Angry Cuts

Anton Albèrt Karlström, Lovisa Dahl,
Emma Edvardsson and Hannes Ingelhag,
Norrköping

March 12, 2014

Abstract

Write an abstract here!

1 Introduction

1.1 Background

Angry cuts is a project of modeling and simulating the scenario of a weight attached to a pendulum converted into a projectile in two dimensions. The final concept is a simulation of several weights, each attached to a pendulum. All the ropes are cut at the same time which results in a projectile movement for all the weights. In this state the weights interact with both the walls and each other.

Figure 1 shows the first idea, where a pendulum was supposed to hit a tower of wooden bars. The projectile movement is inspired by the slingshot aiming towards a given target in the game *Angry birds*. The idea of releasing a weight from a pendulum is claimed from *Cut the rope*. In order to create a scene with real physics, the collisions in the scene require careful calculation of the collisions, both with static and dynamic objects. Since objects in equilibrium are difficult to handle in simulation, the idea was modified to have the weights alone acting in a room with walls and floor, and extend with several balls instead of the wooden bars. The definitive idea can be seen in *figure 2*, the scene consists of a pendulum in a room, which later will be released as a projectile.

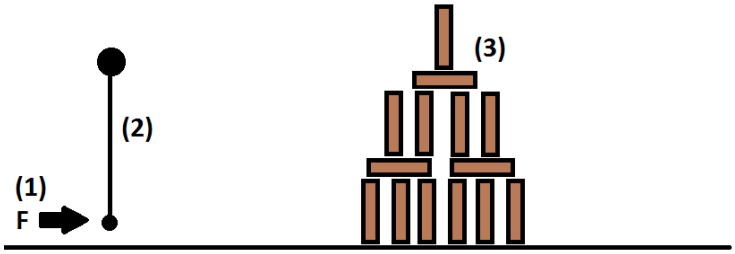

Figure 1: Original idea

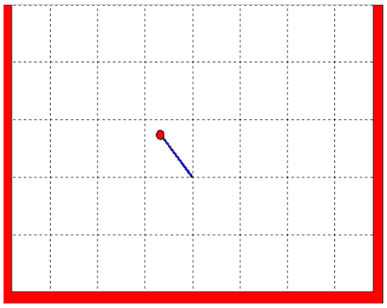

Figure 2: Modified idea

1.2 Purpose

The purpose of this report is to describe and follow the development of the scene with pendulums, projectiles and collisions in two dimensions. The scene uses physically correct models and formulas for collisions with static and dynamic objects. Air resistance and energy loss are also taken in consideration on the pendulum, the projectile and the collision.

Another purpose with this project is to use Matlab to test and develop the model before transferring the model into C++ and OpenGL.

2 Physical model

The first part of this project was to find the physical description of the first two parts of the model: the pendulum and the projectile movement. The second part was to implement collision handler and air resistance.

2.1 Pendulum

A physical model of a nonlinear pendulum (1) is required since the system operates with large angles.

$$\frac{d^2\theta}{dt^2} + \frac{g}{L} * sin(\theta) = 0 \tag{1}$$

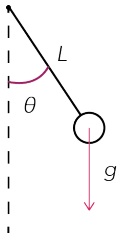

Figure 3: Physical model of a pendulum

As can be seen in *figure 3*, g describes the gravity constant, L the length of the rope of the pendulum and θ is the angle of the pendulum perpendicular to the gravity. $\frac{d^2\theta}{dt^2}$, the angular acceleration, is the relationship between the length of the rope and the angle perpendicular to the direction of the gravity.

2.2 Projectile

The initial velocity of the projectile was retrieved from the velocity of the weight when the rope is cut. In order to transfer the movement into projectile, the velocity was split into horizontal (2) and vertical velocity (3). Besides the initial velocity from the pendulum the gravity was taken into account, which effects the vertical acceleration (4).

$$v_x = \omega * cos(\theta_i) \tag{2}$$
$$v_y = \omega * sin(\theta_i) \tag{3}$$
$$a_y(t) = \alpha * sin(\theta_i) - g \tag{4}$$

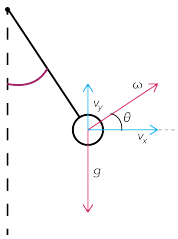

Figure 4: Dynamics of the projectile

θ_i denotes the initial angle, ω is the angular velocity and α the angular acceleration of the pendulum.*Figure 4* describes the initial condition of the projectile movement.

