Design of a 3D Graphics Engine
TJHSST Senior Research Project Proposal
Computer Systems Lab 2009-2010

Joseph Hallahan, period 5

April 7, 2010

Abstract

Physics play a large role in many modern programs. Simulations of every kind and numerous video games use physics equations and functions in order to generate as realistic scenario as possible. The uses of a physics engine, a framework of various pieces of code containing these physics functions, are many. A physics engine that works along with a graphics engine - one that could take on both duties simultaneously, would be very useful indeed. This project involves creating a fully runnable graphics engine that also supports physics relating to collision detection. In addition to displaying and performing physics on 2D images, it will also keep track of keyboard input. C++ will be the language used to create this engine, and the libraries used will be SDL and OpenGL.

keywords: graphics engine, physics engine, SDL, OpenGL, C++

1 Introduction

1.1 Situation/Statement of Problem

The Engine is often the most important part of a program. It contains all the functionality needed for whatever purposes the program may need. Some are used for graphics, some for physics, and some for both. One feature common to all of them is that they can be used for many different projects, rather than just the one they are originally made for. As a result, they must either be able to have all this functionality without needing to rely on any external information, or use such generic external information that any similar program would be able to provide it as well.

1.2 Purpose

The goal of this project is to create a working engine that focuses both on two-dimensional graphics and physics. It should be able to be

useful for various projects, including possibly some agent based modeling, gaming, and simulations. It will need to be able to be used as a standalone program to ensure that it is not tied specifically to the programs I test it with. It will need to do the math related to the drawing and movement of shapes, and also relating to collision handling. The engine will also need to be able to get input from the user, particularly from the keyboard.

2 Background

Graphics engines are very important to the work done by many programmers today. In the process of creating an application, after the initial design phases are complete, the first thing the programming team will typically do is begin work on the application's engine. Some applications, if they are new and different from what has been done before, will need an entirely new engine and, as a result, will require a lot more work. However, if it is similar to one that has already been completed and released, it is much easier to use an existing engine than to create a new one. In addition to reqiring less work, completed engines have already been tested for stability and compatibility, so programs that use them typically do not need as much post-production work.

Because graphics code can become rather complex and not user-friendly, it would make an excellent candidate for use in an engine. Encapsulating all the code needed for rendering graphics would take a large burden off of programmers who will then be able to spend more time on the less rigid parts of projects. The basic code for graphics does not change very much from application to application, so putting it in a separate file for other programs' use would be a good idea.

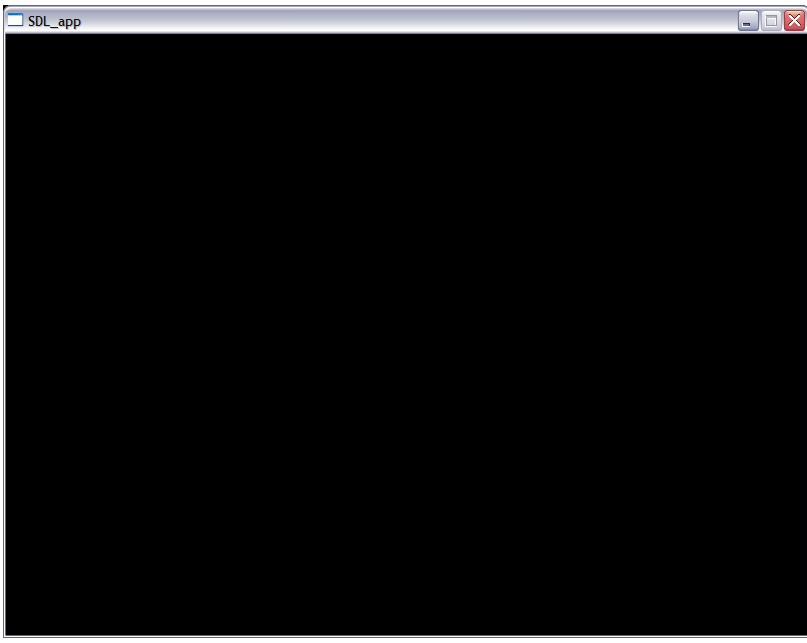

Figure 1: A window created with SDL.

