

C++ is fun – Part 13

at Turbine/Warner Bros.!

Russell Hanson

Syllabus

- 1) First program and introduction to data types and control structures with applications for games learning how to use the programming environment Mar 25-27
- 2) Objects, encapsulation, abstract data types, data protection and scope April 1-3
- 3) Basic data structures and how to use them, opening files and performing operations on files – April 8-10
- 4) Algorithms on data structures, algorithms for specific tasks, simple AI and planning type algorithms, game AI algorithms April 15-17
- Project 1 Due – April 17
- 5) More AI: search, heuristics, optimization, decision trees, supervised/unsupervised learning – April 22-24
- 6) Game API and/or event-oriented programming, model view controller, map reduce filter – April 29, May 1
- 7) Basic threads models and some simple databases SQLite May 6-8
- 8) Graphics programming, shaders, textures, 3D models and rotations May 13-15
- Project 2 Due May 15**
- 9) How to download an API and learn how to use functions in that API, Windows Foundation Classes May 20-22
- 10) Designing and implementing a simple game in C++ May 27-29
- 11) Selected topics – Gesture recognition & depth controllers like the Microsoft Kinect, Network Programming & TCP/IP, OSC June 3-5
- 12) Working on student projects - June 10-12
- Final project presentations Project 3/Final Project Due June 12

Recall, Project 2 due next Wednesday



Today is LAB DAY, time to get familiar with OpenGL and GLUT (because PowerPoint is boring)

- Goals of LAB DAY:
 - Get comfortable with 3D graphics
 - Show off new GLUT/OpenGL/DirectX stuff
 - Make pretty pictures
 - Change lighting on pretty pictures
 - Change viewport, etc. for pretty pictures
 - Make 3D stuff on the computer
 - Understand what's happening with everything

- Installing FreeGLUT/GLUT on Visual Studio 2010:

<http://visualambition.wordpress.com/2010/08/12/glut-and-visual-studio-2010/>

- Install directions for Xcode:

Step by Step: Xcode 4, OpenGL & GLUT in Lion

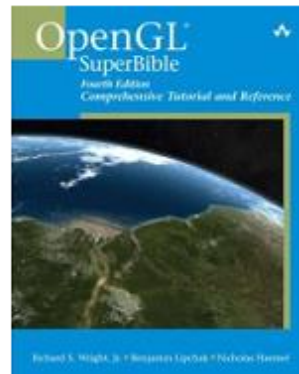
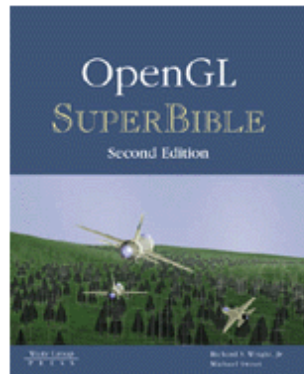
<http://www.autofasurer.net/wp/?p=106>

<http://www.opengl.org/sdk/docs/books/SuperBible/>



OpenGL SuperBible Comprehensive Tutorial and Reference

BY RICHARD S. WRIGHT, NICHOLAS HAEMEL, GRAHAM SELLERS AND BENJAMIN LIPCHAK

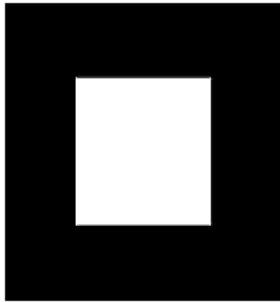


The fifth edition of OpenGL® SuperBible, the newest member of the Addison Wesley OpenGL Technical Library, is now available!

- Sample Code From Fourth Edition
- Complete source code for all platforms (182 MB): SB-AllSource.zip
- Source with pre-built binaries (Windows, 120MB): SB-WinwBin.zip
- Source with pre-built binaries (Mac OS X, 208MB): SB-MacwBin.zip
- Source only, no binaries (Windows, 60MB): SB-WinSrc.zip
- Source only, no binaries (Mac OS X, 71MB): SB-MacSrc.zip
- Source only, no binaries (Linux, 48MB): SB-LinuxSrc.tar.bz2

