Graphical Display of a Physics Simulation

Steven Oetjen
TJHSST Computer Systems Lab, 2006-2007

January 19, 2007

Abstract

A physics simulation, in order to adequately demonstrate physical laws and predict an unlimited number of scenarios, must implement a broad range of mathematical equations and provide the user with the ability to set up a scenario with whatever number of objects and arrangements of these objects that he desires. The goal of this project is to create such a simulation.

1 Introduction

The desired physics simulation and graphical display must govern a set of objects under the laws of physics, such as Newton's laws of motion. Accuracy and precision are important when measuring quantities for use in calculations, but it is also important to have a simulation process all objects and apply physical laws efficiently. The program, in other words, must run in real time or at a speed chosen by the user. Such a simulation is worthwhile and valuable because it provides an opportunity to quickly and easily test or verify phenomena, utilizing a computer's ability to store, manipulate, and display data. The results may attract the interest of many, including students needing an aid in their understanding of the physical world and how it behaves. The results can also be used to make or to confirm predictions on how certain scenarios will be resolved.

This simulation is intended to display objects placed by the user graphically and to display information about those objects on a graphical user interface. The project will allow the user to place any combination of objects, including particles, springs, and ramps, in a graphical display, input values for these objects, such as constants, coefficients, and variables, and run a simulation that will track these values and display the interactions and positions of the objects graphically in two-dimensions. The scope of the simulation will be limited to two-dimensions with particles, springs, and ramps, thus constricting the field of study. The concepts involved are kinematics, dynamics and Newton's laws, energy and work, conservation of momentum and collisions, gravitational force, and electric charge, field, potential, and force. Variables such as mass, displacement, kinetic energy, power, and charge, only to name a few, are required, as well as the relationships between these variables in the form of equations.

2 Background

A similar simulation was written in Java by Erik Neumann, named "My Physics Lab" [2]. Newmann's program consists of a series of java applets that run various scenarios based on user input for chosen quantities. Rather than having a large interface in which the user can create any scenario imaginable, with all the same laws governing each scenario,there are several pre-set scenarios, each with a set of laws governing it, as other laws might not be applicable. This organization allows Neumann to cover a broad range of objects, including linear springs, pendulums, colliding blocks, roller coasters, and molecules. It also graphs two quantities of the user's choice over time, and provides mathematical equations for all scenarios.

A model for pressure on a soft body was created by Matyka [4]. It takes into account gravitational and spring forces, and pressure. It also implements a first order Euler integrater to calulate force accumulations necessary for finding the volume of a body and pressure force distribution. He applies these calculations to a 2D soft ball model and a 3D pressure soft body model.

2.1 Kinematics

Two-dimensional kinematics involve the position, velocity, and acceleration vectors of an object [1]. Since velocity is the derivative of position, the linear approximation of x after a small time interval Δt is,

$$x = x_0 + v\Delta t \tag{1}$$

and the linear approximation of v after Δt is,

$$v = v_0 + a\Delta t \tag{2}$$

As $\Delta t \to 0$, these approximations become more accurate in calculating position and velocity, so a small time interval is necessary.

2.2 Collision Detection and Response

A method for improved collision detection applies matrices and vector space to a sweep method over a polygon soup of triangles [3]. A much simpler method to apply to two-dimensional circle objects to check for overlap is a simple comparison between the sum of their radii and the distance between their centers.

$$r_1 + r_2 \geq \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3}$$

When a collision is detected, that is, if condition 3 is true, then for an elastic collision, the velocities of the objects involved must be changed so that both kinetic energy and momentum of the system are conserved [5].

$$\frac{1}{2}m_a v_a^2 + \frac{1}{2}m_b v_b^2 = \frac{1}{2}m_a v_a'^2 + \frac{1}{2}m_b v_b'^2 \tag{4}$$

$$m_a v_a + m_b v_b = m_a v_a' + m_b v_b' \tag{5}$$

Solving equations 4 and 5 for v_a' and v_b',

$$v_a' = \frac{m_a - m_b}{m_a + m_b} v_a + \frac{2m_b}{m_a + m_b} v_b \tag{6}$$

$$v_b' = \frac{2m_a}{m_a + m_b} v_a - \frac{m_a - m_b}{m_a + m_b} v_b \tag{7}$$

the velocities of both objects after the collision are found. These equations can be extended to two dimensions by performing them for both an x- and y-component.

