Simulating the Basic Effects of an Explosion on Objects

Derrick Huey

Abstract

This paper describes a physics animation simulating the basic linear rigid body dynamics effects of an explosive force on objects in a scene.

Introduction

As video games progress, the physics programming behind them gets better and better. Realistic physics provides not only pretty eye-candy for players to view, but also creates situations for the player that actually feel like they could happen in real life. A bullet can shatter glass. An explosion might knock obstacles out of the way. This paper presents a very basic physics representation of the effects an explosion might have on objects around it. This simulation aims to provide a basic understanding of the principles of linear rigid body dynamics, projectile motion, and friction.

Background

Rigid body dynamics describes the motions of rigid bodies that occupy a specific amount of space. That is to say, it deals with objects that have a fixed volume and specific shape [1]. While explosions tend to have enough force to break things, the pieces are rigid bodies in their own right.

Projectile motion deals with the motion of objects upon which the only force acting is gravity [3]. Once an object has been launched into the air by an explosive force, only gravity is acting upon it and thus it is a projectile. The object's lateral movement remains constant while the vertical motion is determined by the initial velocity and gravity.

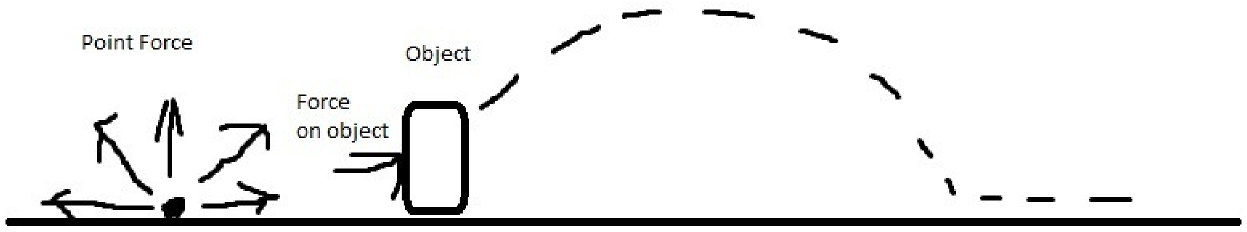


Figure 1: Sketch showing planned process

Method

This program only focuses on the linear aspects of rigid body dynamics. The principle behind this is that each object has a "center of mass". In order to determine the general motion of the object influenced by forces, you simply add up the forces acting on each point of the object and divide this sum by the object's total mass, resulting in the object's acceleration. From there, you can integrate over time to get the body's new velocity and position [1].

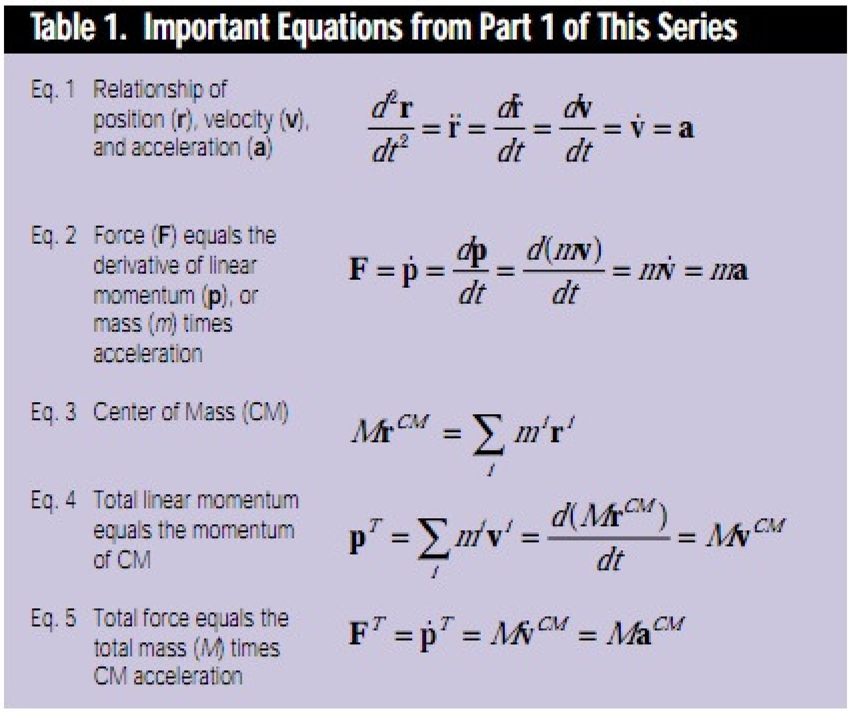
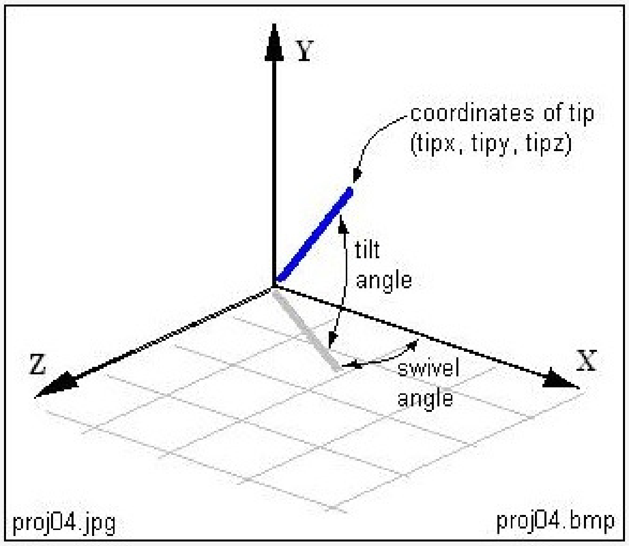