<http://www.sgi.com/products/software/opengl/examples/glut/examples/>

fogst.jpg	Demonstration program exhibiting fog techniques.	fogst.c
fontdemo.jpg	Bitmap and stroke fonts demonstration program.	fontdemo.c
glpuzzle.jpg	3D puzzle that can solve itself automatically.	glpuzzle.c
glutplane.jpg	OpenGL planes a plenty - add and subtract them.	glutplane.c
halomagic.jpg halomagic1.jpg (dinosaur)	Neat haloing effect using the stencil buffer.	halomagic.c
highlight.jpg	This program demonstrates the use of the GL lighting model. Objects are drawn using a grey material characteristic. A single light source illuminates the objects.	highlight.c
lightlab.jpg	Lighting laboratory to experiment with different material properties and lights.	lightlab.c
mjkwarp.jpg	Texture warping example to show many texturing options of OpenGL.	mjkwarp.c
molehill.jpg	Really, really shiny nurbs/evaluators example.	molehill.c
movelight.jpg	<p>This program demonstrates when to issue lighting and transformation commands to render a model with a light which is moved by a modeling transformation (rotate or translate). The light position is reset after the modeling transformation is called. The eye position does not change.</p> <p>A sphere is drawn using a grey material characteristic. A single light source illuminates the object.</p> <p>Interaction: pressing the left or middle mouse button alters the modeling transformation (x rotation) by 30 degrees. The scene is then redrawn with the light in a new position.</p>	movelight.c



whiterect

Figure 1-1 : White Rectangle on a Black Background

Example 1-1 : Chunk of OpenGL Code

```
#include <whateverYouNeed.h>

main() {

    InitializeAWindowPlease();

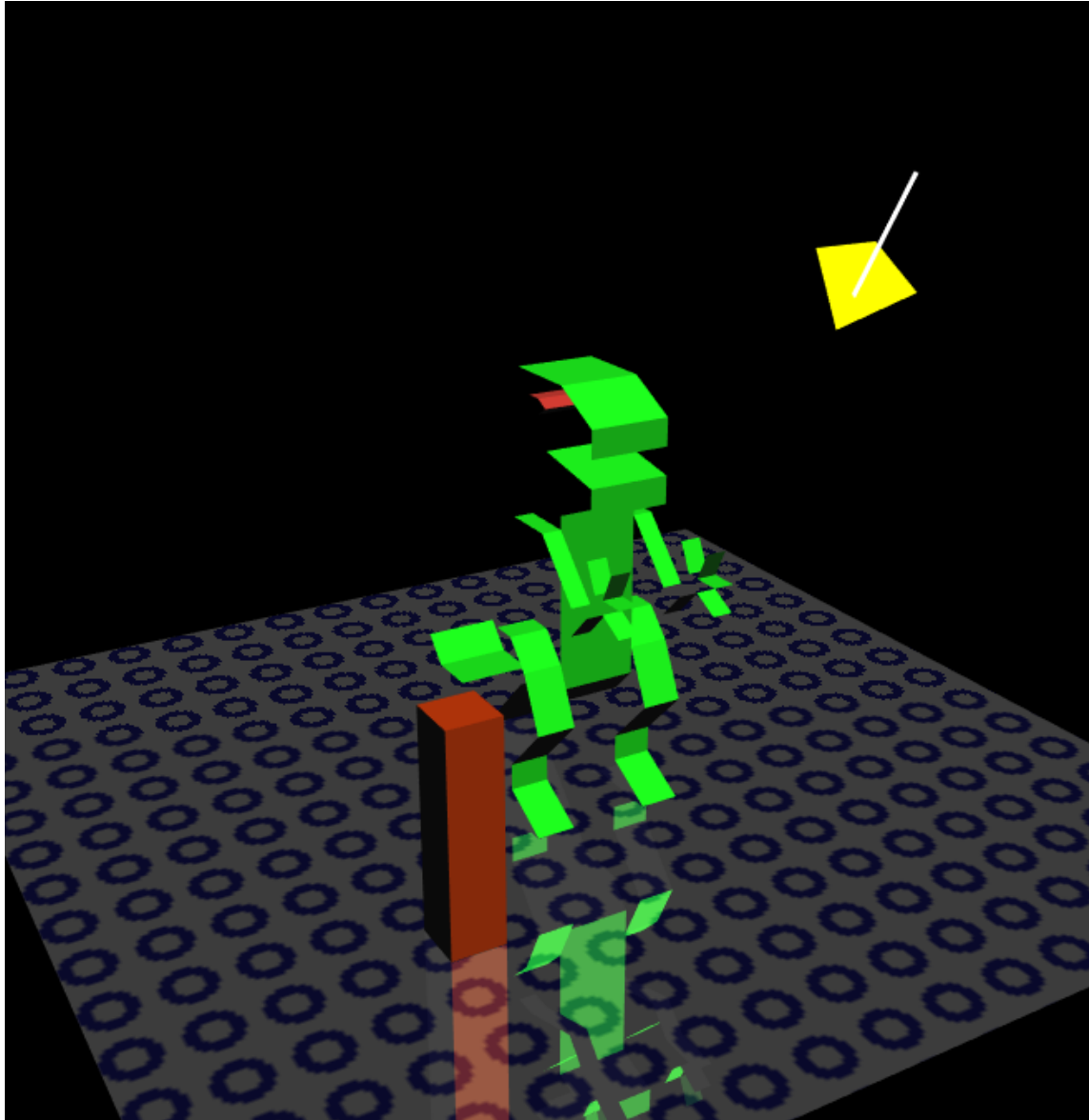
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();
    glFlush();

    UpdateTheWindowAndCheckForEvents();
}
```

