CG-driven Simulation of Energy, Friction, Momentum, and Collisions as a Teaching Tool

Erik Hansen

June 1, 2004

Abstract

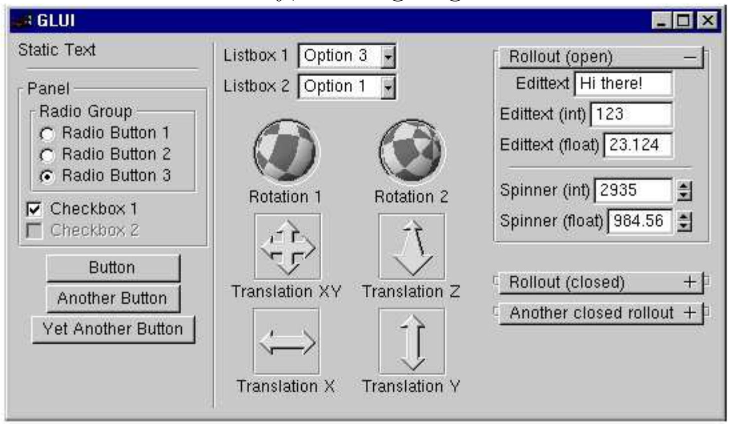
NetLogo provides the ideal interface to teach physics students the basics of 2-D collisions. The NetLogo user interface is entirely graphical and very clear. The user can easily manipulate a number of variables through sliders and switches and observe the results on real-time graphs and a graphic display.

1 Introduction

Modern physicists and engineers rely heavily on computers to preform mathematical calculations. Some of the more complicated physical concepts can be difficult for high school students to grasp without a visual model for them to observe. Psychologists estimate that over 90The computer software industry has produced many programs that attempt to graphically teach math and language to aspiring mathematicians and linguists. This program has been created to assist high school physics teachers in teaching their classes basic 2-Dimentional collisions. The program accurately models concepts of friction, mass, velocity, kinetic energy, and conservation of momentum.

2 Initial Study

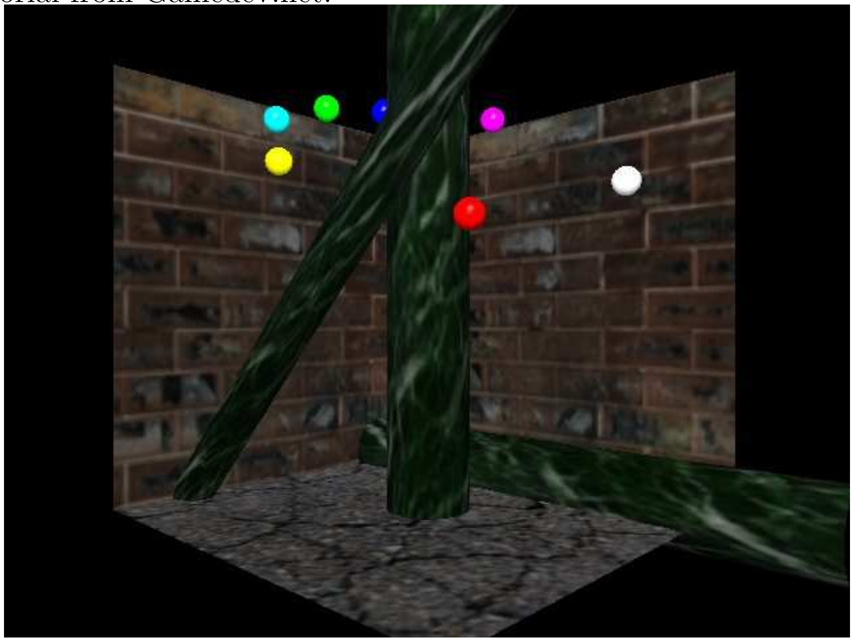
My initial plan was to model collisions and momentum in a 3-Dimentional environment. Using my C++ and OpenGL skills from previous years, I hoped to create an interface in OpenGL that would allow the user to interact with the environment in an easy-to-understand way and simply observe the effects of such interaction. I knew that I would have to study OpenGL syntax and code a great deal – all I could do from my time in Supercomputer Applications and Elements of Artificial Intelligence was render a bunch of shapes and move the camera a bit via keyboard commands. /par For my new plan, I intended to have two windows – one that displayed the environment, and another that provided the user with buttons, sliders, switches, and bubble selections that allowed him or her to interact with the environment. Essentially, I was going for a user interface that looked something like this:



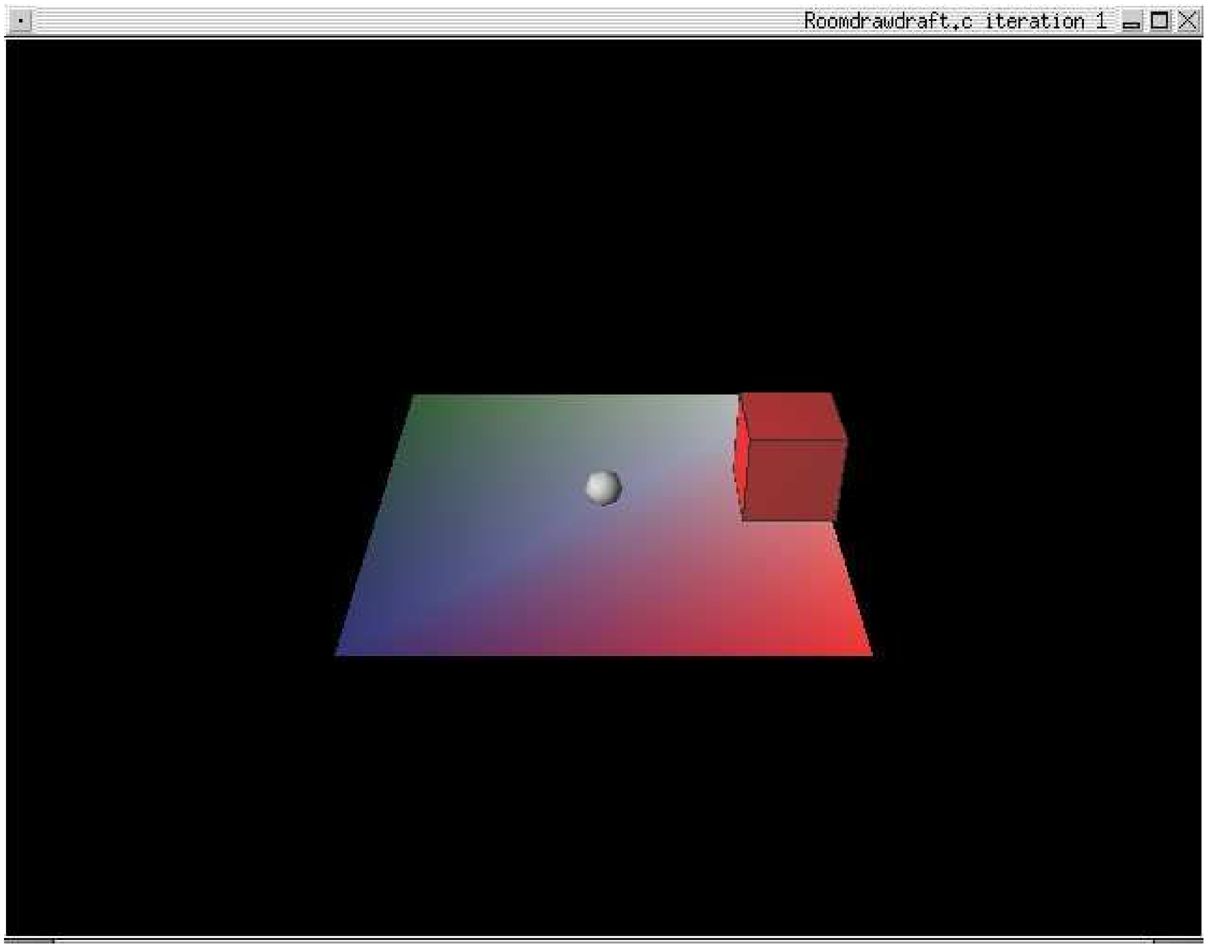
Listbox 1 Option 3
Listbox 2 Option 1
Edittext Hi there!
Edittext (int) 123
Edittext (float) 23.124
Spinner (int) 2935
Spinner (float) 984.56

Of course, the actual boxes and radio buttons would be replaced with variables like ”velocity,” ”mass,” ”Coefficient of Friction,” and any other mathematical variables I might need to complete my project.