Table 1. Important Equations from Part 1 of This Series

Eq. 1	Relationship of position (**r**), velocity (**v**), and acceleration (**a**)	$\frac{d^2\mathbf{r}}{dt^2} = \ddot{\mathbf{r}} = \frac{d\dot{\mathbf{r}}}{dt} = \frac{d\mathbf{v}}{dt} = \dot{\mathbf{v}} = \mathbf{a}$
Eq. 2	Force (**F**) equals the derivative of linear momentum (**p**), or mass (*m*) times acceleration	$\mathbf{F} = \dot{\mathbf{p}} = \frac{d\mathbf{p}}{dt} = \frac{d(m\mathbf{v})}{dt} = m\dot{\mathbf{v}} = m\mathbf{a}$
Eq. 3	Center of Mass (CM)	$M\mathbf{r}^{CM} = \sum_i m^i \mathbf{r}^i$
Eq. 4	Total linear momentum equals the momentum of CM	$\mathbf{p}^T = \sum_i m^i \mathbf{v}^i = \frac{d(M\mathbf{r}^{CM})}{dt} = M\mathbf{v}^{CM}$
Eq. 5	Total force equals the total mass (*M*) times CM acceleration	$\mathbf{F}^T = \dot{\mathbf{p}}^T = M\dot{\mathbf{v}}^{CM} = M\mathbf{a}^{CM}$

Figure 2: Equations relevant to linear rigid body dynamics

Now we need to go into the actual force applied to the points of our objects. To model my explosive force, I used a point-force radiation. That is to say, the force radiates out in all directions from the point (0, 0, 0). I had difficulties finding a suitable equation to calculate the force at a given distance from the originating point, and thus settled on using an inverse distance-squared relationship so that the force tapers off the farther away you are. Because I was dealing with vectors, simply multiplying the original force by the inverse distance-squared was not enough. I needed to determine the x, y, z components of the force. This was done using the principles described in Figure 3. The y component of the force is based on the tilt angle while the x and y components are based on the projection of the vector onto the x-z plane [4].



The y-coordinate of the cannon tip can be found using the following:

```
tiltrads = (tiltdegs / 360) * 2 * pi
tipy = LengthCannon * sin(tiltrads)
```

The length of the projection of the shaft of the cannon onto the x-z plane is found using the following equation:

```
projection = LengthCannon * cos(tiltrads)
```

The x- and z-coordinates of the cannon tip can be found using the following:

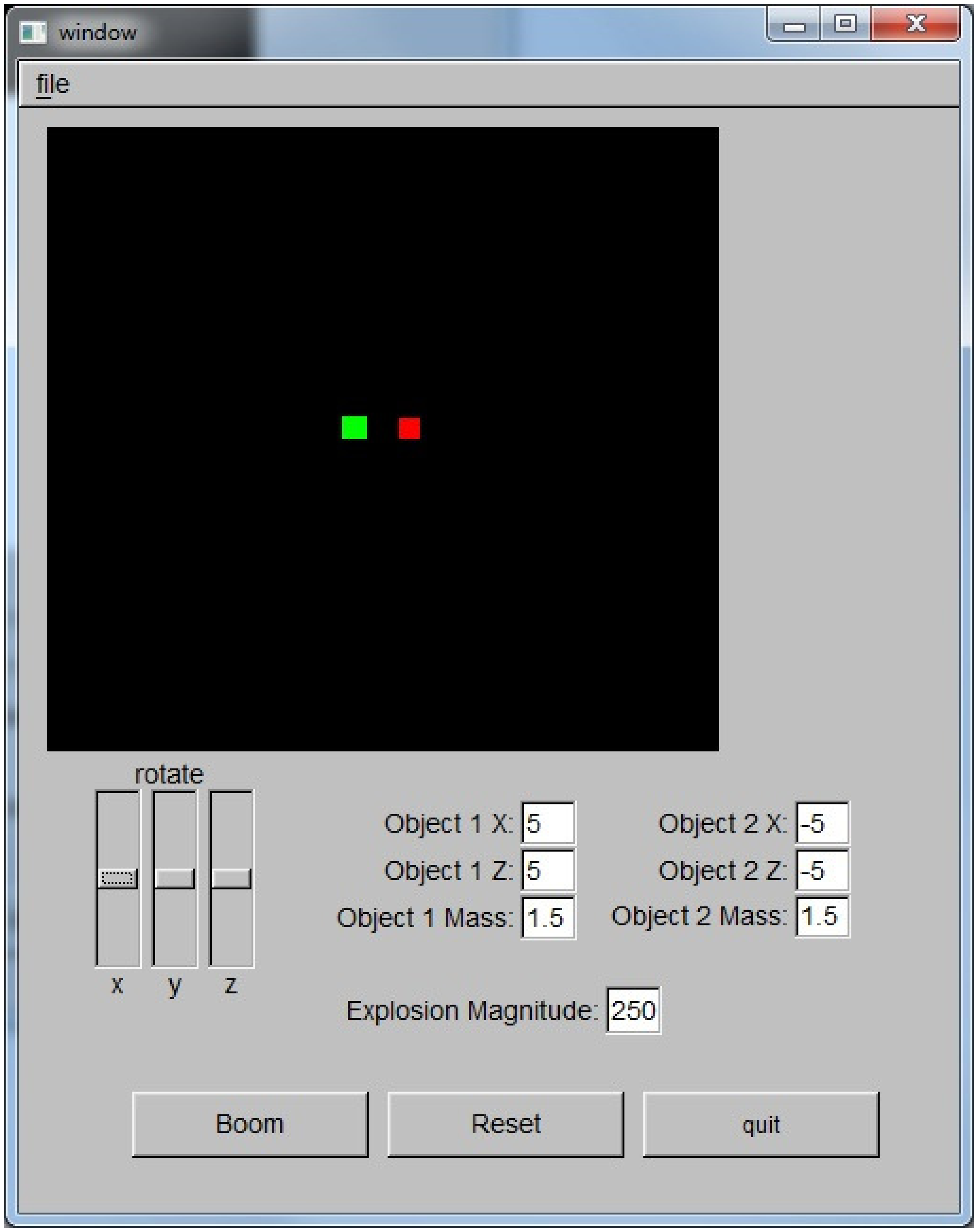
```
tipx = projection * cos(swivlrads)
tipz = projection * sin(swivlrads)
```

Figure 3: Details behind 3D projectile motion used to calculate the force vector components in this program

The force was calculated and applied to each object over the course of one second at intervals of 10 milliseconds. Other forces involved were gravity (while the object is in the air) and friction (while the object is on the ground). To model the effects of friction, the opposite of the object's velocity is multiplied by a user-set friction coefficient between 0 and 1.