2.3 Air resistance

To add air resistance to the model, the velocity should be reduced according to the angle perpendicular to the gravity. The air resistance (5) takes into account the density of air, the drag coefficient of a sphere, the cross sectional area and the velocity of the weight.

$$F_d = \frac{1}{2}(\rho * v^2 * c_d * A) \tag{5}$$

2.4 Collision detection

The second part of the model was to consider the collisions: with walls and floor. This was performed by changing the direction after the collision according to the normal of the surface hit by the weight. The velocity was reduced with a factor to simulate an energy loss from the collision.

The third physical part in the system was collision between two moving objects. In this project all collisions are considered elastic, which means that the objects cannot stick together in the collision. A collision is detected when the distance between two weights is smaller than the sum of the radii. The distances can be calculated using Pythagorean theorem, illustrated in *figure 5*. Since a numerical method is used to calculate the next step of the scene, the weights might overlap and due to that get stuck to each other, an unwanted phenomena. To get a more realistic simulation of the collision, the boundaries of the weights must intersect instead of overlap. Therefore, when a collision is detected, the weights are set to their previous positions and from there smaller steps in the same direction are taken until they intersect, *figure 6*.

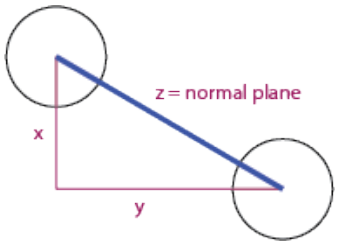

Figure 5: Distance between two weights using the pythagorean theorem

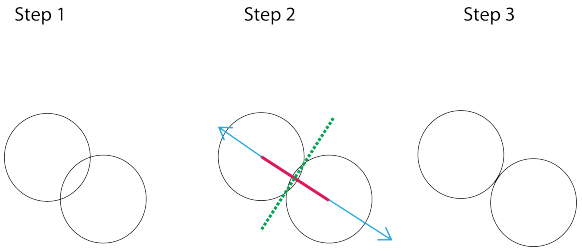

Figure 6:
Step 1: Overlapping weights
Step 2: Collision and normal planes are found
Step 3: Collision can now be calculated

To calculate the direction of the weights after the collision, the normal plane of the weights is used. This can be determined as the distance between center points of the weights. A collision plane is calculated by the normal plane, as illustrated in *figure 7*.

The initial velocity vectors can now be described in the normal- and the collision plane, *figure 8*. These vectors can be obtained by the operation vector dot product. When these vectors are found the new velocitiy vector after the collision can be determined according to 6 and 7.

$$v_{1a} = \frac{1}{m_1+m_2}(v_1(m_1 - m_2) + 2m_2 * v_2) \tag{6}$$

$$v_{2a} = \frac{1}{m_1+m_2}(v_2(m_2 - m_1) + 2m_1 * v_1) \tag{7}$$

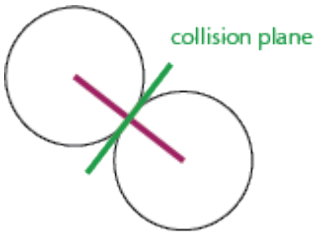

Figure 7: Collision plane

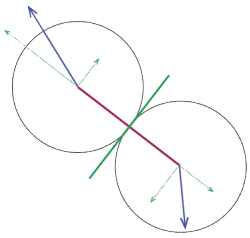

Figure 8: Modified idea

2.5 Numerical implementation

2.5.1 Euler's explicit method

Euler's explicit method (8) is used to calculate the x and y position of the weight and the angular velocity continuously over time of the nonlinear pendulum.

$$y(t+h) = y(t) + f(t, y(t)) * h \tag{8}$$

In (8), h denotes the step length, $y(t)$ is the function and f describes the derivative of y.

2.5.2 Runge-Kutta method

Runge-Kutta is another numerical method. Compared to Euler it gives a more stable system and can be used with a larger step size. The most common form of the Runge-Kutta is RK4, a fourth order method, which is described in (9).

$$\begin{aligned} k_1 &= h * f(x_i, y_i) \\ k_2 &= h * f(x_i + 0.5 * h, y_i + 0.5 * k_1) \\ k_3 &= h * f(x_i + 0.5 * h, y_i + 0.5 * k_2) \\ k_4 &= h * f(x_i + h, y_i + k_3) \end{aligned} \tag{9}$$

By using these four equations the next step can be calculated according to (10).

$$y(i+1) = y(i) + \tfrac{1}{6}(k_1 + 2 * k_2 + 2 * k_3 + k_4) \tag{10}$$

2.6 Graphical implementation

2.6.1 MATLAB

A physical model of one weight was defined and implemented in MATLAB for a simulation test. Constants for the calculations can be found in *Appendix 1*. This was made iteratively, starting with the pendulum alone followed by adding the projectile movement and the last step with collision detection with static objects. The final simulation consisted of a weight attached to a pendulum, see *figure 9*, which was released after a given time and converted into a projectile. The weight would bounce on the walls and the ground until the added energy loss caused it to stop moving.