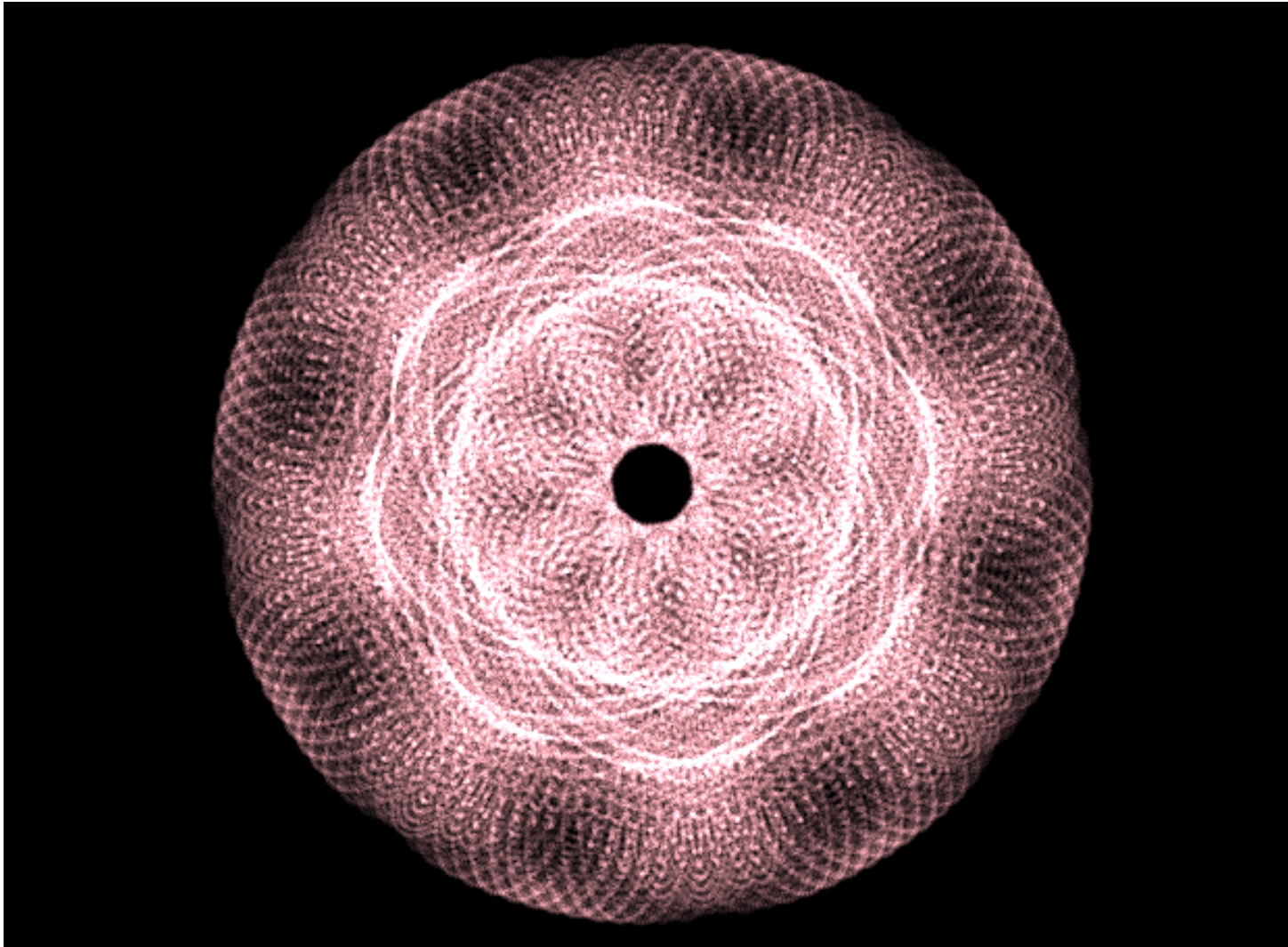
The first line of the **main()** routine initializes a *window* on the screen: The **InitializeAWindowPlease()** routine is meant as a placeholder for window system-specific routines, which are generally not OpenGL calls. The next two lines are OpenGL commands that clear the window to black: **glClearColor()** establishes what color the window will be cleared to, and **glClear()** actually clears the window. Once the clearing color is set, the window is cleared to that color whenever **glClear()** is called. This clearing color can be changed with another call to **glClearColor()**. Similarly, the **glColor3f()** command establishes what color to use for drawing objects - in this case, the color is white. All objects drawn after this point use this color, until it's changed with another call to set the color.