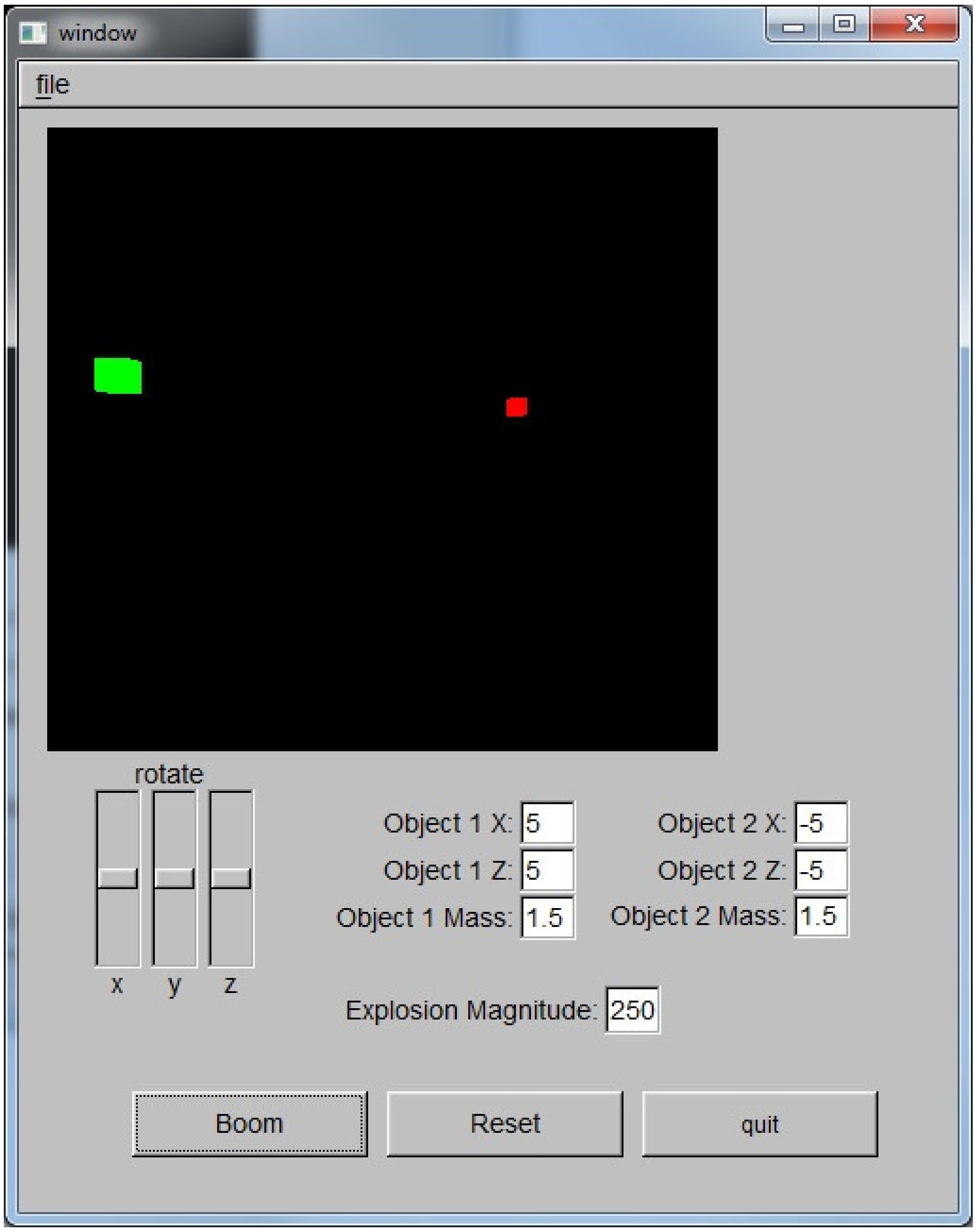
Results

The main issues I ran into while writing this program revolved around the calculation of the angles involved in the force vector component calculation described in Figure 3. When these values were incorrect, the result led to objects moving in the wrong directions and the wrong amount. The main confusion arose from converting the 2D principles projectile motion (x-component = r*cos(theta), y-component = r*sin(theta) to 3D. There was also the issue that the arcsin and arctan functions only returned values between certain ranges, which needed to be adjusted depending on what quadrant of its respective plane a point was in.



