Research and Development of a Physics Engine

by Timmy Loffredo

Feb 15, 2006

1 Abstract

As the processing power of modern computers grows, it becomes more and more feasible to create accurate, graphical simulations of three dimensional physics. In the field of computer games, for example, powerful 3D physics engines are starting to become standard practice. This project aims to research modern physics modeling techniques and create a rudimentary, but powerful, physics engine. Most games focus on rigid body dynamics, a small but important subset of all physics, and so does this project; but it also makes a foray into other areas like cloth simulation. The conclusion of this project is that there are significant computation time drawbacks to a strong game physics engine, and a balance must be struck between speed and realism.

2 Introduction

2.1. Purpose

The purpose of this project is to create a three dimensional physics simulator. The primary goal is to create a simulator that looks real. The main application of this project would be as a physics engine for games. The simulator should at least be able to run something like a bowling game, where the only pertinent physics are collisions, gravity, and rolling along a surface. It's a good topic for the Computer Systems Lab because the physics part is relatively easy - it's getting a computer to do it and graphically display it that is difficult.

2.2. Scope of Study

The scope of the project, by the project's nature, is highly variable. The project will start with a simple part, like kinematics for cubes, and then I will grow the project into collisions for any polygonal solid, time permitting. Most, if not all, of the project will be in the physics subfield of rigid body dynamics. If time is leftover, I will expand the project into joints (like on a crane) or curved surfaces.

2.3. Background

Game physics is a blossoming field which is likely to become a big focus of computer games very soon, when technology makes it appropriate to do intense calculations. Most games currently have a semblance of physics but do not actually use a complex physics engine, they just make common things react the way you would expect them to and leave it alone from there.

A series of 4 articles by Chris Hecker [3,4,5,6] present a simple method for creating a 3D physics simulator capable of collisions for rigid, polygonal solids. This project will loosely follow his methods, along with the methods of Martin Baker [2], who maintains a detailed website for computational physics.

An article by Kurt Miller [7] about basic collision detection and planar techniques will be used as an outline for this project's collision detector. The article is useful in explaining how to computationally decide whether a point is colliding with a plane. Point-plane collisions are really the building blocks of more complicated polygonal collision detectors.

3 Development

3.1 Theory

Every physics engine needs an integrator for determining velocity from acceleration and position from velocity. The simplest of integrators, the Euler integrator, often leads to unstable loops of energy gain in cyclical systems. One step above that integrator is the leap frog integrator, which has an initial offset that gives its estimates much higher precision. Even more precise than that (but computationally much slower) are the Runge-Kutta algorithms, which allow for any degree of precision you can specify. I have chosen to use the leap frog integrator as a balance between speed and precision.

There are many ways to represent 3D orientation. One such way is with Euler angles (pitch, yaw, roll.) Instead of using Euler angles, I have chosen to use a 3x3 special orthagonal matrix because (a) it does not require the use of sines and cosines and (b) it contains no "singularities" - basically, orientations that have problems and need to be handled separately, such as no rotation or 180 degrees of rotation. The matrix works through multiplication to the 3x1 position vector, which produces another 3x1 position vector, now rotated.

3.2 Design Criteria

The two criteria for success are (a) that the physics engine looks realistic and (b) that the physics engine is correct to high precision.

3.3 Resources Used

The project will be in Java and will make use of the light weight java graphics library (lwjgl.) Lwjgl is an opengl port to Java. I will use the netbeans development environment to write the project.

3.4 Procedure/Workplan

The project proceeded linearly through the following phases. Each phase builds upon and requires that the previous phase be working except the Springs phase, which is more of an addendum:

(1) Framework - A driver and openGL code. Everything not related to physics.

(2) Kinematics - Integrating acceleration into velocity into position. The leap frog algorithm was used for integration.

(3) Angular Kinematics - Integrating orientation into rotational velocity into rotation acceleration. A 3x3 special orthagonal matrix was chosen to represent 3D orientation.

(4) Dynamics - The handling of arbitrary forces on objects and the creation of those forces.

(5) Springs - Fixed or free springs that push and pull objects. Springs can also be used to simulate things like cloth and jello cubes by creating networks of small masses with springs attached between them.