The next OpenGL command used in the program, **glOrtho()**, specifies the coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen. The next calls, which are bracketed by **glBegin()** and **glEnd()**, define the object to be drawn - in this example, a polygon with four vertices. The polygon's "corners" are defined by the **glVertex3f()** commands. As you might be able to guess from the arguments, which are (x, y, z) coordinates, the polygon is a rectangle on the z=0 plane.

halomagic



harmonograph



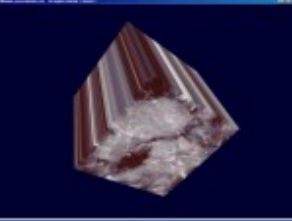


movelight

Welcome to movelight.
Right mouse button for menu.



Hold down the left mouse button
and move the mouse horizontally
to change the light position.

http://www.spacesimulator.net/wiki/index.php?title=3d_Engine_Programming_Tutorials

	<p>Texture Mapping How to load an image and use it to cover an object. This technique is called texture mapping and is the most responsible for the realism of the scene.</p> <p>English Italian</p>	<p>OpenGL 3.3 Porting English</p>	<p>Linux MacOS MacOSX(Cocoa) Windows(VC6) Windows(VC.NET)(OpenGL3.3)</p>
	<p>3ds Loader The data structure for a complex object can't be written by hand. There are a lot of programs that help to create 3d meshes in a very quick way. In this lesson we study how to load a 3ds file, a format that is really famous on the net.</p> <p>English German Italian</p>	<p>OpenGL 3.3 Porting English</p>	<p>Linux MacOS MacOSX(Cocoa) SDL Windows(VC6) Windows(VC.NET)(OpenGL3.3)</p>
	<p>Vectors, Normals and OpenGL Lighting Lighting is the other important thing, after texture mapping, that adds to the scene further realism. Without lights all the objects seems flat. We will study also Vectors and normals because they are indispensable to make lighting calculations.</p> <p>English German Italian</p>	<p>OpenGL 3.3 Porting English</p>	<p>Linux MacOSX(Cocoa) Windows(VC6) Windows(VC.NET)(OpenGL3.3)</p>

Example 1-2 : Simple OpenGL Program Using GLUT: hello.c

```
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{
    /* clear all pixels */
    glClear (GL_COLOR_BUFFER_BIT);

    /* draw white polygon (rectangle) with corners at
     * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
     */
    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

    /* don't wait!
     * start processing buffered OpenGL routines
     */
    glFlush ();
}

void init (void)
{
    /* select clearing (background) color */
    glClearColor (0.0, 0.0, 0.0, 0.0);

    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*
 * Declare initial window size, position, and display mode
 * (single buffer and RGBA). Open window with "hello"
 * in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0; /* ISO C requires main to return int. */
}
```

Double Buffering

Most OpenGL implementations provide double-buffering - hardware or software that supplies two complete color buffers. One is displayed while the other is being drawn. When the drawing of a frame is complete, the two buffers are swapped, so the one that was being viewed is now used for drawing, and vice versa. This is like a movie projector with only two frames in a loop; while one is being projected on the screen, an artist is desperately erasing and redrawing the frame that's not visible. As long as the artist is quick enough, the viewer notices no difference between this setup and one where all the frames are already drawn and the projector is simply displaying them one after the other. With double-buffering, every frame is shown only when the drawing is complete; the viewer never sees a partially drawn frame.

