Development of a Physics Engine

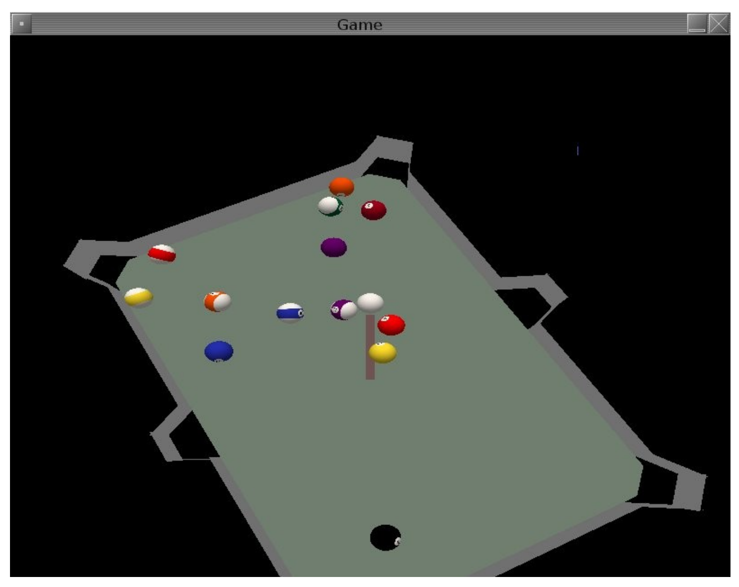
Timmy Loffredo

TJHSST Computer Systems Lab 2005-2006

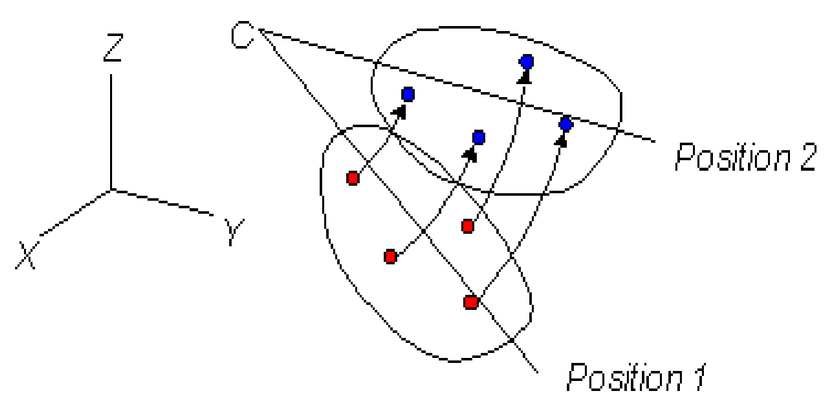
Abstract

Accurate and fast physics simulation is becoming increasingly standard in the gaming industry. A good physics engine can earn big accolades for a game, while a sub par physics engine can significantly bring down the overall immersive experience of a game. The goal of this project is to create a working physics engine independent of any game it might run on. A rudimentary implementation of 3D graphics using LWJGL, an OpenGL port for Java, is also part of the project for visual testing and demonstrations.

The simulator's scope encapsulates a section of Newtonian physics called rigid body dynamics. In this system, objects cannot bend, break, or deform in any way. Any number of arbitrary 3D polygonal solid can be defined in the simulator, as well as spheres, point masses, and plane surfaces. All such objects are subject to Newton's three laws of physics. Gravity and air friction can be activated, if the user decides to do so. Arbitrary and capricious test forces are allowable. The objects interact with each other through collisions. The project also has an implementation of springs, which can be attached either to objects or free space.

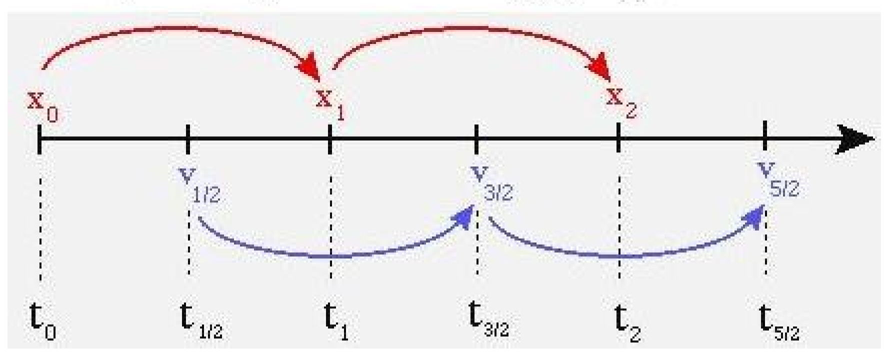


This screen shot was taken from the program simulating a pool game. The user can hit the cue ball whenever the other balls are resting. Collisions are detected between a sphere and another sphere by checking the distance between the spheres against the sum of their radii. Collisions against a surface are detected using a planar equation that tells which side of a plane a point is on.