The environment would at first glance resemble the output from the OpenGL ’collision’ tutorial from Gamedev.net:



I spent more than two months working toward this goal. I proceeded to work my way through the gamedev.net tutorials online and learned all about polygon shading, proper lighting techniques, camera placesment, texture mapping, and blending. Collision detection, however, proved to be more than I could handle. The gamedev.net tutorial on collisions (the program pictured above) was way over 15 pages of code. I coudn’t understand all the variables and functions. I asked help from two esteemed classmates, David Raber and Raphael Mun, but they couldn’t seem to help either. I eventually managed to construct a small environment, but the actual animation, let alone calculations, remained unfinished:



Then, Mr. Latimer found a development program called NetLogo created by students and faculty at Northwestern University. He highly encouraged that some of the class experiment with it some, and I decided to give it a shot. NetLogo is in many ways much simpler to use than C++ – the first iteration of my project in NetLogo was only two pages long, yet accomplished much more than my OpenGL iteration. This is because much of the 'programming' in NetLogo is actually accomplished through graphical means. NetLogo, due to its simplistic nature, could not render 3D environments – and even 3D objects could only be displayed at extreme drain to a system's resources. Even so, I found NetLogo to be the ideal development tool. My project would have to be done in 2D, but this allowed me much more freedom in the variables I could monitor and greatly simplified the equations I had to use.

3 Background and Theory

My ultimate goal was to create a program that could teach collisions and friction to non-physicists. I hoped to accomplish this through a simple graphic interface and an easy-to-understand environment display. Of course, in order to teach physics, I had to learn physics.

Since this was a 'senior' techlab project, and Thomas Jefferson students are required to take physics junior year, I had already learned much about momentum ad collisions. There are three equations that govern the movement of objects in a one-dimentional environment:

V = V0 + (X - X0) / T X = X0 + V0T + 0.5*A*T2 V2 = V02 + 2*A*(X - X0)

V=Velocity (meters/second)

V0 = Initial velocity (meters/second)

X = X position (meters)

X0 = Initial X position (meters)

T = Time (seconds)

A = Acceleration (meters/second/second)

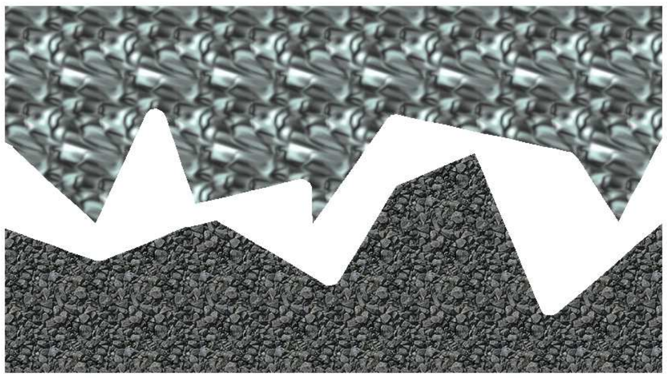
These equations, however, only model one-dimentional motion. Motion in two dimensions requires the addition of a Y variable. The Y variable is essentially similar to the X variable, except that wheras X keeps track of the position of an object in the X plane of motion, Y keeps track of an object in the Y plane of motion. The two position equations we will need to use, then, are:

X position = (initial X position)+(X velocity * time interval) + (0.5*Xacceleration * time)

Y position = (initial Y position)+(Y velocity * time interval) + (0.5*Yacceleration * time)

Using these equations, a mathematician (and, therefore, a computer) can accurately calculate an objects velocity, position, or acceleration given the rest of the variables. This program, however, involves more complicated calculations. The above equations do not account for friction.

Friction is a unique 'force' in that it can only exist in the presence of another force. Friction always acts in the opposite direction of the applied force, and always to a specific degree depending on the material of the surface. Friction is caused by indentations on two flat surfaces on an almost microscopic level:



Microscopic Diagram of Two 'Flat' Surfaces

Friction resists force. If one of the plates in the above diagram were pushed to the right, then nooks and edges of the other plate would exert force to the left. Similarily, if one plate was pushed to the left, then the indentations on the other plate would resist the motion by pushing to the right. Different materials have stronger or weaker friction than others. For instance, ice is very smooth and hardly resists motion at all. On the other hand, rubber has many microscopic mountains and valleys and hence can grip surfaces very well.