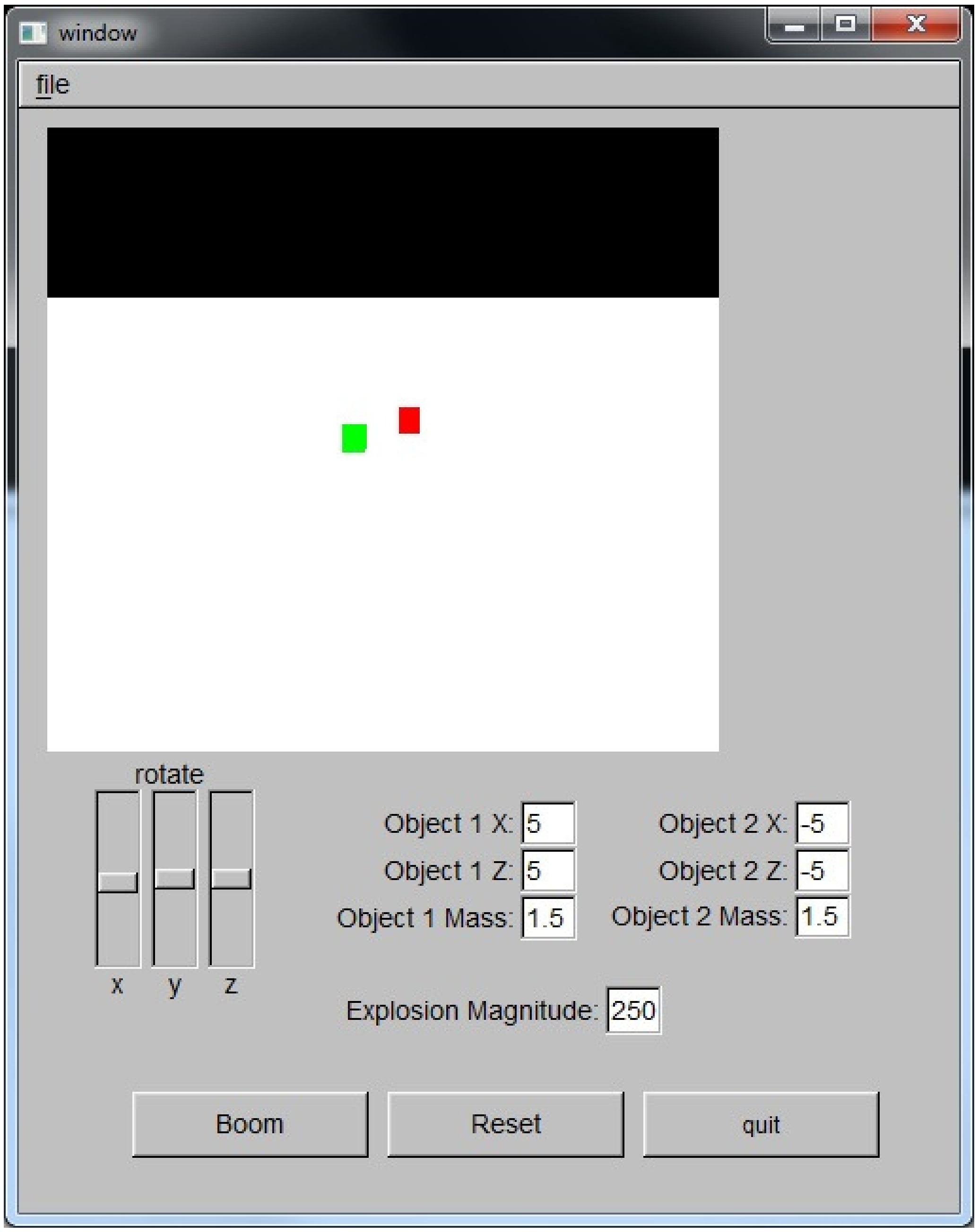
Object 1 X: 5
Object 1 Z: 5
Object 1 Mass: 1.5
Object 2 X: -5
Object 2 Z: -5
Object 2 Mass: 1.5
Explosion Magnitude: 250