2.3 Contact With Ramps and Response

In order to determine if a projectile is in contact with a ramp, the ramp can be represented by a linear equation of the form $y = mx + b$. Using variables b_{min} to represent the y-intercept of the line, b_{max} to represent that intercept plus an additional $\frac{r_{proj}}{cos\theta}$ to compensate for the radius of the projectile, and an additional $t_{ramp}cos\theta$ to compensate for the thickness of the ramp, two conditions must be true for contact between the ramp and projectile.

$$x_{ramp,1} \leq x_{proj} \leq x_{ramp,2} \tag{8}$$

$$mx_{proj} + b_{min} + t_{ramp}cos\theta \leq y_{proj} \leq mx_{proj} + b_{max} + t_{ramp}cos\theta \tag{9}$$

If a projectile is in contact with a ramp, it exerts a force on it, and so by Newton's third law, that every action force has an equal and opposite reaction force, the ramp must exert an opposite force back. This force is perpendicular to the surface of the ramp. The equations for the force components exerted on a projectile due to the ramp, exerting horizontal force, are

$$F_x = -F_{horiz}sin^2\theta \tag{10}$$

$$F_y = F_{horiz}cos\theta sin\theta \tag{11}$$

while the equations for the force components exerted on a projectile due to the ramp, exerting vertical force, are

$$F_x = F_{vert}sin\theta cos\theta \tag{12}$$

$$F_y = -F_{vert}cos^2\theta \tag{13}$$

In addition, if conditions 8 and 9 are true, the velocity of the projectile must be changed so that the component parallel to the surface of the ramp is unchanged, and the component perpendicular to the ramp is reversed.

$$v_{\parallel}' = v_{\parallel} \tag{14}$$

$$v_{\perp}' = -v_{\perp} \tag{15}$$

The components of the projectile's velocity are found parallel and perpendicular to the ramp, using the equations,

$$v_{\parallel} = v_x cos\theta + v_y sin\theta \tag{16}$$

$$v_{\perp} = -v_x sin\theta + v_y cos\theta \tag{17}$$

If the perpendicular component of the velocity is negative, meaning the projectile is going into the ramp, the velocity of the projectile needs to be changed so that the component parallel to the ramp is unchanged and the perpendicular component is reversed. In order to do this, adjustments are made using the following formulas:

$$v_x = v_{\parallel} cos\theta + v_{\perp} sin\theta \tag{18}$$

$$v_y = v_{\parallel} sin\theta - v_{\perp} cos\theta \tag{19}$$

By substituting equations 18 and 19 into equation 16, the desired relationship in equation 14 is obtained.

$$v'_{\parallel} = (v_{\parallel} cos\theta + v_{\perp} sin\theta)cos\theta + (v_{\parallel} sin\theta - v_{\perp} cos\theta)sin\theta$$

$$v'_{\parallel} = v_{\parallel}(cos^2\theta + sin^2\theta) + v_{\perp} sin\theta cos\theta - v_{\perp} cos\theta sin\theta$$

$$v'_{\parallel} = v_{\parallel}$$

Similarly, by substituting equations 18 and 19 into equation 17, the desired relationship in equation 15 is obtained.

$$v'_{\perp} = -(v_{\parallel} cos\theta + v_{\perp} sin\theta)sin\theta + (v_{\parallel} sin\theta - v_{\perp} cos\theta)cos\theta$$

$$v'_{\perp} = -v_{\parallel} cos\theta sin\theta + v_{\parallel} sin\theta cos\theta - v_{\perp}(sin^2\theta + cos^2\theta)$$

$$v'_{\perp} = -v_{\perp}$$

3 Development

There are several requirements that must be met to ensure success of the simulation. Inputs must be doubles able to be parsed, given by the user through JTextFields, to any given precision, in the range of doubles able to be stored in memory, and given as often as desired by the user between runs of the simulation. Outputs must be displayed as objects in their respective positions and values in JTextFields and JLabels, to an accuracy of a $\pm 1\%$ error, in the range of doubles able to be stored in memory, and given as frequently as the Timer fires. Calculations for all the objects must be completed within the time elapsed during the timer delay in order to keep the simulation time proportional to real time. This proportionality will be created by the time scale factor the user provides to speed up or slow down the simulation as desired.

Success for most of the requirements can be observed qualitatively, such as if objects appear with different names and colors, if values are being recorded to a certain decimal place, if objects overlap or collide properly, or if ramps are drawn with the correct slope and thickness. The requirements that demand quantitative testing are the accuracy of the output, and the efficiency of processing time. The output can be compared to theoretical values or examined in terms of a known equation, and the processing time can be judged by comparing elapsed simulation times to elapsed real time for different numbers of active objects.

The project was developed by dividing functions into several iterations of design, programming, and testing. The tasks of these iterations include implementing each concept and set of equations and laws. These groups are kinematics, dynamics, momentum, gravitation, energy, and electrostatics. Each iteration proceeded with the following steps:

Design Classes, methods, and variables were added or modified, and the function of each element and the means of communication of information between relevant classes was determined.

Programming The necessary changes to the program were made. All programming was done in Java, using Swing and AWT packages for graphics and GUI components.