3 Development

3.1 Structure in C++

In the first semester of this year I explored the OpenGL library and learned how to use it in a single C++ source code file. My first project was simply a program that could draw various 2D or 3D shapes in OpenGL, but this was very basic and not really useful outside of that single project. However, it did demonstrate how OpenGL was used, and was able

to render text and get user input as well as simply rendering shapes. In the second quarter, I expanded on this by attempting to use multiple source code files in a project, but unfortunately I thought doing so would be more similar to Java and, as a result, the source files would not compile. While a lot of my code was sound, there was no way to make the files talk to each other and find all the methods needed to work. However, the methods I wrote are very similar to those I needed for this quarter's project. One major difference my project has undergone this quarter is the introduction of a new library, SDL, into the project. SDL, or Standard DirectMedia Layer, is a useful library for C++ that allows fairly fast graphics, audio, and input to be used. However, the main point of my project is to create a graphics/physics engine, not to verify these claims of speed. The other advantage SDL has is its compatibility with OpenGL and the relative ease with which they work together. All that is needed to use OpenGL code in an SDL project is to change three or four lines of code. Processing input and window creation with SDL is much easier than with OpenGL, and that is what I use it primarily for in this iteration of my project. I use OpenGL for the actual drawing itself. I'm not using any of SDL's audio abilities in this project, although they are supposedly well implemented as well. I am using the same sort of structure as I had last quarter, with a source file containing function and class definitions, a header file containing declarations for those functions and classes, and a test file containing a class extending the engine and the program's "main()" method.

3.2 Collisions and Physics

While there are many ways of detecting collisions between different objects, the handling of them is a different method. The simplest way is with rectangles, in which in the event of a collision (in a frictionless vacuum and assuming the collision is completely elastic) the rectangles both simply reverse their x and y velocities. Probably some of the hardest ways involve trying to figure out what to do when multiple concave shapes with some curved edges all collide. In certain difficult scenarios what some programmers do is imagine what is known as a "bounding box" that completely or at least mostly encapsulates each irregular shape, and then treats each bounding box using the aforementioned method for rectangle collision handling. Although in real life there are many different variables affecting how collisons work, to keep my program simpler for now I assume several things about my colliding objects. I assume that all collisions are elastic, in a frictionless vacuum, the objects are all identical mass, and there are no external forces acting on anything, such as gravity.

4 Tests and Analysis

My program, though it is called "Pong2," actually cannot, at the moment, play a game of Pong. It was originally going to, but most of my time this quarter was spent improving the engine rather than working on the test program. Unexpected weather resulted in losing a lot of the time I might have oth-

erwise spent on this; however, I am satisfied with what got done on the engine itself so far. For the first time, I may not be starting a whole new project next quarter - I will be able to continue working on this one. Adding SDL to the project was beneficial in that it now correctly creates a window, something it was unable to do as of last quarter. My goal for the initial testing of my engine was just to create a few circles and rectangles and bounce them around the screen. The circles would use basic bounding boxes for collision detection and handling, and the rectangles would be easy enough to deal with. However, although the Circle class worked flawlessly and without generating any errors, the Rectangle class for some reason would not work properly. Although it was nearly identical to the Circle class, it kept throwing " 'Rectangle' is not a type" errors. I eventually had my code looked at and mostly fixed, but as of right now my test program is for circles only.

5 Results

I would have preferred to put more functionality in the engine this quarter, but I am at least pleased that I got my structure issues from last quarter resolved. Apparently my second quarter OpenGL code was flawed in some way, because my drawing code didn't work at first. This may have just been due to a missing line or two, since it didn't generate any errors or anything. The switch from separate files to a Code::Blocks (my IDE) project permitted the files all to be compiled and linked correctly on my Windows system, but unfortunately I do not have the Linux skills necessary to do the same thing from a command line. Luckily Code::Blocks is apparently open-source and therefore compatible with Linux, so by next quarter I hope to be able to install it as part of my personal quota on the school computers.

6 Conclusion

Overall, even with the scheduling issues this quarter, I have completed a lot of work toward my eventual goal, and next quarter I hope to begin more complicated collision handling and physics code. This program (Pong2.cpp) correctly ran despite not containing a single line of graphics or physics code. As a matter of fact, the fact that the main() method consists of three lines is a testament to the success of the engine in taking a workload off of a programmer using it. This project is actually an engine, as opposed to my earlier efforts: a single one-time-use file and a header file used as a library. My code not only creates all the resources needed for a program to run, but also runs it and stops when it is finished. My biggest changes to this iteration of my project were adding this extra functionality, adding SDL, and beginning to add some physics methods. Next quarter I hope to be able to improve upon my collision and physics code, and come up with a program that is able to test all of that. My "Pong" program will go by the wayside in favor of a new test program that can illustrate the full functionality of the engine.

7 Bibliography

- Development of a 3D Graphics Engine (Kassing)
- Modular Architecture for Computer Game Design (McNeill)
- Multi-threaded Game Engine (Tulip, Bekkema, Nesbitt)
- Interactive 3D Geometry in OpenGL (Welsh)
- FROG: The Fast & Realistic OPENGL Displayer
- Simple and rapid collision detection using multiple viewing volumes (Fan, Wan, Gao)
- Some code used from "http://www.tjhsst.edu/ dhyatt/superap/opengl.html"