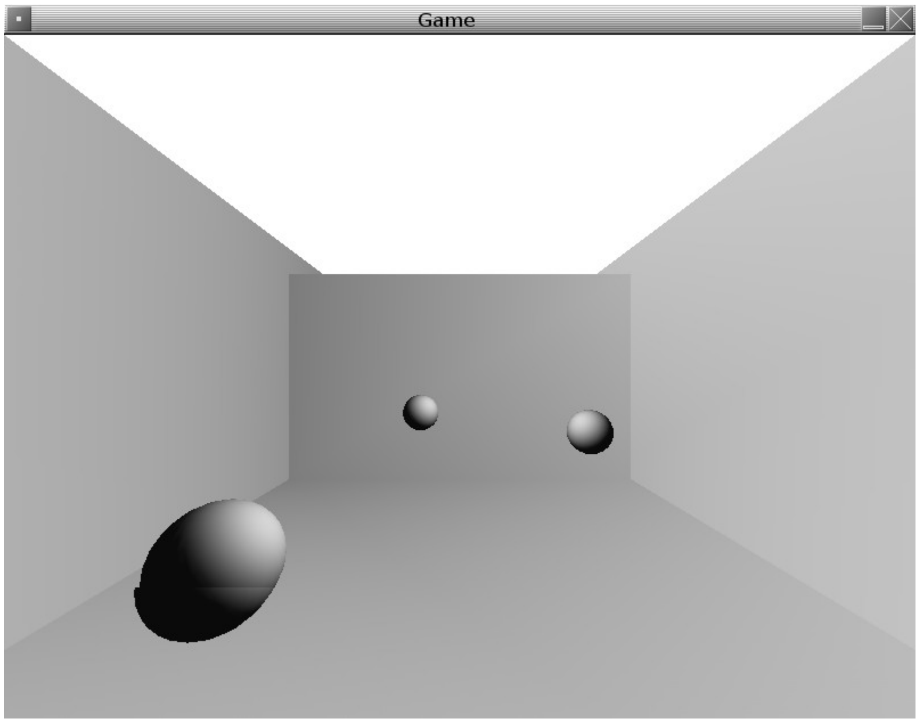
(6) Collision Detection - Finding out when objects are colliding and at what point/edge/face specifically.

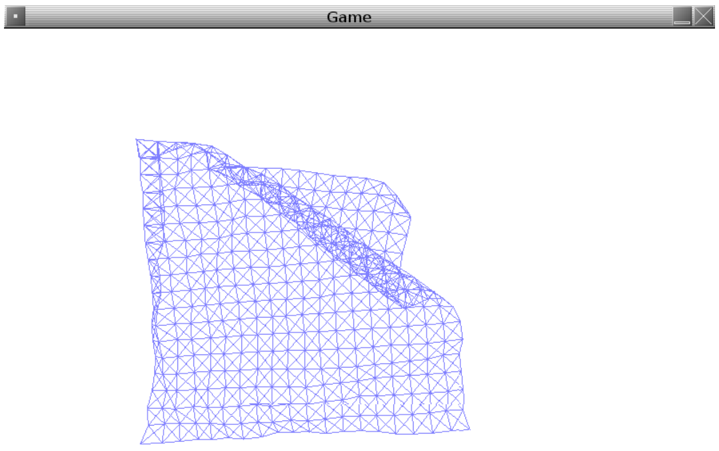
(7) Collision Handling - Responding to a known collision.

(8) Rest - A special module for objects resting on a surface such as a floor; they should not be colliding with the floor every frame.

3.5 Testing

After each stage, a test program will ensure that everything is working perfectly and looks fine graphically.
Here are two screen shots of tests. The first is of a spherical collision test, the second is a spring network test.





4 Results and Discussion

At this stage, the results are preliminary. Everything up to dynamics works splendidly and looks fairly real, aside from a fishbowl effect due to the perspective I chose for viewing the simulation. Springs are working especially well, and can do more than I expected.

Collisions have been less successful. Spherical collisions are easy enough and work well, but polygonal collisions sometimes have odd results. The objects bounce off each other in unusual directions.

5 Conclusions and Recommendations

In this project, I have learned that the world of 3D physics is vast and complicated, even for a program specifically designed for implementing physics and not as a subset of a game's engine. The subfield of rigid body dynamics alone is large enough for a game company to tackle but never fully tame. In the end, a balance must be struck between how much a game company values their coding time and computational time versus how much they value the realism and immersion of their game.

For further research, I recommend that the researchers focus on other subfields of physics for their research besides rigid body dynamics; for example, water and wave effects, aerodynamics, or personal simulation.

6 References

[1] Author unknown. Collision Detection. 2001. 25 Nov. http://www.edenwaith.com/products/pige/tutorials/collision.php.
[2] Baker, Martin. 3D Physics. 2005. 4 Nov. http://www.euclideanspace.com/physics/.
[3] Hecker, Chris. "Physics Part 1: The Next Frontier." Game Developers Magazine. Oct.-Nov. 1996: 12-20.
[4] Hecker, Chris. "Physics Part 2: Angular Effects." Game Developers Magazine. Dec.-Jan. 1996: 14-22.
[5] Hecker, Chris. "Physics Part 3: Collision Response." Game Developers Magazine. Feb.-Mar. 1997: 11-18.
[6] Hecker, Chris. "Physics Part 4: The Third Dimension." Game Developers Magazine. June 1997: 15-26.
[7] Miller, Kurt. Basic Collision Detection. 2000. 21 Jan. http://www.flipcode.com/articles/article_basiccollisions.shtml.
[8] Witkin, Andrew, David Baraff, and Michael Kass. An Introduction to Physically Based Modeling. N.p.: n.p., 1997.