An object's friction is dentoed by the 'coefficient of friction,' or mu. The higher the coefficient of friction, the stronger grip a material has. Here is a table of some common materials and their respective coefficients of friction:

Surface Material: Coefficient of Static Friction:

Ice: 0.1

Glass: 0.3

Polished Wood: 0.35

Asphault (wet): 0.4

Human skin: 0.6

Asphault (dry): 0.6

Rubber: 0.8

For my project, I wanted the user to be able to simulate collisions on just about any surface. So, I made the surface of the environment variable – the user can simply adjust the coefficient of friction directly. I even allowed the user to simultaneously simulate two seperate surfaces side-by-side. There are two friction sliders, one for the dark area ('darkfriction') and one for the light area ('lightfriction'). This way, the user can accurately simulate collisions on two different surfaces at once. Also, the multiple surface types allow the user to observe how sliding over one surface affects kinetic energy and momentum as opposed to sliding over another. If the puck is losing a certain amount of energy every second due to friction, then as the puck moves to an area with greater friction, the energy graph will become steeper. In this way, the user can easily see how different friction values affect the rate of loss of velocity, energy, and momentum.

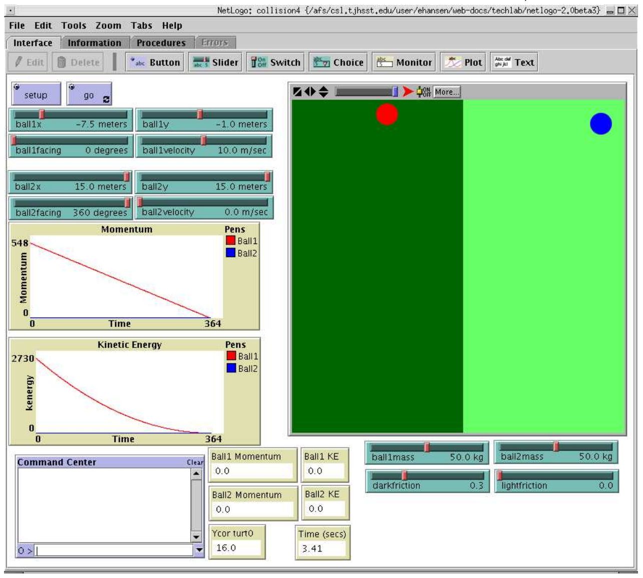
The coefficient of friction is called a coefficient because it is a multiplier for the magnitude of the force caused by friction. The force exerted by friction is equal to mu * normal force.

Mu is the coefficient of friction. The normal force is the force between the two surfaces. So, there are two ways one can affect friction: by increasing the coefficient of firction, or increasing the normal force. The normal force and be increased by making the object on top more massive, so it presses down harder on the object beneath it.

For example:

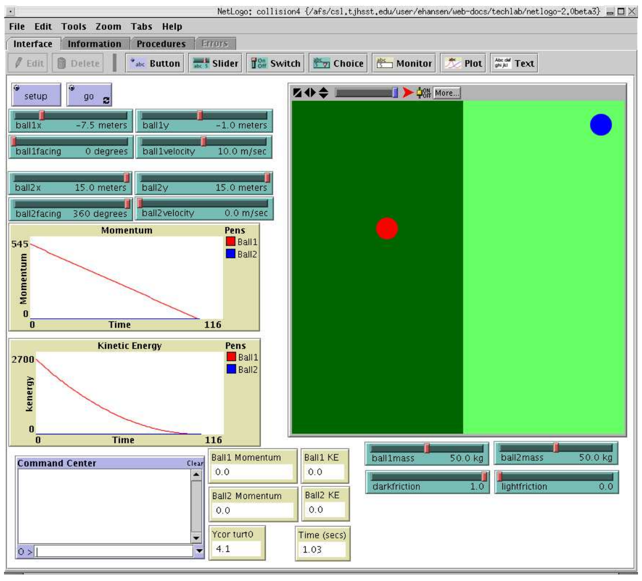
Two surfaces have a mu of 0.4 (wet asphault) and a normal force of 4,000 newtons. The force of friction would then be 0.4 * 4000 = 1,600 newtons. If the same scenario were to occur on dry asphault, however, our equation would become 0.6 * 4000 = 2,400 newtons – much stronger. Finally, if were were to take the original equation and change the normal force between the two surfaces to 6000 newtons, then our equation would become 0.4 * 6000 = 2,400 newtons. Because collisions.nlogo has sliders to adjust both the coefficients of friction and the mass of each puck, the user can easily simulate any combination.

To see coefficients of friction in action, observe the following two screenshots.


