Comp 394: Interactive Computer Graphics
Assignment 5: Crayon Physics

Handed out: Fri 4/17
Due: Fri 5/1 at 3:30pm

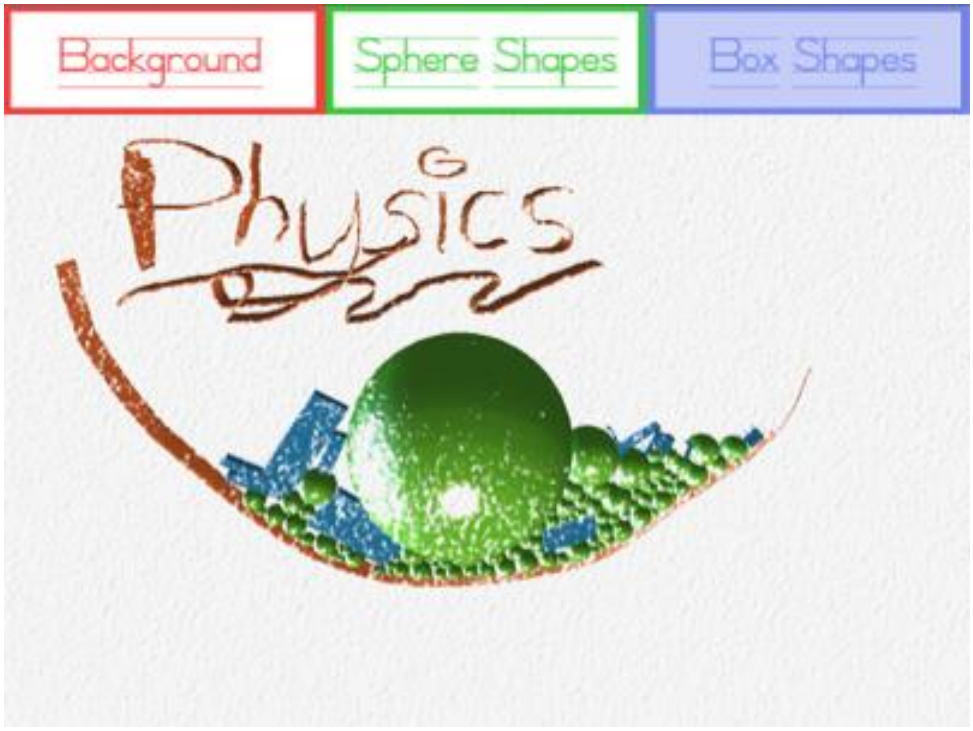

Figure 1: In this assignment, you'll make a simple interactive game that simulates physics of hand-drawn shapes. The game is inspired by www.crayonphysics.com.

1. Introduction

So far in the class, we've learned how to use a professional-level computer graphics toolkit, G3D. Game studios, movie studios, and others programming sophisticated interactive computer graphics applications regularly combine the graphics toolkits they use with other toolkits to support physics simulation, sound, and other effects. In this assignment, we'll do the same thing – you'll learn how to combine G3D with a Physics engine, and in the process, you'll create a program that you should feel quite pleased showing off to friends and family!

Physics engines appear in graphics programs of all types, ranging from games such as Angry Birds to advanced modeling tools such as Maya. For simulating 3D physics, one of the most popular toolkits today is Bullet3D (http://bulletphysics.org). In this assignment, we're going to simplify the physics a bit by limiting ourselves to physics within a plane (i.e., 2D physics), so we'll work with the Box2D physics toolkit, which is probably the current best / most popular 2D physics engine.

Learning Goals

In completing this assignment, your goal should be to learn:

- How to create an interface between a high-end graphics toolkit and another helpful toolkit, e.g., a physics engine.
- How to link physics simulation with graphics, including making simplifying assumptions about the physics when appropriate, e.g., mixing 2D simulation with 3D graphics.

