

Programming Assignment 1: Getting Started with OpenGL

Handed out Thu, Sep 7. The program must be submitted to the grader by Thu, Sep 21 (any time up to midnight). Submission instructions will be forthcoming. Here is the late policy: up to six hours late: 5% of the total; up to 24 hours late: 10%, and then 15% for every additional day late.

Overview. The goal of this assignment is to review the basics of OpenGL and GLUT (and to have fun) in anticipation of the major programming project, which will be handed out later. Your objective is to implement a simple interactive program using OpenGL, subject to the general requirements presented below. The emphasis is on showing that you know how to use OpenGL to achieve certain visual effects, to manipulate simple 3-dimensional models, and to process graphical user inputs through the keyboard and mouse. Producing an interesting or playable game is not important (of course, the grader will need to test the various elements of your program).

Good Code Structure. (Roughly 10%) Before you begin, take a few moments to plan out your program's design. Although this is a small program, the goal here is to build a framework that will make it easy to add more features without scrapping your old framework entirely. Although a small program like this one could be reasonably implemented with a single .cpp file, such a minimalist approach will not work for the final semester project. We will provide you with a *template program*, which provides a simple, but general structure on which you can build. The general program structure is described in the *Template Program* section below.

Required Elements. You have a great deal of latitude in implementing this assignment, but to demonstrate that you have mastered the OpenGL programming skills needed for the semester project, your program should contain all of the following elements.

3-dimensional Rendering: (Required) Your program should render and manipulate objects in 3-dimensional space. (But the physical movement and interaction can be 2-dimensional in nature.)

Mouse and Keyboard Input: (Roughly 15%) Your program should accept inputs from the keyboard, both regular keys (e.g., 'q' for "quit") and special keys (e.g., '↑' for moving the camera up), mouse button clicks, and mouse motion (e.g., using the mouse to aim a gun and a mouse click to shoot it).

Lighting and Textures: (Roughly 20%) Your scene should consist of multiple light sources (not the same ones we gave in the sample program) and texture mapping involving at least two different image files (which you will provide).

Variable numbers of objects: (Roughly 10%) Our template program supports a fixed number of objects. Through the use of lists and/or arrays, your program should allow for multiple objects. For example, by hitting a key the user can create new objects in your world. (You may place upper and lower limits on the number of such objects.)

Collision detection/response: (Roughly 15%) Your program should have some form collision detection (e.g., by testing the bounding box of the moving objects) and should provide some

nontrivial form of physical response to such collisions (e.g., bouncing off or sticking together). It need not be physically realistic, but we will provide a handout on how to handle elastic collisions between spheres. Collisions should be easy enough to generate that the grader can readily test this feature.

Composite objects: (Roughly 10%) Complex objects are often formed by grouping together a number of simpler pieces. At least one object in your program should involve replicating a single model in multiple positions. (For example, to draw the legs of a table, create a single leg object and an associated drawing method. To draw the entire table, but this drawing routine four times, each time subject to a different OpenGL transformation.)

Compound motion: (Roughly 15%) The parts of some object in your program (perhaps a composite object) should be able to move relative to each other (e.g., as in the flapping of a bird's two wings or two limbs connected by a hinged joint.) This is designed to demonstrate that you understand how to apply OpenGL's transformation capabilities.

Your Submission. For grading purposes, your program must be able to be compiled under Windows using Visual Studio.NET (either 2003 or 2005) and run on a Windows machine that has OpenGL, GLU, and GLUT installed. Your submission should contain all the files that the TA will need to compile, execute, and test your program. If your program makes use of any other files or libraries you must include them in your submission. In addition your submission should contain:

- A file `ReadMe.txt` explaining how to compile and run your program. (Ideally the TA should just have to build the Release version and run it from within Visual Studio, but if some other form of execution is needed, please explain it here.)
- Explain how to use your program and describe any special features you have implemented (very important), and any bugs or limitations that you are aware of.
- Provide all the files needed for compiling your program. (E.g. The Visual Studio solution (.sln) file and the project (.vcproj) file.
- Your source files.
- Any additional files (e.g. image files needed for texture mapping).

These should all be combined in a bundle (as either a .zip or .rar file) and then submitted to the grader. Instructions will be forthcoming.

Acknowledging Sources. There are many resources in books and on the web containing sample OpenGL code. The goal of this project is to show that you have mastered the ability to use OpenGL to achieve a specific goal, and not that you can copy code. You are allowed to make use of any *small* code fragments from any external sources that you like, provided that you carefully cite your sources in your ReadMe file. (In particular, you should have written at least 2/3 of the code that you submit.)

The use external source code with giving credit is considered cheating. Cite all your sources, or prepare to suffer the consequences. If you are in doubt, check with me.

Template Program. The template program has the following principal objects. Feel free to use and modify these files (without acknowledgement) in preparing your program. (Or you may start from scratch, if you prefer.)

Game: Basic game object. It contains the following major objects.

Control Controls the general flow of the game including pause and un-pause.

World: Represents the state of the world, that is, all the objects, characters, and their geometric and graphical properties. For this program, the World contains the following elements: Lights, Floor, Table, and Ball. All of these objects are derived from a common base class, WorldEntity.

Lights: Stores the light sources for the scene.

Floor: Stores a texture-mapped floor.

Table: Stores a green table top that floats about the floor.

Ball: Stores a shiny red ball that bounces upon the table top.

All of these WorldEntity objects support the following methods. For some of these objects the operation makes no sense, and so they may simply be left undefined.

reset(): Reset this object back to its original state as in the start of the program.

advance(): Advance the state of the animation incrementally.

processEvent(): Process some user input.

redraw(): (Re)draw the object.

Camera: Represents the camera position and projection characteristics. The camera supports methods for moving the camera and zooming it in and out. As with WorldEntity objects it provides the reset() and processEvent() operations, and it also provides methods for setting up the project (analogous to redrawing for WorldEntity objects) and can respond to requests to reshape the window and toggling into full screen mode. methods as WorldEntity objects,

Event: This object represents a single user input event. Non-user (system-generated) events such as redrawing and window reshape events are handled by a separate mechanism.

EventHandler: Contains callback functions that are used to handle user and system events.

Geom3D: Provides utilities for processing points and vectors in 3-dimensional space (e.g., dot product, cross product, conversion of degrees to radians.)

RGBpixmap: Provides methods for inputting and outputting images represented as .bmp files.

Texture: Stores a texture-map (from RGBpixmap) along with methods for processing them through OpenGL.

Utilities: Provides basic error handling procedures for the game.