangular ba



projection

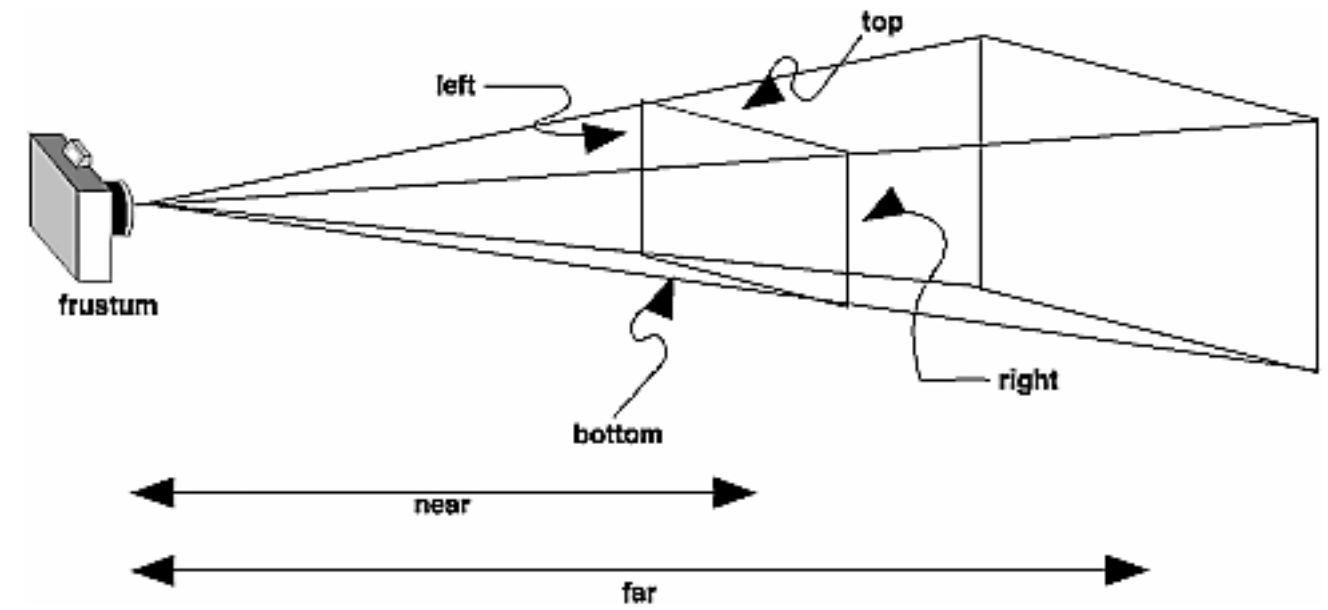
```
glFrustum(float left, float right,
          float bottom, float top,
          float near, float far);
```

- Perpective projection:
  - The farther away an object lies from the camera, the smaller it appears after projection.
  - Parallel lines converge at the horizon.
  - View volume (called frustum) is a truncated pyramid with a rectangular ba



projection

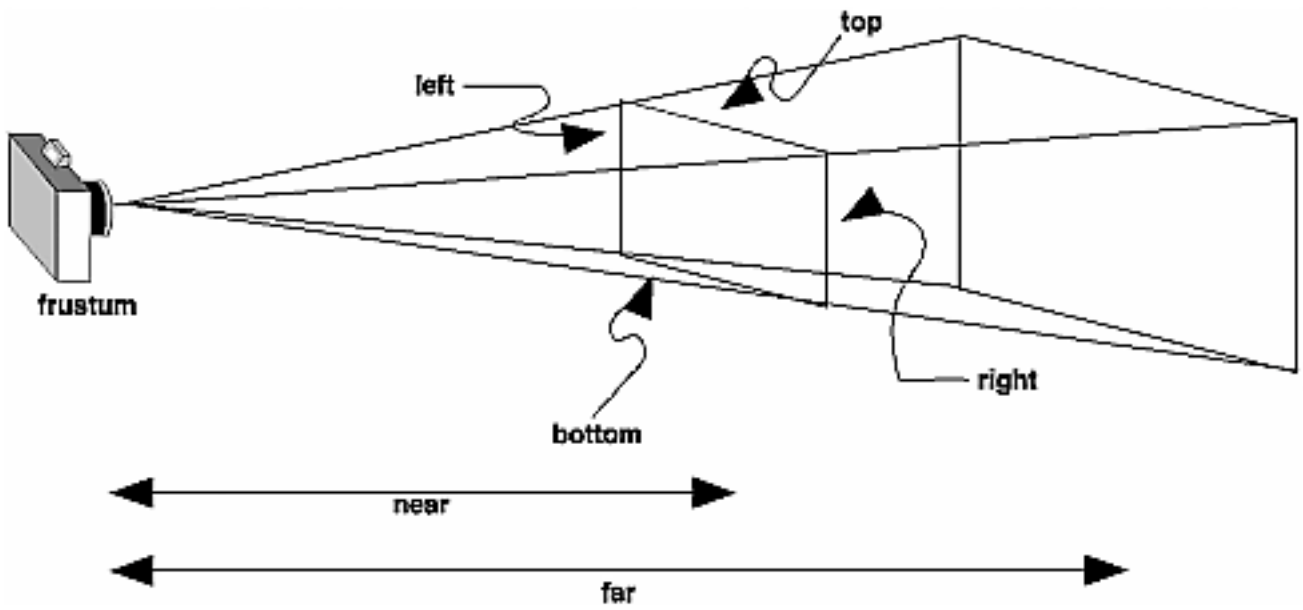
- Perpective projection:
  - The farther away an object lies from the camera, the smaller it appears after projection.
  - Parallel lines converge at the horizon.
  - View volume (called frustum) is a truncated pyramid with a rectangular ba



projection

```
gluPerspective(float fovy, float aspect,
               float near, float far);
```

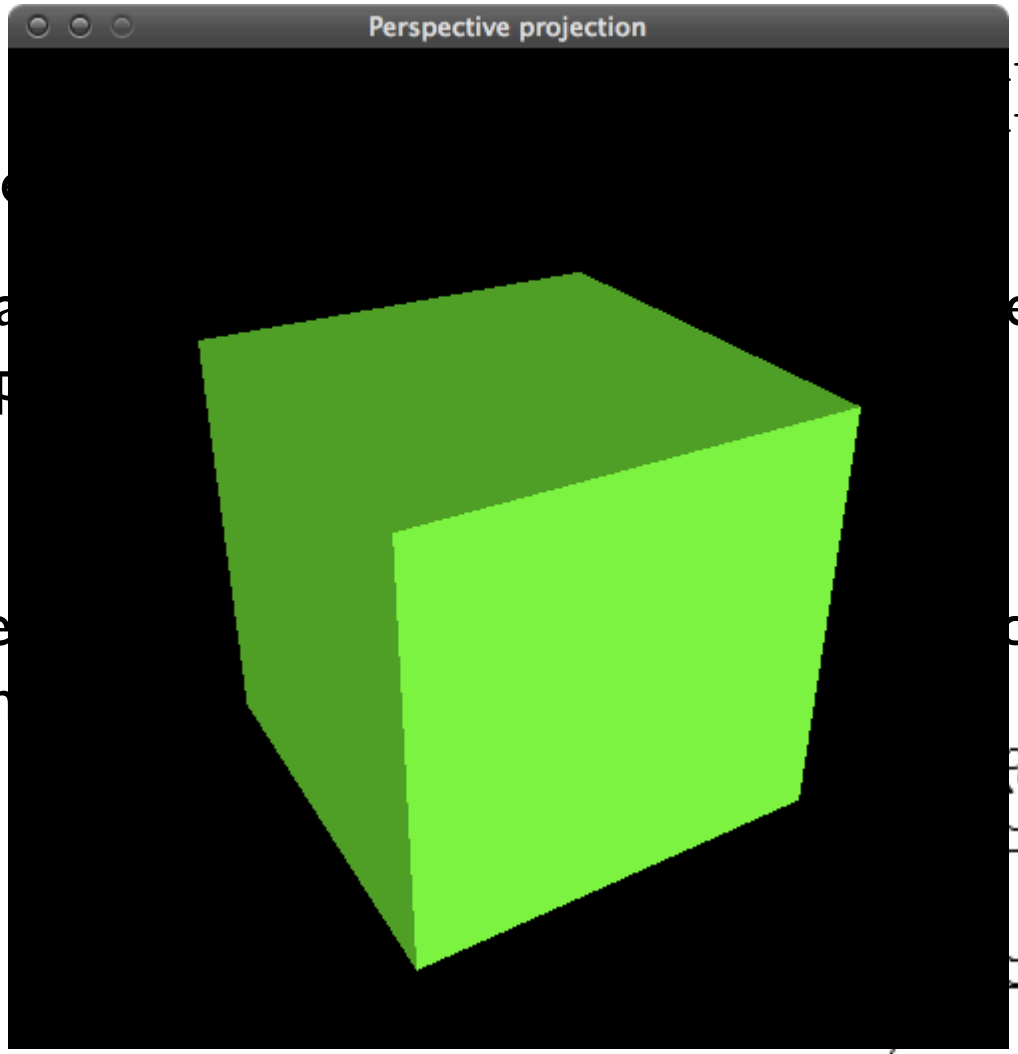
- Perpective projection:
  - The farther away an object lies from the camera, the smaller it appears after projection.
  - Parallel lines converge at the horizon.
  - View volume (called frustum) is a truncated pyramid with a rectangular ba