Rotation and orientation in three dimensions is tricky. There are several ways that they can be represented in modern physics engines. This project uses 3x3 matrices. This matrix, when multiplied by the 3x1 position vector, gives you the rotated position vector.

$$v_{x,n+\frac{1}{2}} = v_{x,n-\frac{1}{2}} + \Delta t\, a_x(x_n, y_n, t)$$
$$x_{n+1} = x_n + \Delta t\, v_{n+\frac{1}{2}}$$
$$v_{x,\frac{1}{2}} = v_0 + \frac{\Delta t}{2}\, a_x(x_0, y_0, 0) + \left(\frac{\Delta t}{2}\right)^2 \frac{\partial a_x}{\partial x}\Big|_{(x_0,y_0,0)}$$



As we all know from physics, velocity is the integral of acceleration over time and position is the integral of velocity over time. When working with arbitrary forces, however, analytic integrals are impossible. Instead, an approximation for integrals must be made, using what is called a numerical integrator. One such integrator is called the Euler method, where the function is assumed to stay constant over some delta t, and is recalculated afterward for the next delta t. This method introduces lots of error, therefore, I used the leapfrog method. It is basically the same, except it uses the midpoint of the function as its approximation instead of the left or right endpoint.

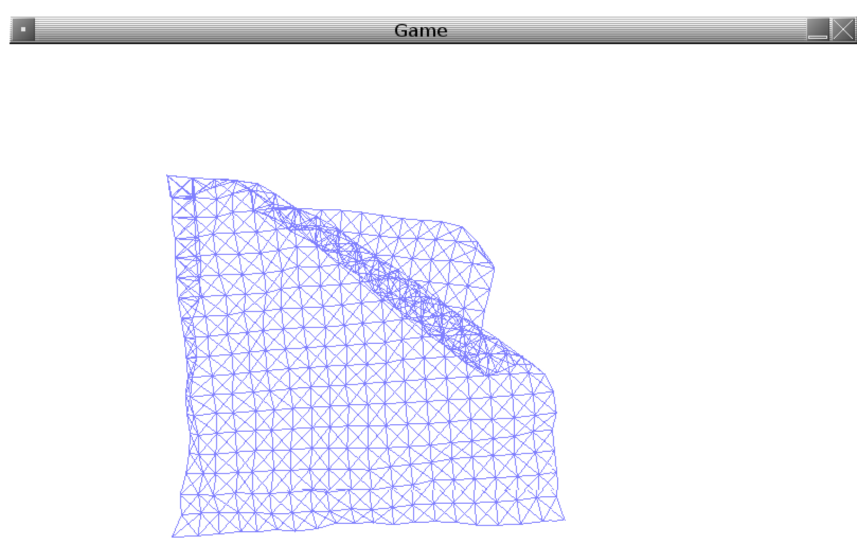
Results

Designing the program with arbitrary forces in mind from the beginning paid off. Rotational kinematics, which I thought was going to be a very difficult subject, ended up performing flawlessly and adding lots of realism.

Springs have been surprisingly useful and impressive for simulation. Springs can make simulations more fun and interactive, and make interesting patterns when combined in some kind of structure. A network of point masses connected with springs, for example, simulates cloth very well. I never expected to be able to simulate cloth, but in the end, it was a fairly easy thing to do.

Collisions are more problematic. Implementing collisions can be split into two tasks: collision detection, deciding when a collision has taken place; and collision response, changing the object's position/velocity/acceleration when they do collide. My current collision detector works well for spheres, but only intermittently for other solids depending on how the solids collide. Collision response also has problems. Objects often bounce off in the wrong direction if the colliding objects start off with angular momentum.

This project was a learning experience and a success. Trying to create my own physics engine gave me a good idea of the depth of computer physics. It also gave me the background research I need to pursue my interests in other areas of physics, like deformable bodies and joints. I am proud of the crowning achievement of this project, a playable pool game. In the end, I am glad I did the project, and would encourage future tech lab students to consider creating a physics engine for their project.



This screen shot was taken from a prototype of the program simulating a network of point masses with springs attached between orthogonal and diagonal neighbors. The only thing being drawn are the springs themselves. The network acts much like some kind of cloth – a napkin or something similar. The more point masses and the higher the density of point masses, then the more accurate it looks.