2. Overview of Requirements

For this assignment you'll be hooking up the Box2D physics engine to the G3D-based support code we provide for you. The support code gives you a fairly full-featured application out of the box. Most notably, as shown in Figure 1, the interface for the application includes a menu at the top of the screen. You click the menu to set the current drawing mode to either Background, Sphere Shapes, or Box Shapes. Drawing a background shape creates a static object in the scene, like a ramp. Drawing a sphere or box shape creates a dynamic object that should react to gravity and all the other objects in the scene. We have already implemented enough of the interface to allow you to create background shapes, boxes, and spheres and to render them using GLSL shader programs that make them look like they were drawn with crayons. What remains for you to do is to hook up the physics simulation to make the objects move.

3. Specific Requirements and Suggested Approach

Link your G3D program with Box2D: For this assignment, you'll be working with Box2D, so you need to change the way that you build your G3D program in order to make it include the Box2D library header files and link with the Box2D library file. Instructions to help you do this on Windows and OSX systems will be posted to Moodle.

Setup the Physics Simulation: The graphics for this application will be drawn in 3D so that we can get some nice shading effects on the spheres and other objects. However, we're making the simplifying assumption that all the physics simulation happens within a plane. So, when you setup your Box2D simulation, you should use circles rather than spheres and rectangles rather than cubes, etc. Box2D should be set to have gravity of 9.8 m/s/s in the down direction. Beyond that, your main challenge will be to figure out how communicate between Box2D and G3D. For example, when a new sphere is added to the scene in G3D, you'll need to add a corresponding object to your physics simulation. Then, you'll need to keep the physics and graphics in sync by advancing the physics simulation once each graphics frame – a good place to do this is inside G3D's `onSimulation()` function.

Interaction: By using the interface we provide, the user should be able to create any of a number of different shapes in the world. The sphere and box shapes should be dynamic, and the background shapes should be static. You can assume that the

user will *not* draw self-intersecting background paths. Pressing 'q' should reset the simulation to the same state it was in when the program is started, i.e., it should delete all the objects in the scene.

4. Wizards

All of the assignments in the course will include great opportunities for students to go beyond the requirements of the assignment and do cool extra work. I don't offer any extra credit for this work – if you're going beyond the assignment, then chances are you are already kicking butt in the class. However, I do offer a chance to show off… while grading the assignments I will identify the best 4 or 5 examples of people doing cool stuff with computer graphics. We'll call these students our "wizards", and after each assignment, the students selected as wizards will get a chance to demonstrate their programs to the class!

Ideas for wizardry include, but are certainly not limited to:

1. Setting up a challenge for the user, perhaps in the spirit of Crayon Physics or other physics-based games.
2. Using a three-dimensional physics engine such as Bullet 3D to have fully three-dimensional physics.
3. Using a library such as the $1 gesture recognizer to allow the user to create circles by drawing a circle, boxes by drawing a box, and other things by drawing something else. Implementations are available by searching for the phrase "$1 gesture recognizer". The original paper for this library is "Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes".
4. Supporting dragging, editing, deleting, or modifying existing objects.
5. Improving the rendering to look (in some way) more impressive.

If your ideas for going beyond the requirements would make your code more difficult for me to grade, please help me by submitting a non-wizardly version of your assignment it its own sub-folder of your moodle submission.

5. Support Code

The moodle page also has a download link for support code to help you get started. The support code for this assignment is a simple G3D app based on the tinyStarter example in the G3D10/samples directory.

The support code defines a program structure and everything you need to read and parse the bunny model file. **To make locating data files simpler, I have a header file called config.h that contains absolute paths to your data files. You should edit this file with the full path (e.g., "C:\Users\bjackson\assignment4" or "/Users/bjackson/assignment4/") where you've placed the data files.** I will modify this file appropriately when grading your assignment.

6. Handing In

When you submit your assignment, you should include a README file. This file should contain, at a minimum, your name, partner's name, and descriptions of design decisions you made while working on this project. If you attempted any "Wizard" work, you should note that in this file and explain what you attempted.

When you have all your materials together, zip up the source files, README, and data directory and upload the zip file to the assignment hand-in link on our Moodle site, which you will see on our calendar page for the day that the assignment is due.