If you are using the GLUT library, you'll want to call this routine:

```
void glutSwapBuffers(void);
```

Example 1-3 illustrates the use of `glutSwapBuffers()` in an example that draws a spinning square as shown in Figure 1-3. The following example also shows how to use GLUT to control an input device and turn on and off an idle function. In this example, the mouse buttons toggle the spinning on and off.

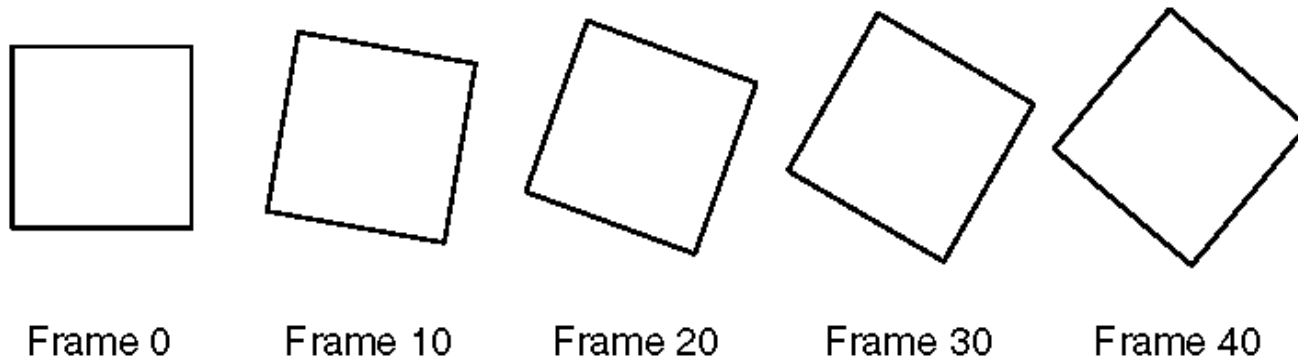


Figure 1-3 : Double-Buffered Rotating Square

Example 1-3 : Double-Buffered Program: double.c

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>

static GLfloat spin = 0.0;

void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
```

```

    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
    glutSwapBuffers();
}

void spinDisplay(void)
{
    spin = spin + 2.0;
    if (spin > 360.0)
        spin = spin - 360.0;
    glutPostRedisplay();
}

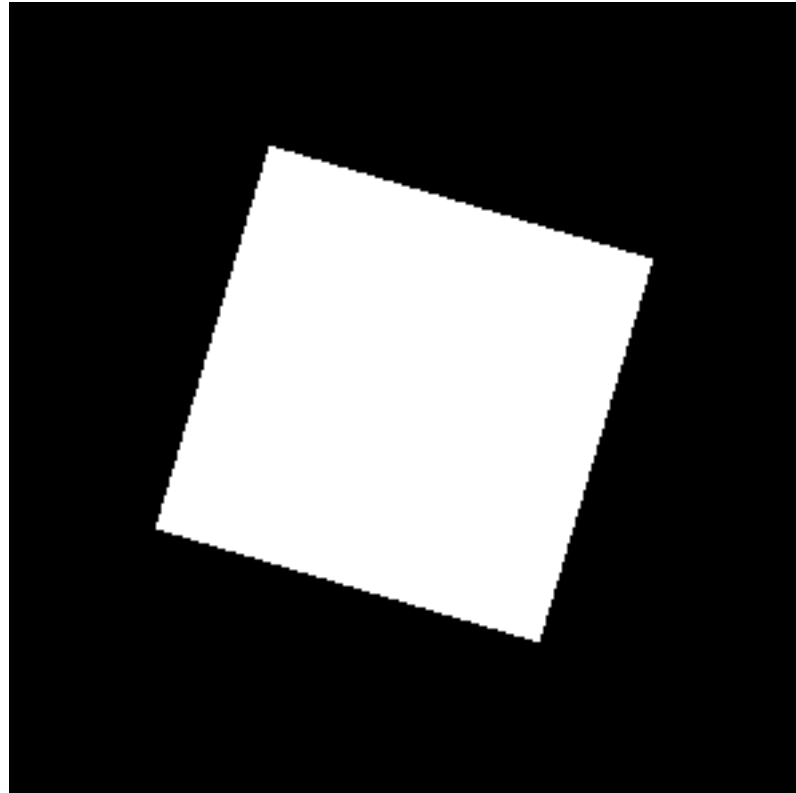
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(spinDisplay);
            break;
        case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        default:
            break;
    }
}