Figure 4: Side view, original positions



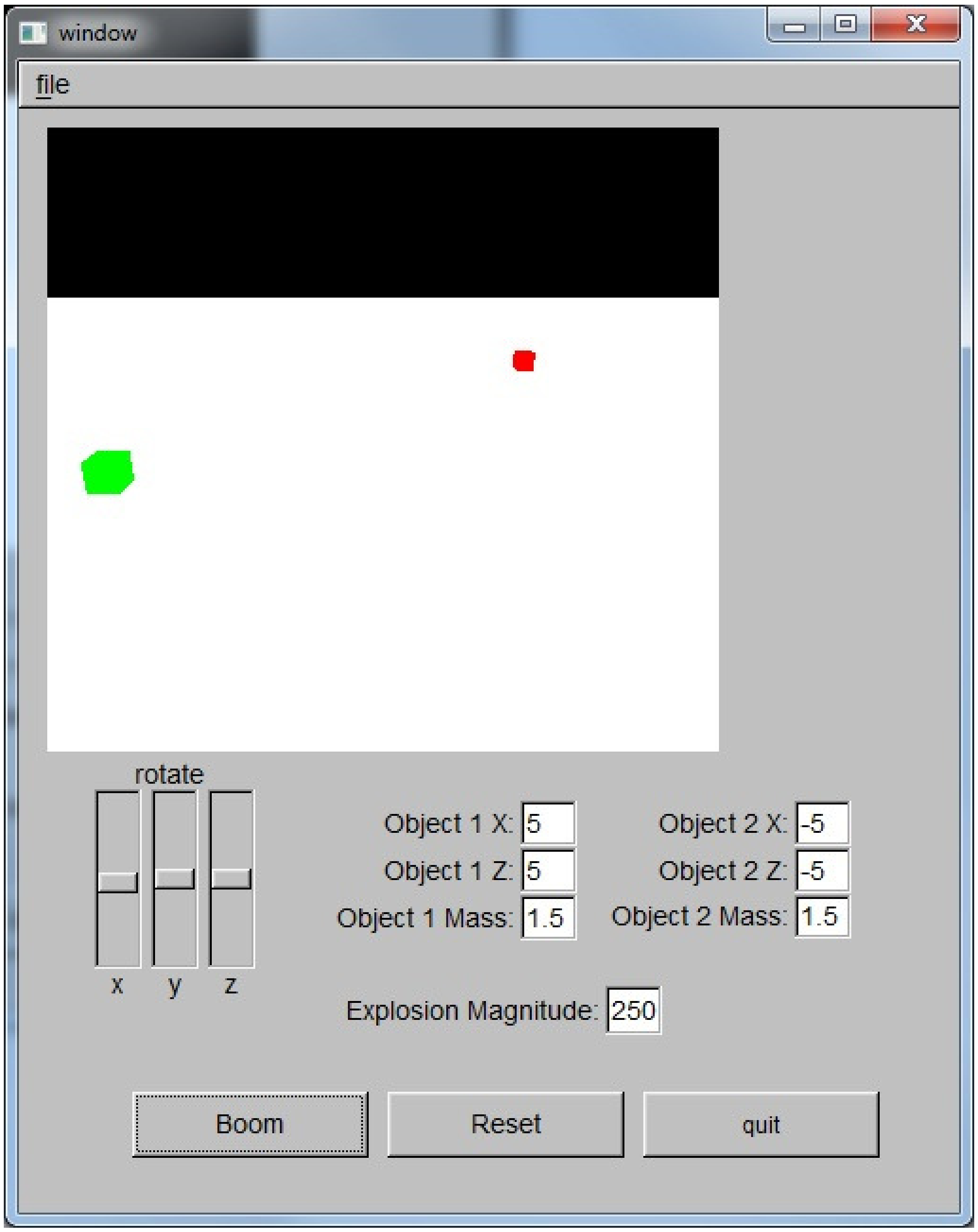
Object 1 X: 5
Object 1 Z: 5
Object 1 Mass: 1.5
Object 2 X: -5
Object 2 Z: -5
Object 2 Mass: 1.5
Explosion Magnitude: 250

Figure 5: Side view, objects in motion



Object 1 X: 5
Object 1 Z: 5
Object 1 Mass: 1.5
Object 2 X: -5
Object 2 Z: -5
Object 2 Mass: 1.5
Explosion Magnitude: 250

Figure 6: Slant view, original positions



Object 1 X: 5
Object 1 Z: 5
Object 1 Mass: 1.5
Object 2 X: -5
Object 2 Z: -5
Object 2 Mass: 1.5
Explosion Magnitude: 250

Figure 7: Slant view, objects in motion

Related Work

Physics calculations and simulations exist in all modern video games, though obviously with more realistic animations than those presented in this paper and program. Scientists and engineers are constantly working with simulators in order to improve various safety procedures as well. A specific work that demonstrates the angular aspects of rigid body dynamics that would be added in future work is an example program written by Chris Hecker [2].

Future Work

As described, currently the program only deals with the linear components of rigid body dynamics. This shows us the general motion of the object as a whole. However, it lacks the rotational aspects and collision response aspects of rigid body dynamics, which are responsible for the sort of "tumbling" action we see in real-world scenarios. The next step would be to add these aspects to the program to depict a more accurate picture.

References

[1] Chris Hecker. Physics, The Next Frontier. *Game Developer Magazine (Oct/Nov 96).* Pages 12-20, available from http://chrishecker.com/images/d/df/Gdmphys1.pdf

[2] Chris Hecker. OpenGL 3D Physics Sample, available from http://chrishecker.com/images/3/33/Gdphys3d.zip

[3] The Physics Classroom. Projectile Motion, available from http://www.physicsclassroom.com/class/vectors/U3L2a.cfm

[4] Tom Nally. Projectile Motion in 3D Space, available from http://babek.info/libertybasicfiles/lbnews/nl130/proj3d.htm