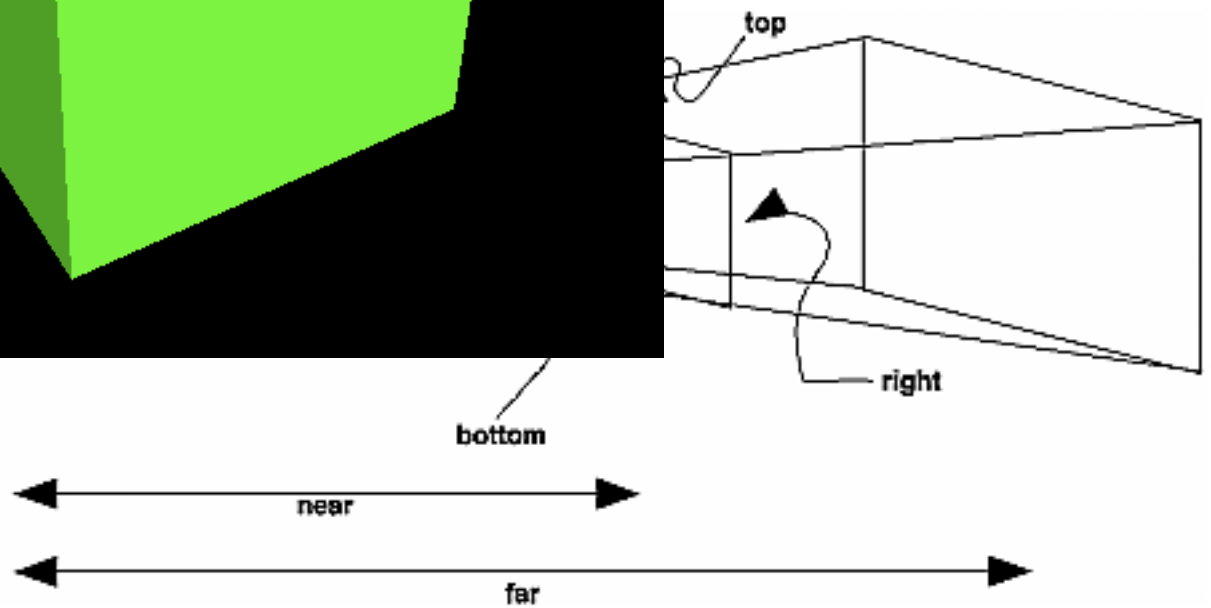


projection

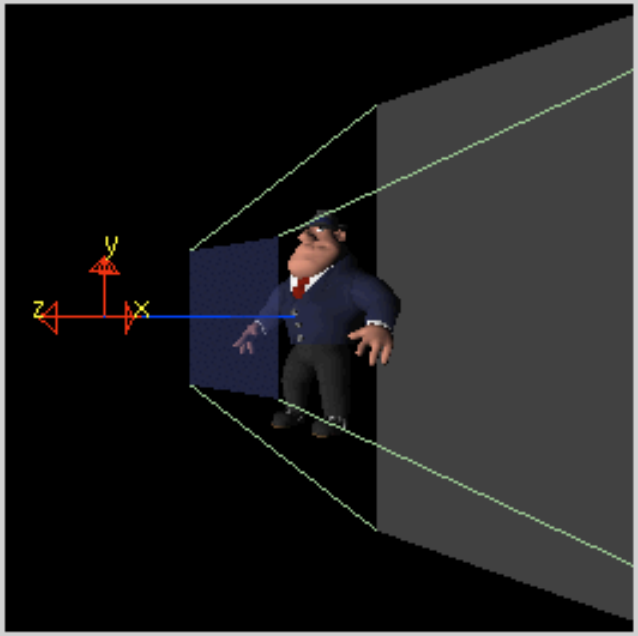
```
t fovy, float aspect,
t near, float far);
```

- Perpective proj
  - The farther a
    smaller it app
  - Parallel lines
  - View volume
    with a rectan





World-space view



Screen-space view

Command manipulation window

```
                    fovy  aspect  zNear  zFar
gluPerspective(     60.0  , 1.00  , 1.0  , 10.0  );
    gluLookAt(      0.00  , 0.00  , 2.00 ,     <- eye
                    0.00  , 0.00  , 0.00 ,     <- center
                    0.00  , 1.00  , 0.00 );    <- up
```

Click on the arguments and move the mouse to modify values.

the depth buffer

```
glEnable(GL_DEPTH_TEST);
```

Topics for today

- Lighting
- Useful bits and pieces
  - repeating key events
  - fullscreen mode
  - animation that is indepenent of CPU speed
- Textures

1/lighting

it's all a fake

light in OpenGL consists of

light in OpenGL consists of

light in OpenGL consists of

- ambient light
  - scattered light (seemingly coming from all directions)

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

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

light in OpenGL consists of

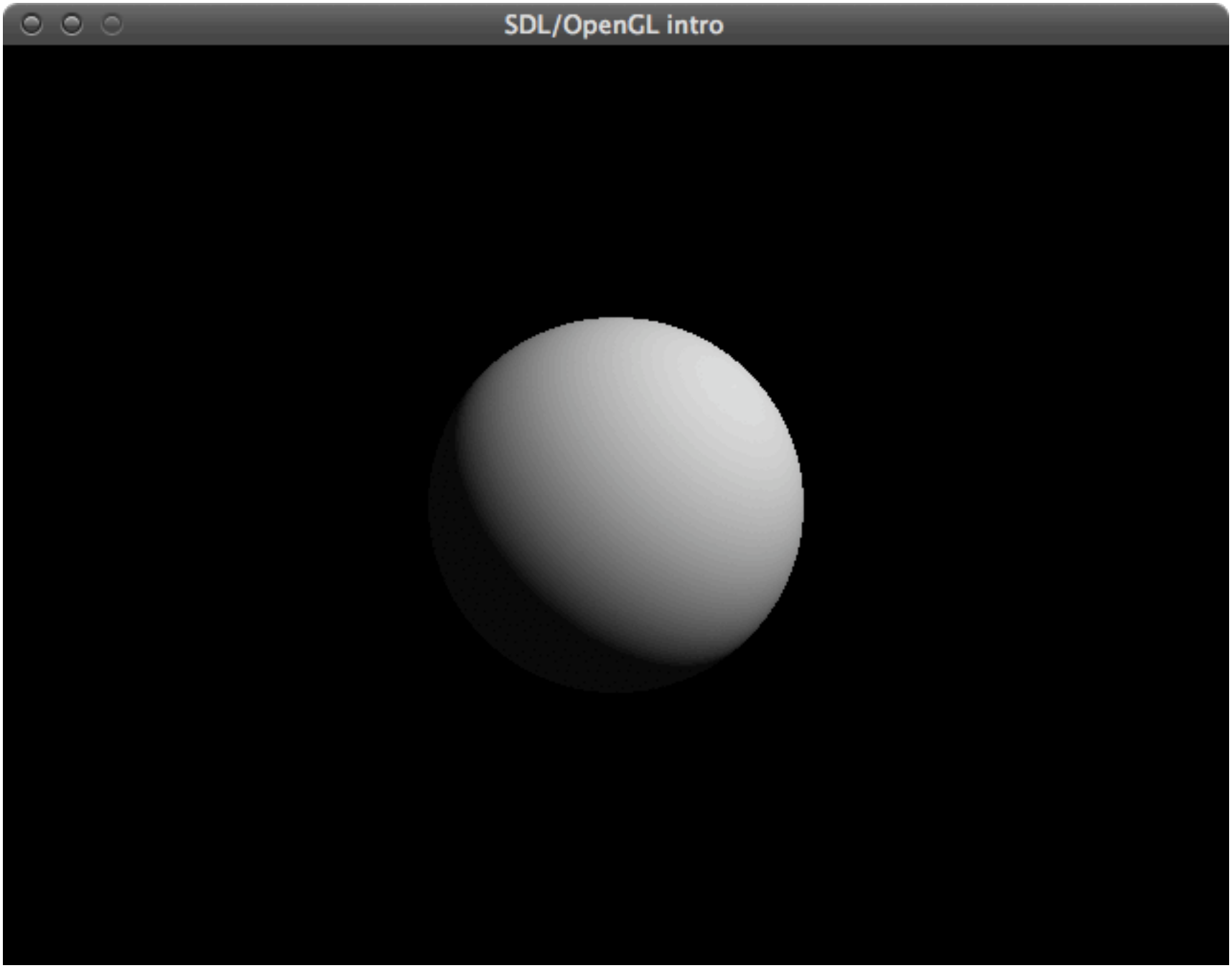
- ambient light
  - scattered light (seemingly coming from all directions)
- diffuse light
  - light coming from one direction
  - scattered evenly when bouncing off a surface
- specular light (“shininess”)
  - light coming from one direction
  - bounces off the surface in a preferred direction
- emitted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