Testing To test a given implementation of a law, the user will input arbitrary values for all necessary fields involved in or dependent on that law. The output generated by these random types of input will be compared to theoretical values derived from mathematical formulas. To analyze the error, a percent error calculation will be performed on these two values, and the accuracy will be gauged by that percent. In this way, the accuracy of the program's ability to model predictable phenomena can be measured and displayed in chart format.

Aside from implementing equations and laws, several other components to the program must be implemented.

1. GUI Structure and Layout
2. Graphics Panel Distance and Time Scale
3. Support of Multiple Objects/Multiple Types and Classes of Objects
4. Support of Coordinate and Polar Vector Conversions
5. Collision Detection
6. Improved Integration Methods

The major classes involved are the *GraphicsPanel*, which displays the graphical output, and the *GUIPanel*, which displays the numerical output and reads in the user's input. Other critical classes include *Projectile*s and *Ramp*s, the objects themselves which store their own quantities, and *ProjectileInputs* and *RampInputs*, which provide a GUI layout for input to each object. The architecture is designed so that changes in the future can be easily accomodated. The program is modularized so that new features can be added in without interfering with existing ones. For example, additional types of objects can be created by creating a class for the object and a class for its inputs. Room on the GUI can be allocated for that input, and functions can be added to draw and implement appropriate laws for those objects.

4 Results and Conclusion

To be a successful and useful simulation, the program must implement all equations and relationships correctly, and process all data efficiently. Specific requirements must exist for processing time, kinematics, collisions, ramp forces, and ramp collisions.

4.1 Processing Time

No matter how great the number of objects, the program's calculations must not take longer than the timer's firing delay, so that the program runs proportionally to real time. For the processing time analysis, the simulation was run for 10 seconds for each number of objects, n_{obj}, and the real and simulation times elapsed were recorded, as well as a percent error calculation.

Processing Time Analysis

n_{obj}	t_{sim}	t_{real}	% error
1	9.5	10.30	7.77
5	9.4	10.10	6.93
10	9.5	10.07	5.66
50	9.3	9.86	5.68
100	9.3	9.99	6.91

The program's processing time successfully meets the requirement, because there is no upward trend in simulation time when the number of objects is increased, so it is not being slowed down by multiple objects, and it is running proportionally to real time. The percent errors appear large because of the human error involved in starting and stopping both a stopwatch and the simulation. The times are consistent, and the percent errors stay relatively constant.

4.2 Kinematics

Using any given inputs, a projectile's position at any time later should be accurate to its theoretically predicted position within $\pm 1\%$. To test the program's accuracy with kinematics in cartesian coordinates, the program was given the following inputs, and the outputs for x were recorded at various times. These values are compared with theoretical values, and a percent error calculation is shown.

Value Accuracy for Cartesian Coordinates

Inputs

x_0(m)	v_{x0}(m/s)	a_{x0}(m/s^2)
7.893	9.342	1.087

Outputs

t(s)	x_{sim}	x_{theor}	% error
1.4	21.961	22.037	0.345
2.2	30.957	31.076	0.383
3.4	45.755	45.939	0.400
4.5	60.695	60.938	0.399
5.5	75.418	75.715	0.392
6.2	86.371	86.706	0.386
7.2	102.942	103.330	0.376
8.1	118.785	119.222	0.367
9.3	141.279	141.781	0.354
10.2	159.176	159.727	0.345

Note: theoretical values were calculated using the equation for constant acceleration, $x = x_0 + v_{x0}t + \frac{1}{2}a_{x0}t^2$.

The accuracy of the values for cartesian coordinates meets the requirement, because for all of the checked points in time, the simulation's value was within ±1% of the theoretical value.

To test the program's accuracy with kinematics in polar coordinates, the program was given the following inputs, and the outputs for x were recorded at various times. These values are compared with theoretical values, and a percent error calculation is shown.

Value Accuracy for Polar Coordinates

Inputs

$\|r\|$(m)	θ_r(°)	$\|v\|$(m/s)	θ_v(°)	$\|a\|$(m/s^2)	θ_a(°)
32.725	45	7.834	120	2.087	240

Outputs

t(s)	$\|r\|_{sim}$	$\|r\|_{theor}$	% error
1.0	34.629	34.525	0.302
2.2	35.990	35.764	0.631
3.3	36.292	35.974	0.884
4.1	36.166	35.805	1.008
5.1	36.148	35.782	1.021
6.4	37.879	37.627	0.670
7.2	40.893	40.781	0.275
8.4	49.215	49.369	-0.311
9.0	55.186	55.476	-0.523
10.2	70.570	71.116	-0.768

The error of the values stayed relatively constant, but did reach above a 1% error. This could be because during previous tests, only one dimension was tested, but for this test, both dimensions were necessary because of the nature of polar coordinates. Error could have multiplied with two dimensions. Accuracy in two dimensions is not as high as desired, so a better method of integration is needed to increase the level of accuarcy.