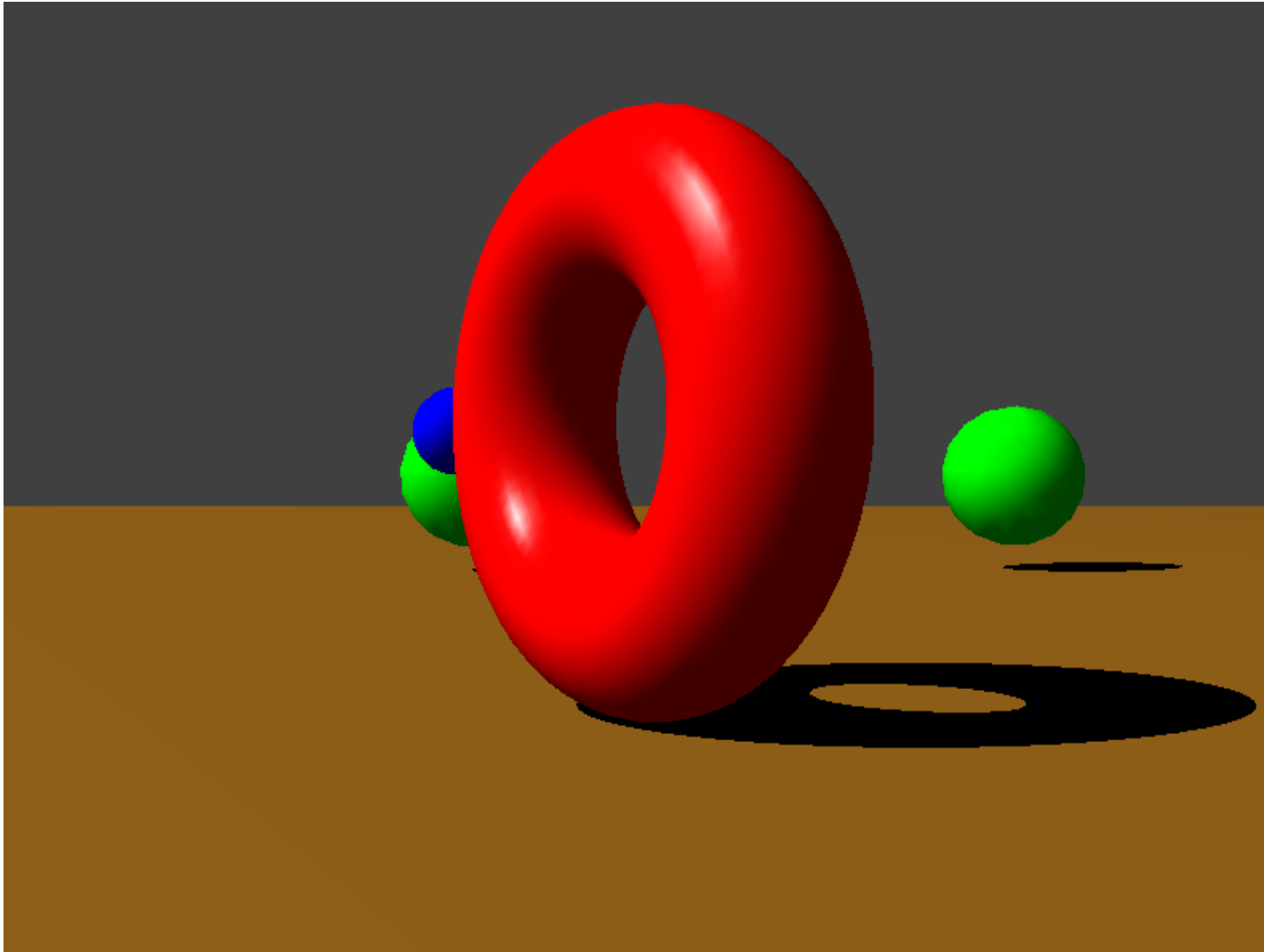
/*
 * Request double buffer display mode.
 * Register mouse input callback functions
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