NetLogo: collision4 {/afs/csl.tjhsst.edu/user/ehansen/web-docs/techlab/netlogo-2.0beta3}
File Edit Tools Zoom Tabs Help
Interface Information Procedures Errors
Edit Delete Button Slider Switch Choice Monitor Plot Text
setup go
More...
Momentum
Pens
Ball1
Ball2
548
Momentum
0
0
Time
364
Kinetic Energy
Pens
Ball1
Ball2
2730
kenergy
0
0
Time
364
Command Center
Clear
0 >

ball1x -7.5 meters
ball1y -1.0 meters
ball1facing 0 degrees
ball1velocity 10.0 m/sec
ball2x 15.0 meters
ball2y 15.0 meters
ball2facing 360 degrees
ball2velocity 0.0 m/sec
Ball1 Momentum 0.0
Ball1 KE 0.0
Ball2 Momentum 0.0
Ball2 KE 0.0
Ycor turt0 16.0
Time (secs) 3.41
ball1mass 50.0 kg
ball2mass 50.0 kg
darkfriction 0.3
lightfriction 0.0

A friction coefficient of 0.3.


NetLogo: collision4 {/afs/csl.tjhsst.edu/user/ehansen/web-docs/techlab/netlogo-2.0beta3}
File Edit Tools Zoom Tabs Help
Interface Information Procedures Errors
Edit Delete Button Slider Switch Choice Monitor Plot Text
setup
go
More...
Momentum
Pens
Ball1
Ball2
545
Momentum
0
0
Time
116
Kinetic Energy
Pens
Ball1
Ball2
2700
Kenergy
0
0
Time
116
Command Center
Clear
0 >

ball1x -7.5 meters
ball1y -1.0 meters
ball1facing 0 degrees
ball1velocity 10.0 m/sec
ball2x 15.0 meters
ball2y 15.0 meters
ball2facing 360 degrees
ball2velocity 0.0 m/sec
Ball1 Momentum 0.0
Ball1 KE 0.0
Ball2 Momentum 0.0
Ball2 KE 0.0
Ycor turt0 4.1
Time (secs) 1.03
ball1mass 50.0 kg
ball2mass 50.0 kg
darkfriction 1.0
lightfriction 0.0

A friction coefficient of 1.0.

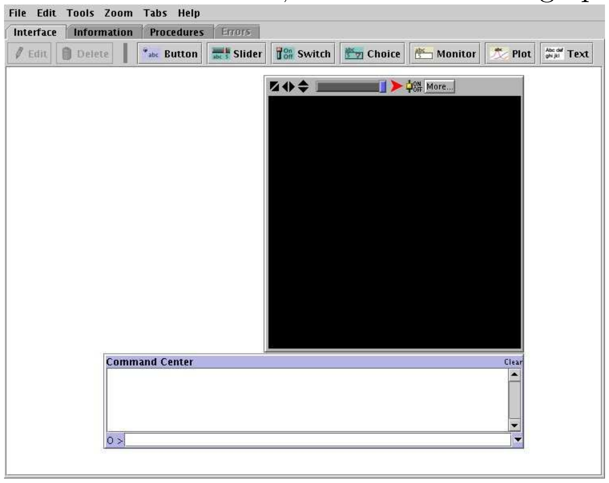
When comparing the two images, take particular care to notice the box labeled 'time' in the bottom center of the display. For the screenshot with 1.0 friction, it only took the ball 1.03 seconds to come to a stop. When the surface's friction coefficient was 0.3, the ball took 3.41 seconds to come to rest! This is over three times as long. The graphs look similar, but friction obviously has a huge effect on the motion of flat surfaces. Because the Coefficient of Friction is just that – a coefficient – it is relatively simple to program. Every time interval (0.001 seconds, according to the program), the program subtracts a certain amount from the puck's velocity. This amount is based on friction. Obviously, the greater the friction (and therefore the friction coefficient), the more subtracted velocity. If you are driving a car on an asphault road and hit the brakes, you will come to a stop much faster than you would if you were driving on a frozen pond!