4.3 Collisions

For any collision between two objects, the momentum of the system before the collision must be equal to the momentum after the collision, and for elastic collisions, the same is true of kinetic energy. To test this function, two objects with different masses and velocities were collided, and the initial and final momentums and kinetic energy values were calculated.

Two-Dimension Collision

	m (kg)	v_x (m/s)	p_x (kg·m/s)	KE_x (J)	v_y (m/s)	p_y (kg·m/s)	KE_x (J)
a	1.0	8.457	8.457	35.760	3.078	3.078	4.737
b	5.0	0.000	0.000	0.000	-2.000	-10.000	10.000
a'	1.0	-5.638	-5.638	15.894	-5.385	-5.385	14.501
b'	5.0	2.819	14.095	19.867	-0.307	-1.537	0.236

total		total	
p_x	8.457	KE_x	35.760
p'_x	8.457	KE'_x	35.760
p_y	-6.922	KE_y	14.737
p'_y	-6.922	KE'_y	14.737

In this test, momentum and kinetic energy were conserved in both $x-$ and $y-$components. This means that each of these quantities are equal before and after the collision. For example, $p_x = p'_x$. The test was successful.

4.4 Ramp Forces

When a projectile is exerting a force on a ramp, the ramp must exert an equal and opposite force on the projectile. With gravity acting on the projectile, it should slide down the ramp, accelerating parallel to the ramp's surface. A range of angles, both positively and negatively sloped, will be tested to ensure projectiles accelerate parallel to the surface in all cases. The angles to be tested will be -60°, -45°, -30°, 0°, 30°, 45°, and 60°, all measured from the positive x-axis.

Acceleration Direction of Projectile Sliding Down Various Ramps

θ_{ramp}	$\theta_{acceleration}$
-60.0	-60.0
-45.0	-45.0
-30.0	-30.0
0.0	0.0
30.0	30.0
45.0	45.0
60.0	60.0

The projectile always is accelerated by the ramp in the same direction the ramp slopes in. Therefore, this test verifies that for any slope of the ramp, a projectile will accelerate in that same direction.

4.5 Ramp Collisions

When a projectile has a velocity component perpendicular and into the surface of a ramp, and they collide, the angle of incidence must equal the angle of reflection after the projectile bounces off. A range of angles, both positively and negatively sloped, will be tested to ensure projectiles bounce off of ramps correctly. The angles to be tested will be -60°, -45°, -30°, 0°, 30°, 45°, and 60°, all measured from the positive x-axis. For each velocity adjustment test, the projectile had the same initial velocity of 10 m/s, directed at -90° clockwise from the positive x-axis, and the angle of the ramp varied.

Angles of Incidence and Reflection of Projectile Reflected Off Various Ramps

θ_{ramp}	θ'_v	θ_i	θ_r
-60.0	-30.0	30.0	30.0
-45.0	0.0	45.0	45.0
-30.0	30.0	60.0	60.0
0.0	90.0	90.0	90.0
30.0	150.0	60.0	60.0
45.0	180.0	45.0	45.0
60.0	-150.0	30.0	30.0

The angle of incidence is always equal to the angle of reflection, no matter what the angle of the ramp is. Therefore, this test verifies that for any slope of the ramp, a projectile will bounce off of correctly, as $v'_{\parallel} = v_{\parallel}$ and $v'_{\perp} = -v_{\perp}$.

4.6 Conclusion

The simulation is successful in most areas, and is certainly useful for predicting physical phenomena and aiding students in their understanding of physics. All calculations are performed within the timer delay, so there is no lag when the simulation is run, and all formulas for effects of collision have been verified by testing. Kinematics in one dimension are calculated within ±1%, but the major flaw is that when expanded to two dimensions, the calculations can occasionally rise slightly beyond the desired ±1% error. To fix this, a more accurate method of integration is needed to counteract error that propogates from linear approximations.

References

[1] D. C. Giancoli, Physics: Principles with Applications, Prentice Hall, 1995. http://wps.prenhall.com/esm_giancoli_physicsppa_6/0,8713,1113739-,00.html (November 2, 2006)

[2] E. Neumann, "My Physics Lab - Physics Simulation with Java", 2004. http://www.myphysicslab.com/ (November 2, 2006)

[3] K. Fauerby, "Improved Collision Detection and Response", 2003. http://www.peroxide.dk/papers/collision/collision.pdf (December 11, 2006)

[4] M. Matyka, "How To Implement a Pressure Soft Body Model", 2003. http://panoramix.ift.uni.wroc.pl/ maq/soft2d/howtosoftbody.pdf (November 2, 2006)

[5] R. Fitzpatrick, "Collisions in 1-dimension", 2006. http://farside.ph.utexas.edu/teaching/301/lectures/node76.html (November 30, 2006)