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();
}
```



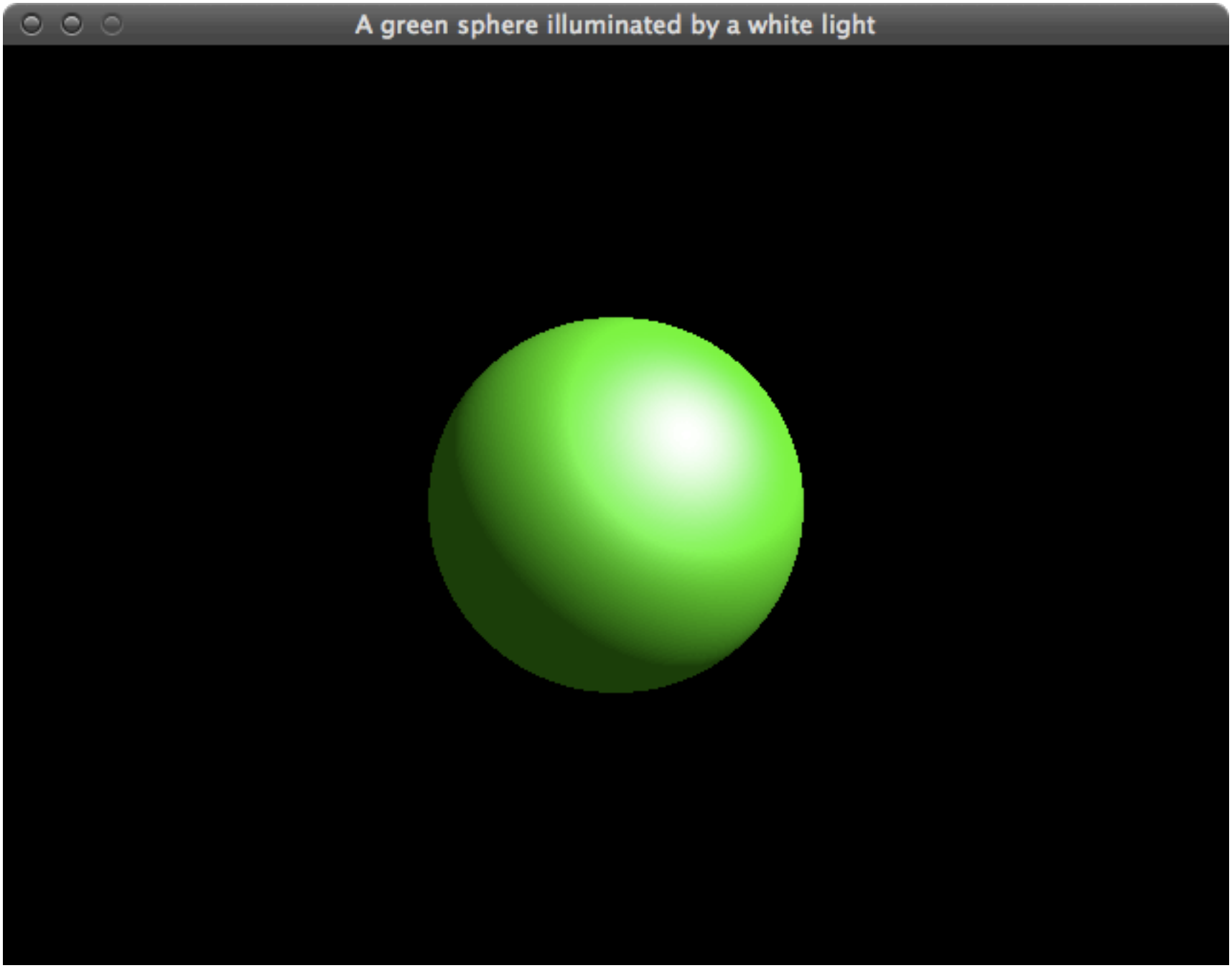
firstlight.cpp

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

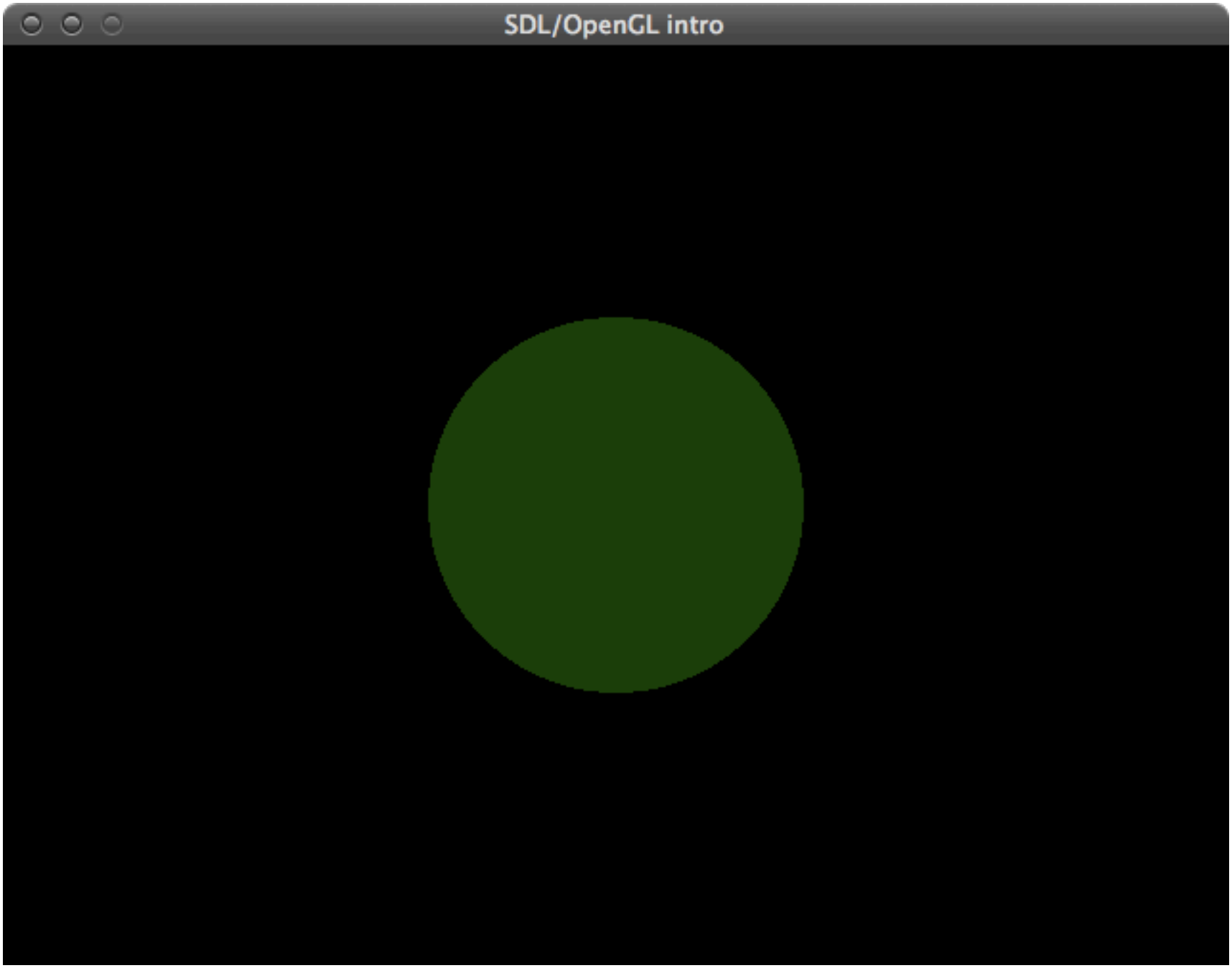
lighting example