}

```

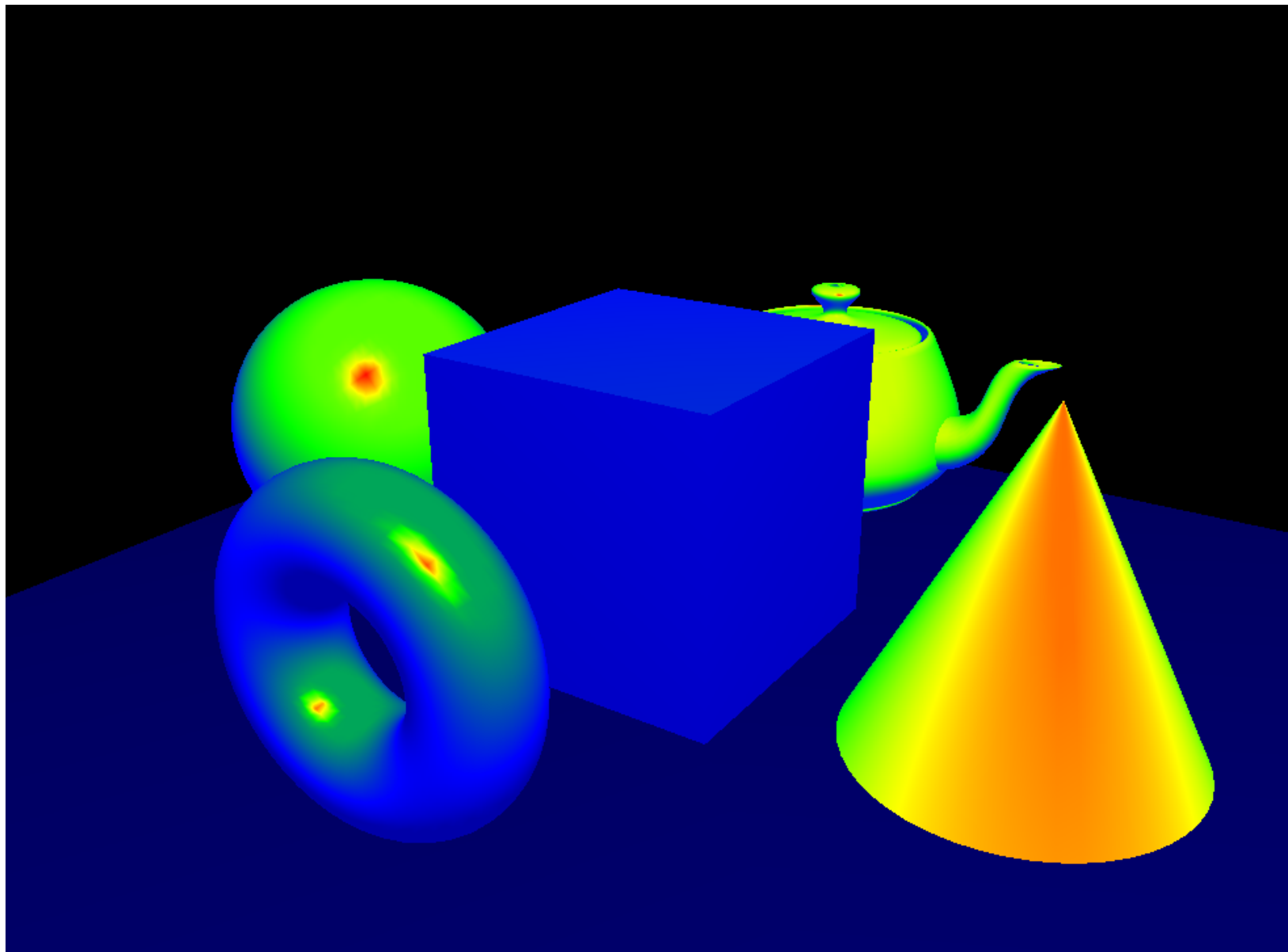

doublebuffer spinning rectangle



sphereworld



fragmentshaders.cpp



shadowmap