4 Testing

As previously mentioned, my research project started out as a three-dimensional concept. Testing the project in this form is was very slow and tedious, as the only experience I had had in three dimensional programming was my junior year supercomputer applications course (also taught by Mr. Latimer). I had to constantly scour the internet for more information on whatever subject I was working on at the time. I had to learn three dimensional shapes, animation, camera placement and movement, coloring, time, and how to implement mathematical equations. Since I had done some of those before, I thought that it would not be all that much work to learn the rest. Either I overestimated my own abilities or I underestimated the material, because that was not the case. I spent a good couple of months working toward a useable OpenGL environment and input device, but my efforts never panned out, even with the help of www.gamedev.net and a number of my classmates.

So, I began to search for other mediums that I could use for my program. Most three-dimensional graphic engines seemed at least as difficult as OpenGL to use, so I had no luck there. I had to limit myself to languages that could be used for linux programs – since all the computer systems lab computers run linux, I couldn't very well use a windows-based code. Then, my instructor Mr. Latimer announced that some graduate students from Northwestern University were working on a new 'language' called NetLogo that shared many features with an old drawing program called "LogoWriter." NetLogo is very user-friendly, enabling both graphics and textual code to be used when writing a program. Using such a simple-to-code language gave me some dirty looks from the more elite of the computer systems lab, but I went ahead and did so anyway.

Here is what a basic, non-edited NetLogo program interface looks like:



The most obvious features are the two large boxes, one black and one white. The black box is the display window which shows what happens during your program. Note that even though your program starts with a display window, it does not HAVE to be utilized – you can write programs that involve no sort of graphical output whatsoever. But, NetLogo was designed to use graphics. An entirely text-based program can easily be created using C++ and would not take advantage of NetLogo's unique output design.

Virtually all graphics programs, no matter what language they are written in, use a large repeating loop. Most NetLogo programs use a 'go' button to begin their main loops. For NetLogo and similar programs, the user initiates the main loop simply by hitting the 'go' button (refer to previous screenshots). The loop will then repeat until the end conditions are met. In my program, the end conditions are two fairly simple 'if' statements, reprinted here:

ask turtle 0 [if(distance-nowrap turtle 1 ¡ 2.25) [set velocity 0 ask turtle 1 [set velocity

```
0 ] ] ]
```

and

```
if (velocity ¡ 0) [ set velocity 0 ]
```

The first one merely asks how close one puck is to the other. If they are close enough to overlap, then they are touching, and a collision has occurred so I must stop the simulation. The second one is simple to understand in practice, but the concept needs some explanation. The way I have set up my equations, each puck's velocity decreases at every time increment based on friction and mass. Since I subtract something from the velocity every hundredth of a second, the velocity WILL eventually end up less than zero. If the simulation were to continue at this point, the puck would begin to accelerate in the opposite direction – which, obviously, makes no sense. When you slide a book accross a tabletop it does not turn around and come back to you, it merely stops.

5 Expansion and Further Study

Despite my progress with this project, I did not manage to finish as much as I wanted. One especially interesting trait was never implemented – what exactly happens to the momentum and kinetic energy of the two pucks AFTER they collide? Adding in code to dictate what happens after a collision would further expand upon my project and be an even greater teaching aid to high school students and teachers.

6 Sources

Simple animation http://www.cyberloonies.com/animation.htm C and C++ faqs http://www.programmers

OpenGL resources http://www.frii.com/ martz/oglfaq/gettingstarted.htm OpenGL official

site http://www.opengl.org 50 OpenGL tutorials, from absolute newbie to advanced users –

VERY useful! http://nehe.gamedev.net/ Possible similar program at Cornell http://www.mae.cornell.edu/c

Adding a Graphical User Interface http://www.cs.unc.edu/ rademach/glui/ Quick Reference

to gl/glu/glut Funcion Libraries http://www.tjhsst.edu/ rlatimer/supercomp/resourceLinks.html

Netlogo http://ccl.northwestern.edu/netlogo/ Kinematics Overview wlcnt2.wlc.edu/intranet/faculty$_staff/$

$//www.csail.mit.edu/research/abstracts/abstracts03/index.html$

Fourth Edition Physics By Douglas C. Giancoli

7 Appendix 1: Code

NOTE: NetLogo's code is unique and hard to convert into complete textual form. As I have stressed multiple times in my paper, NetLogo is mostly based on graphical code. Some textual code has been written by me (only a few pages' worth) while the rest is automatically generated based on the buttons and slider and graphs you create in the NetLogo construction interface. I have clearly marked the where the code manually written by me ends and the automatically-generated code begins. Since the automatically-generated code was generated by the computer, I cannot comment it. All the code can be found on seperate pages accompanying this document on the project website.