```
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
  // ....
```

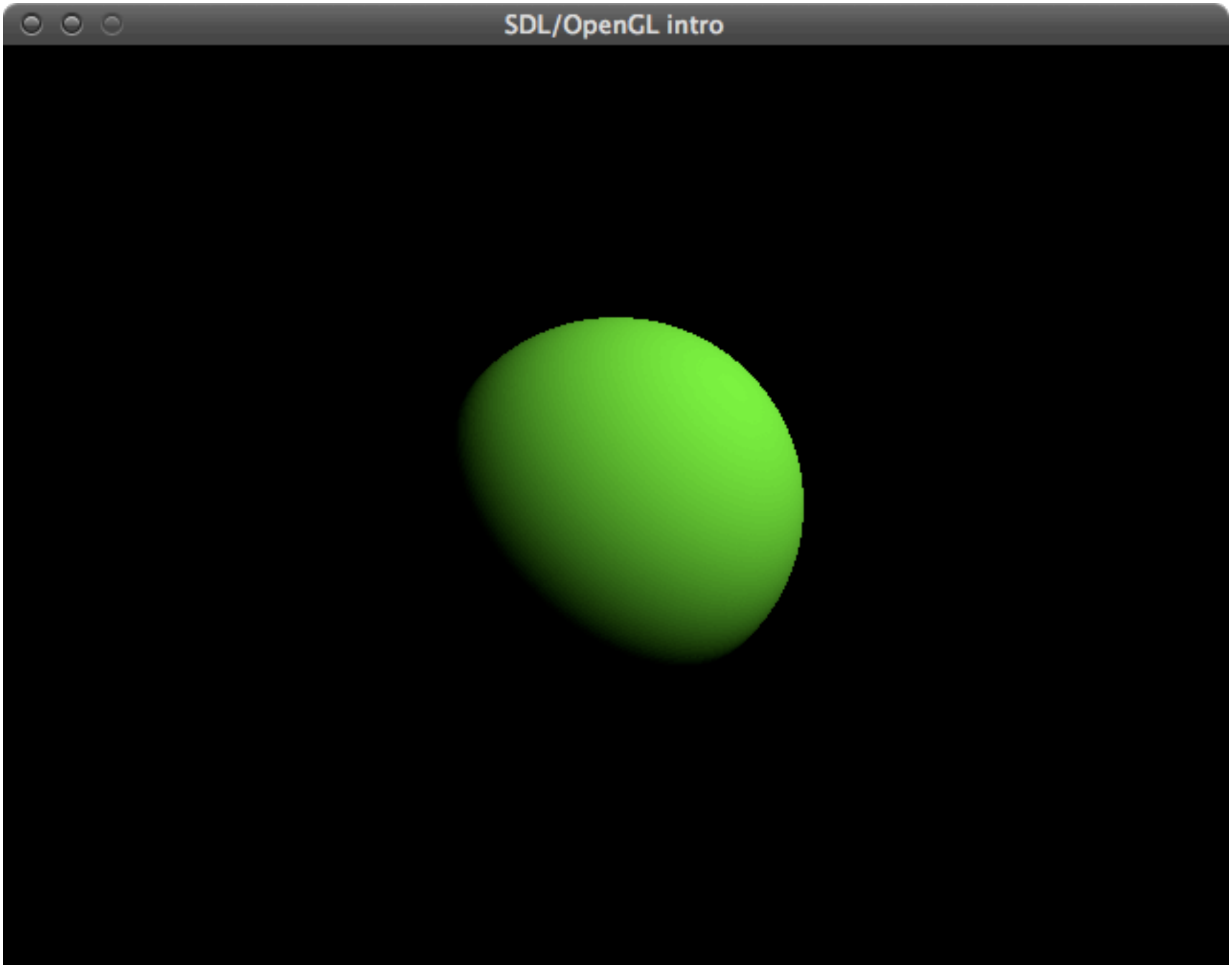


materialcolour.cpp

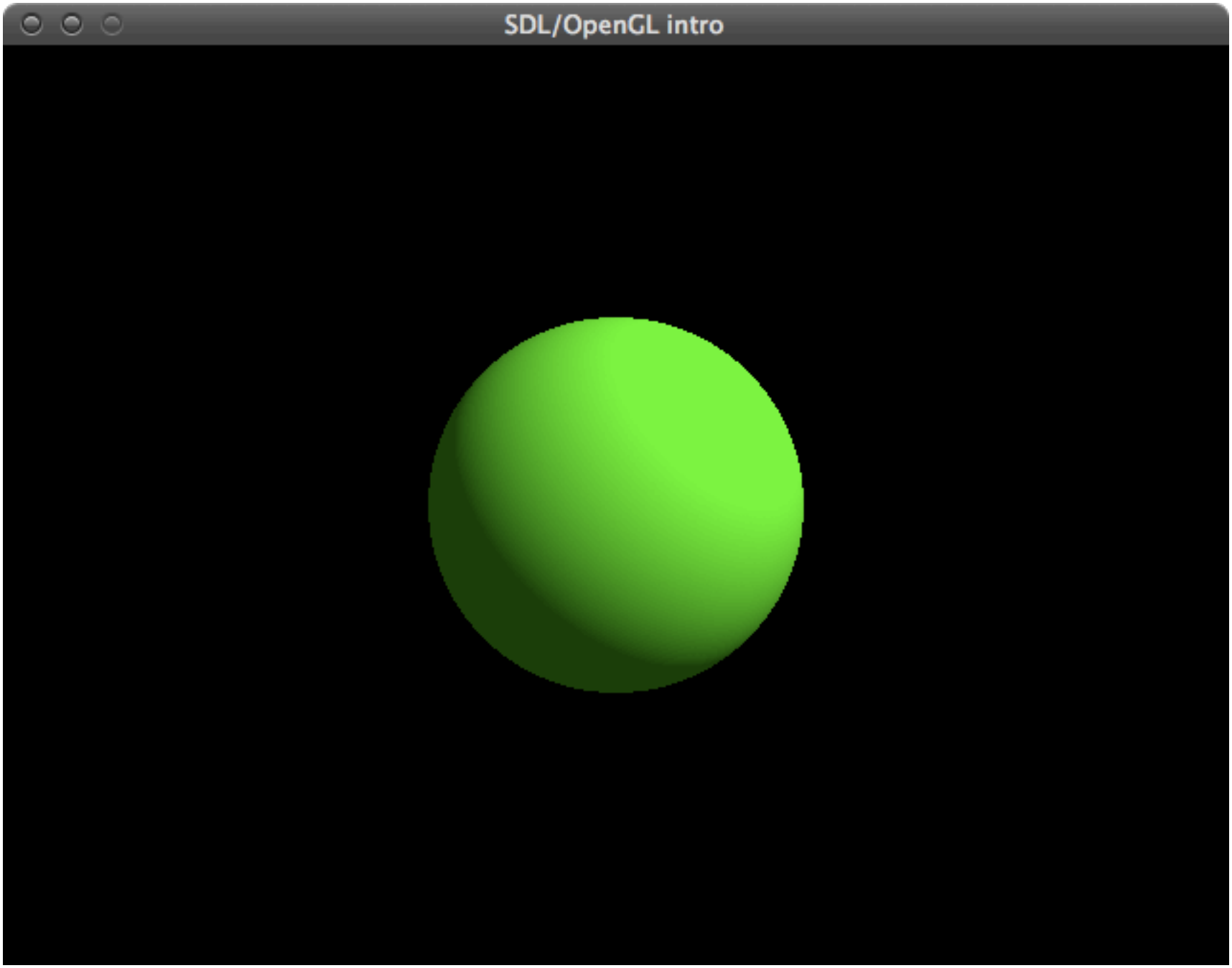
let's have a closer look
at the light components



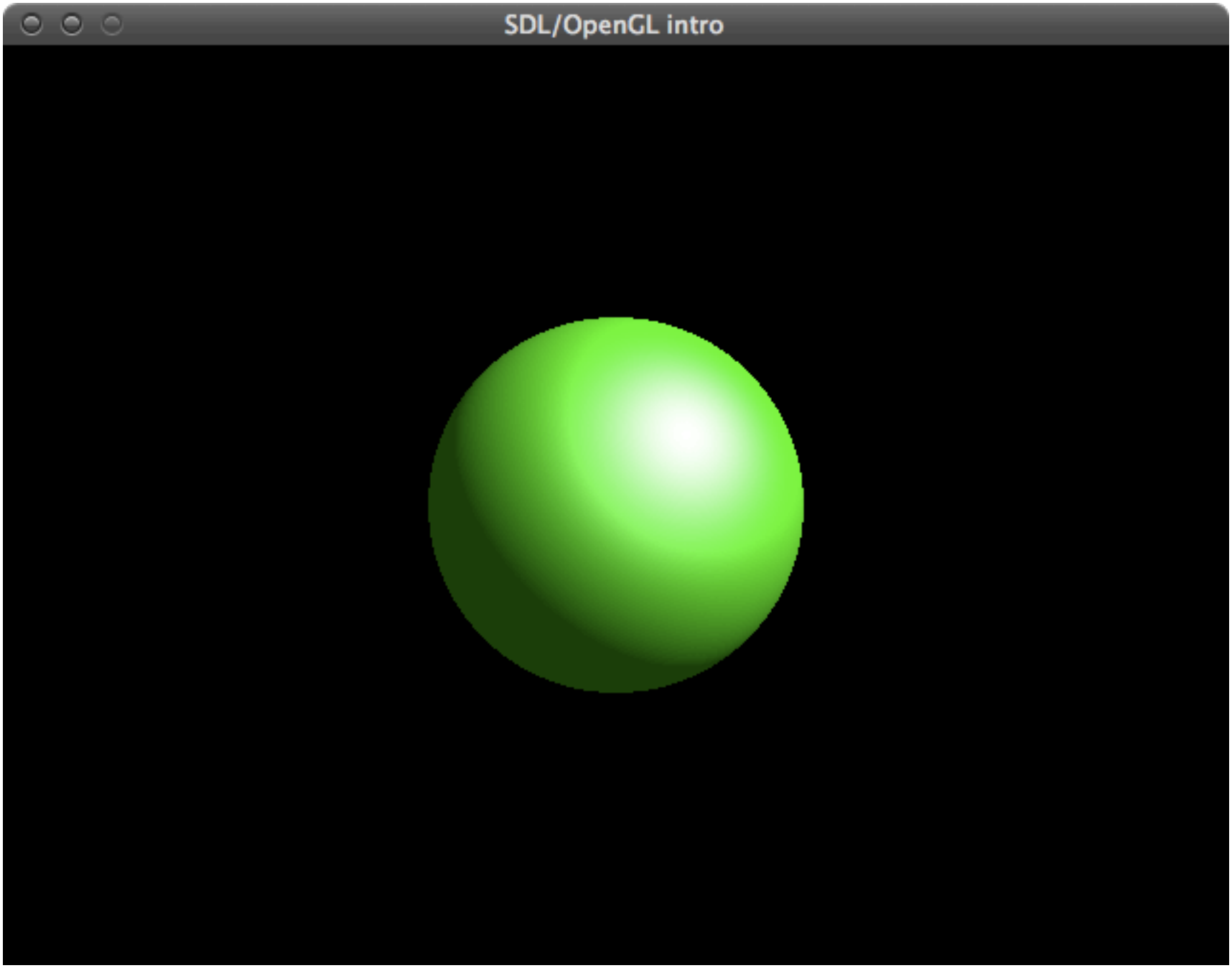
components.cpp *(ambient light only)*



components.cpp *(diffuse light only)*



components.cpp *(ambient and diffuse light)*



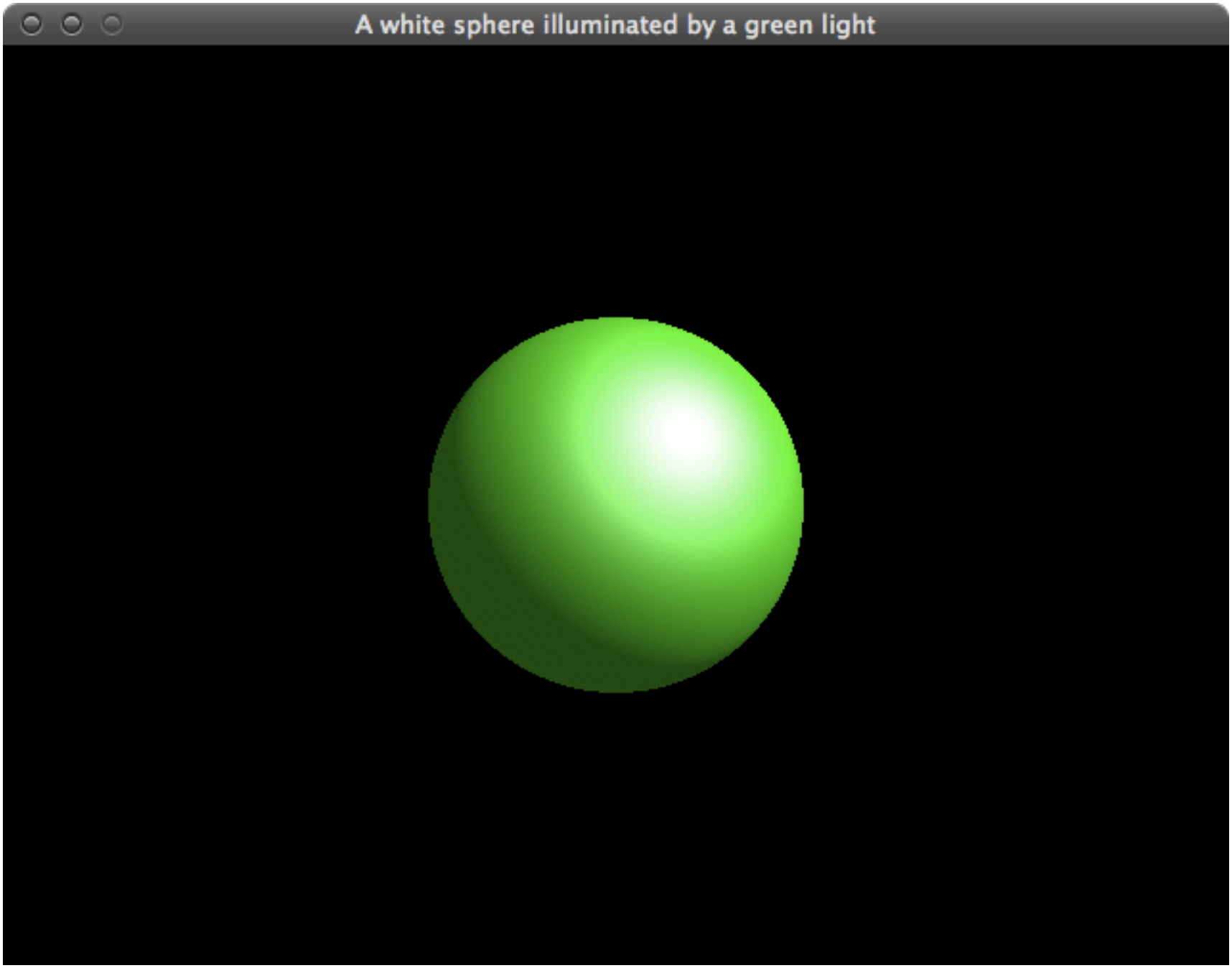
components.cpp *(ambient, diffuse and specular light)*

light source properties

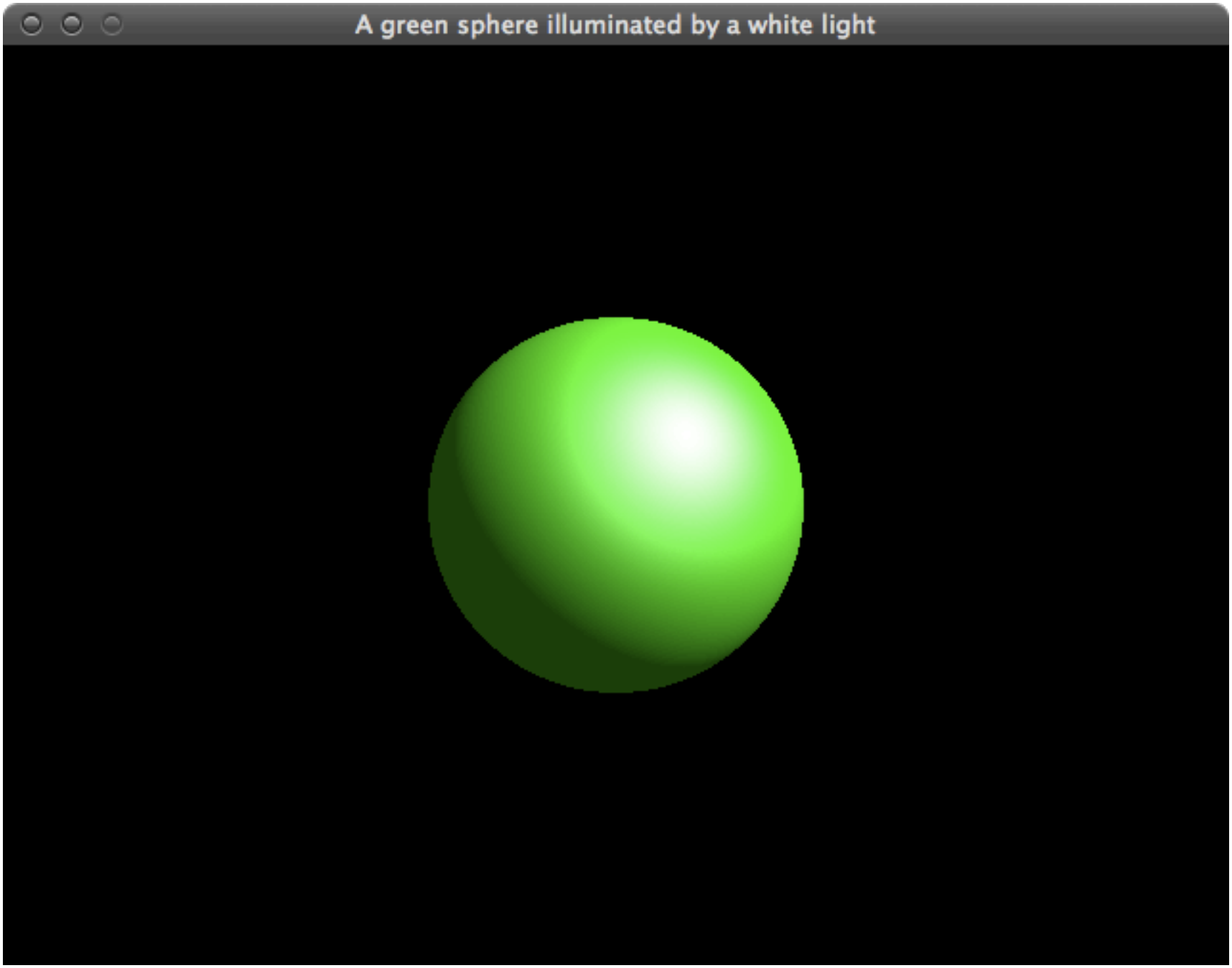
- Properties of light sources can be changed using **`glLight*()`** calls
- Available properties:
  - **`GL_AMBIENT`** (r, g, b, a – default: 0 0 0 1)
  - **`GL_DIFFUSE`** (r, g, b, a – default: 1 1 1 1)
  - **`GL_SPECULAR`** (r, g, b, a – default: 1 1 1 1)
  - **`GL_POSITION`** (x, y, z, w position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

coloured light example

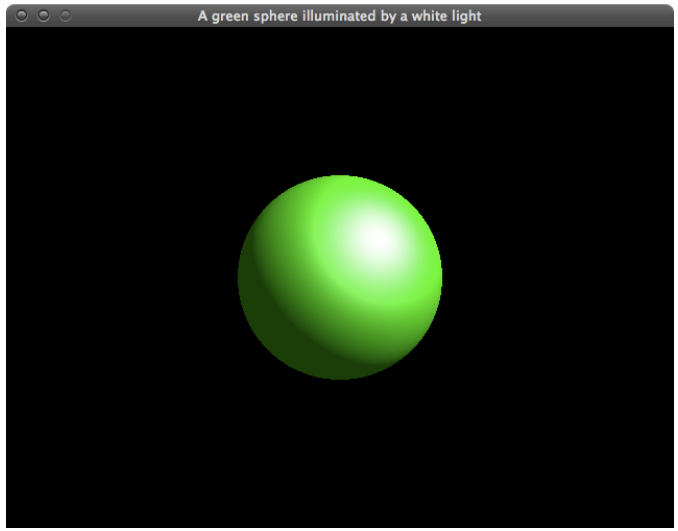
```
void myinit(int width, int height)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 10.0 };
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

  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);
  // ...
```



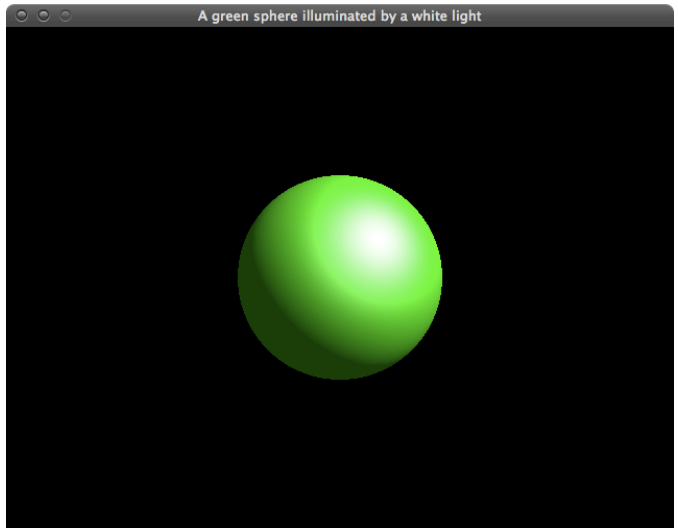
lightcolour.cpp



materialcolour.cpp

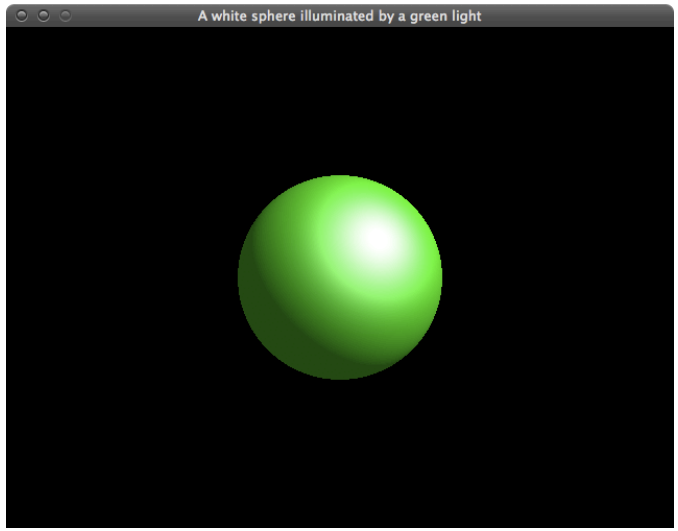


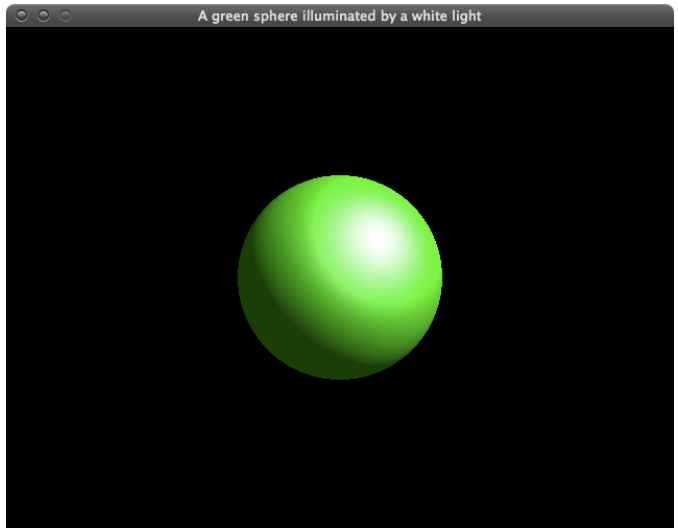
A green sphere illuminated by a white light



A green sphere illuminated by a white light

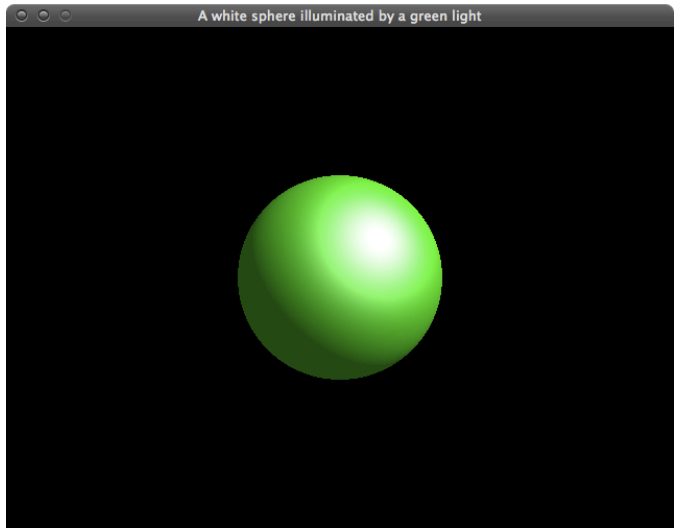
A white sphere illuminated by a green light

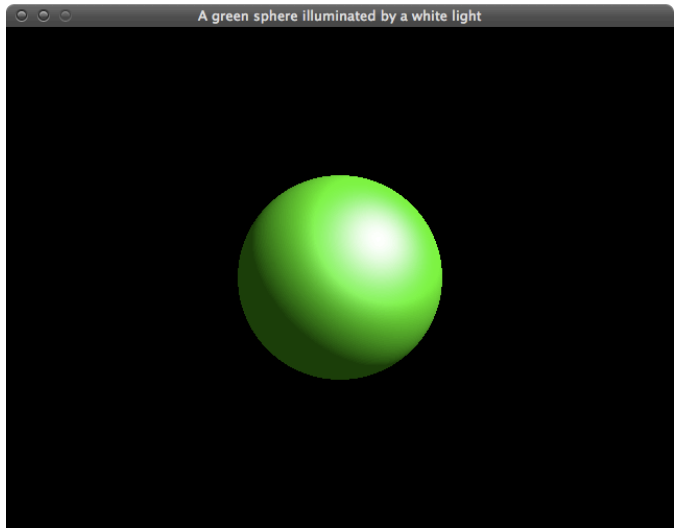




A green sphere illumi by a white light

A white sphere SAME RESULT minated by a green light

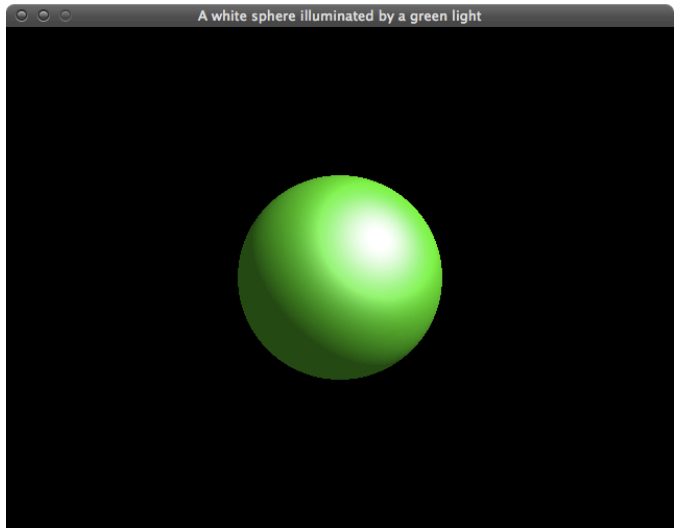




A green sphere illuminated by a white light

ALMOST same RESULT

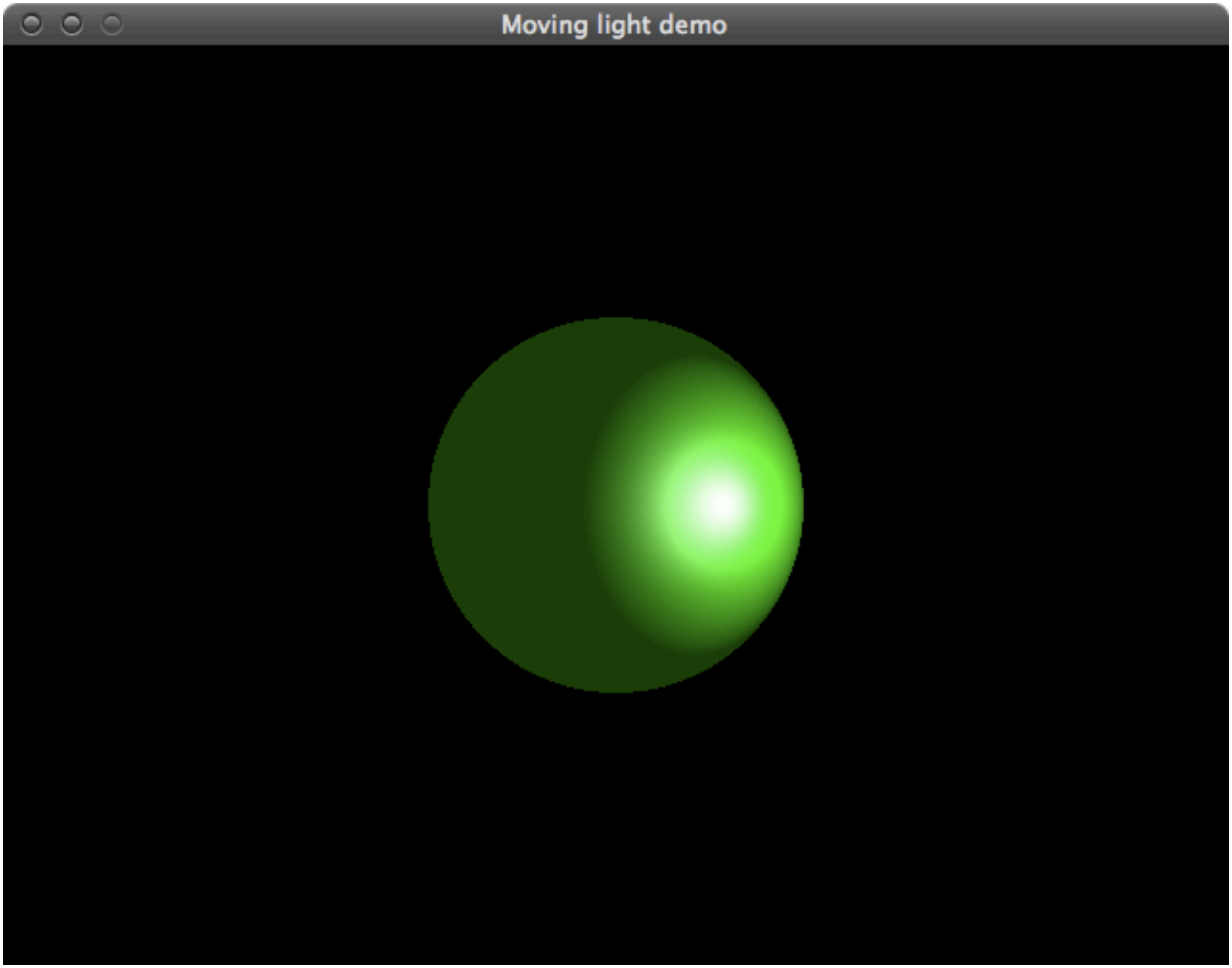
A white sphere illuminated by a green light



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

shading models

shading models

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

shading models

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

shading models

- flat shading
  - face normals (one color per polygon)
- gouraud shading
  - vertex normals (one colour per vertex, interpolated over the polygon along edges and scanlines)
- phong shading
  - interpolate vertex normals at each pixels (not just the colour values)

shading models

in OpenGL:

- flat shading
  - face normals (one color per polygon)
- gouraud shading
  - vertex normals (one colour per vertex, interpolated over the polygon along edges and scanlines)
- phong shading
  - interpolate vertex normals at each pixels (not just the colour values)

shading models

in OpenGL:

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

GL_FLAT

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

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

shading models

in OpenGL:

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

GL_FLAT

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

GL_SMOOTH

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

shading models

in OpenGL:

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

GL_FLAT

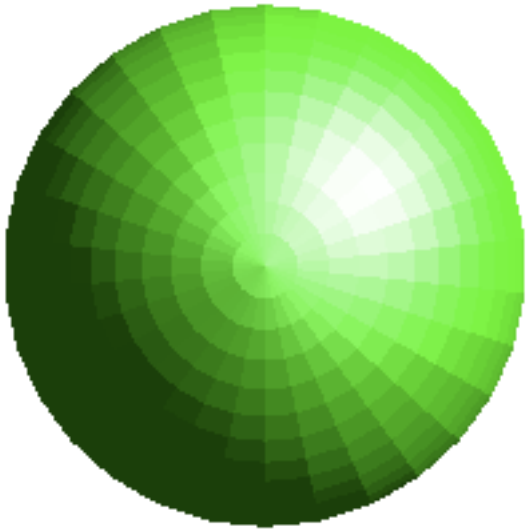
- gouraud shading
  - vertex normals (one colour per vertex, interpolated over the polygon along edges and scanlines)

GL_SMOOTH

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

not implemented!

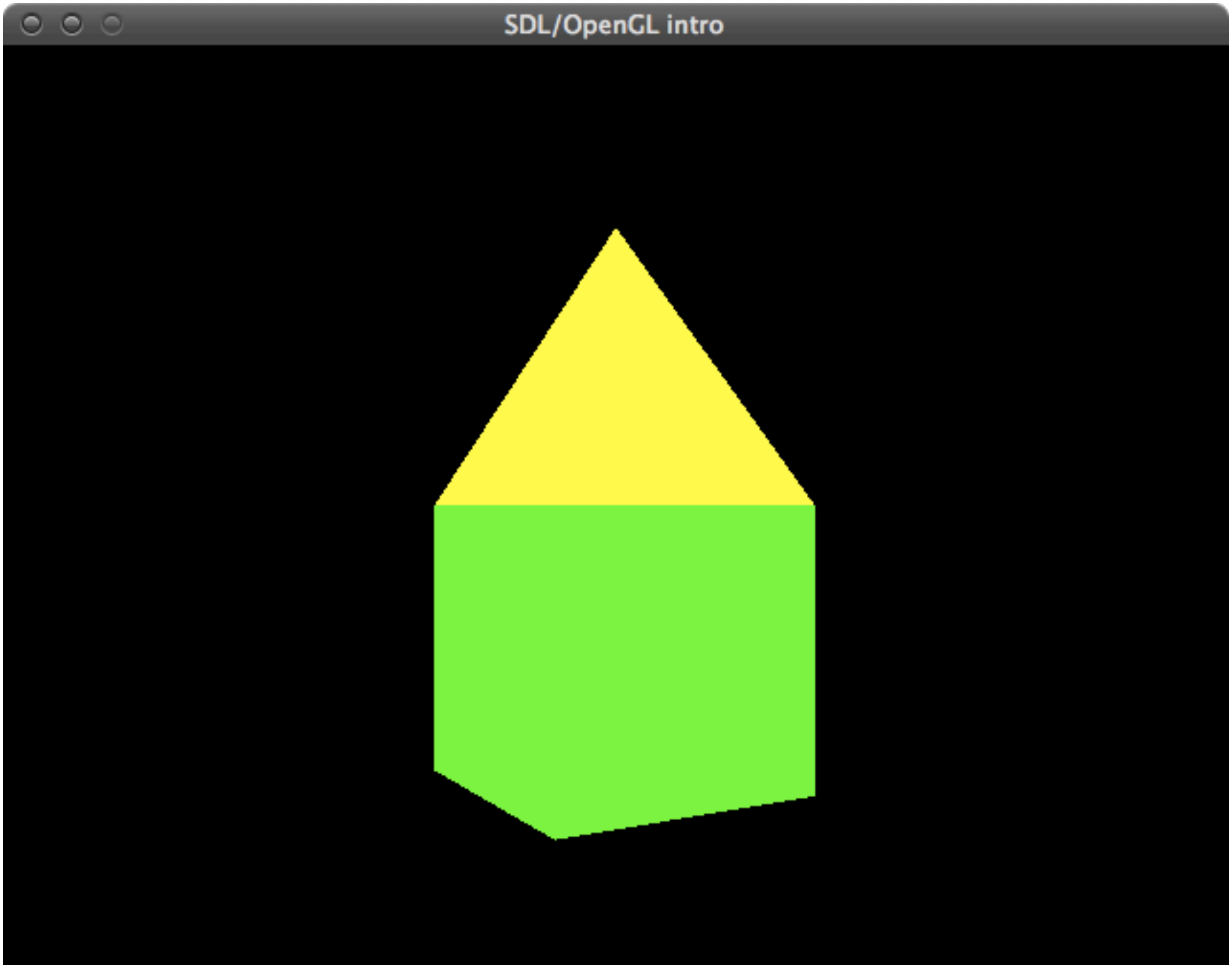
Flat shading vs. Gouraud shading

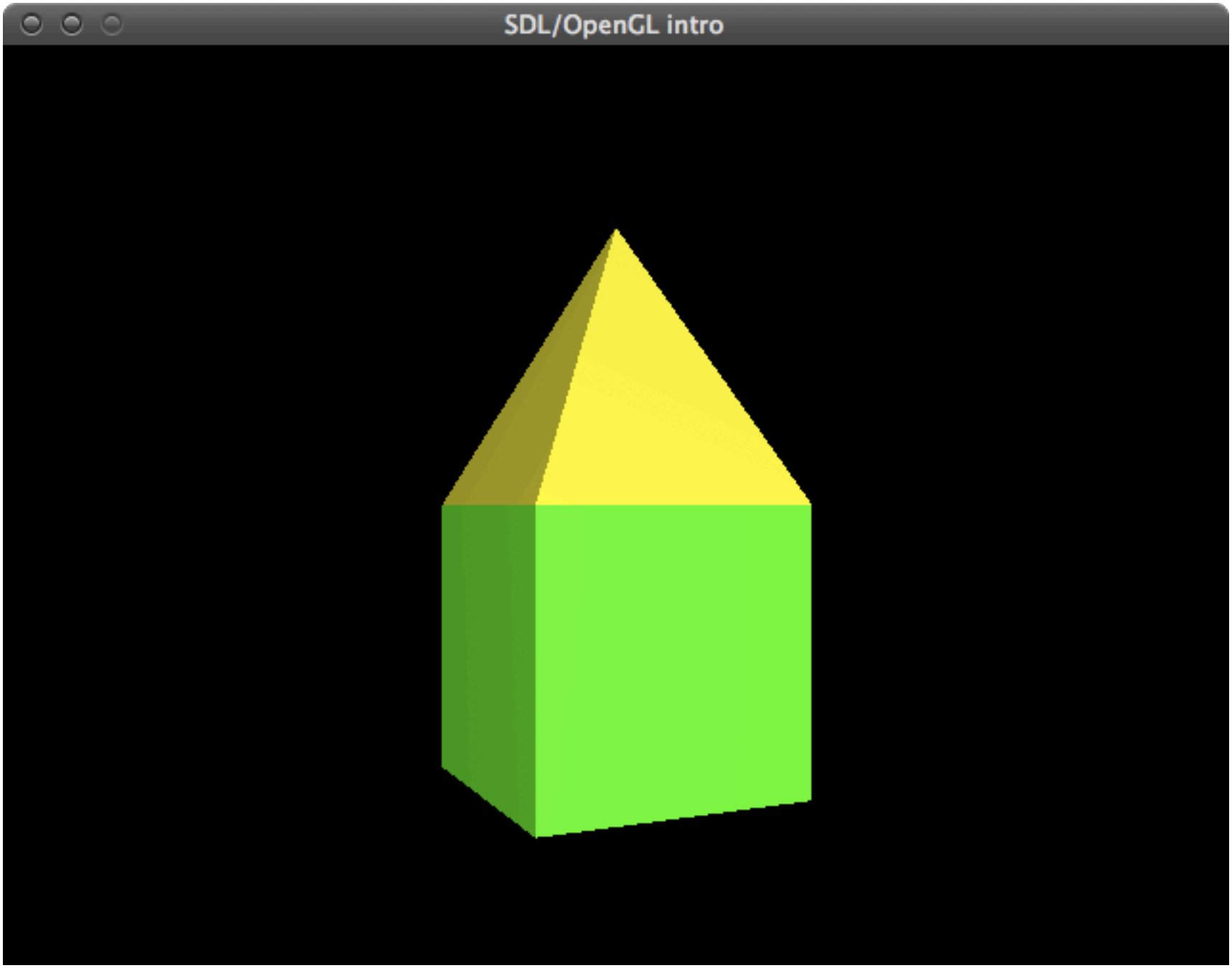


```
glShadeModel(GL_FLAT);
```

```
glShadeModel(GL_SMOOTH);
```

now add lighting to our
3D example


SDL/OpenGL intro


SDL/OpenGL intro

what you need

what you need

- set up a light source

what you need

- set up a light source
- use glMaterial instead of glColor

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate normal vectors

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate normal vectors
  - faces must be defined in counter-clockwise order

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate normal vectors
  - faces must be defined in counter-clockwise order
  - to test: glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate normal vectors
  - faces must be defined in counter-clockwise order
  - to test: glEnable(GL_CULL_FACE);
glFrontFace(GL_CCW);
  - normals should be unit length

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate 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)

what you need

- set up a light source
- use glMaterial instead of glColor
- calculate 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);

thanks! :)