The second part was to add interaction between two weights in the same room. This step required a fully working model of one weight and an additional collision management to handle collision between two dynamic objects. *Figure 10* shows two weights of different radii.

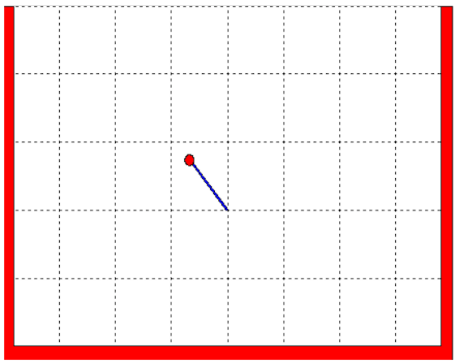

Figure 9: Plot of one pendulum

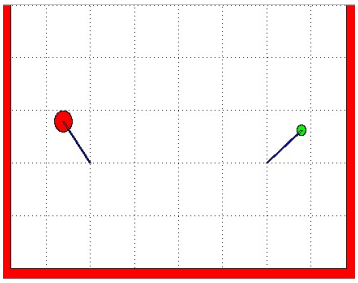

Figure 10: Plot of two pendulums

2.6.2 OpenGL 3.3

To improve the graphical representation of the model the code was converted to C++ and OpenGL. The software used in the project were Sublime Text 2, Visual Studio and CMake.

A possibility to change the number of weights in the scene was implemented, in such way that the pendulums were drawn with even intervals according to the number of weights. *Figure 11* and *12* presents the scene with three weights in OpenGL. The constants used for these figures are the same as previous.

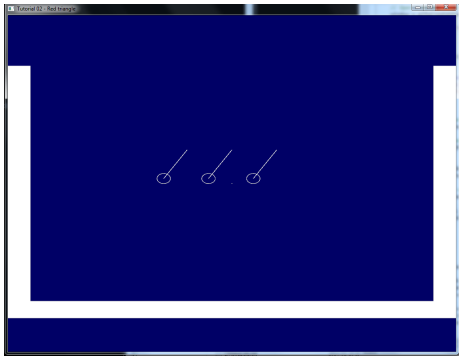

Figure 11: Simulation of three pendulums, first version

Figure 12: One projectile, first version

3 Result

The result retrieved from this project is a program where the user could enter the desired number of pendulums in a GUI. The trigger of movement for the pendulums are their initial angle given by the user or in program. During the execution of the program the user could follow the flow of energy in the scene. The weights are positioned depending on the given number and are set to a random color. In *figure 13* and *14* seven weights are implemented.

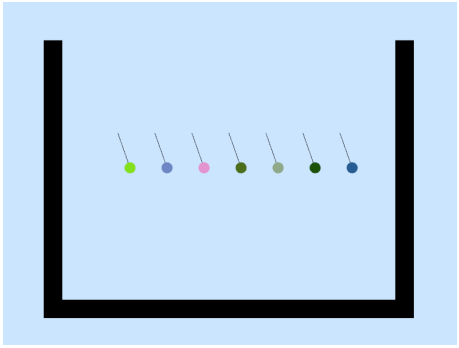

Figure 13: Seven weights simulated in OpenGL

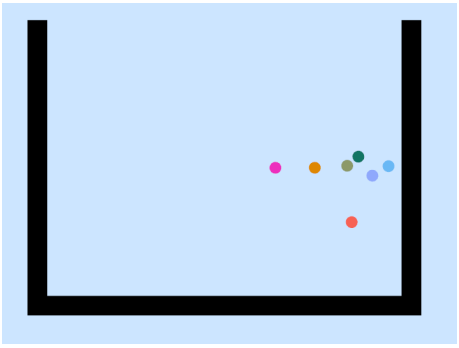

Figure 14: Seven bouncing weights in a room

In the first part, the pendulums were affected by air resistance. In this scene the balls have the properties of lead and the air resistance coefficient was calculated from the circular shape.

4 Discussion

The implementation of the physics into code started as soon as the idea was found which made this project robust and stable from the beginning. The work was most of the time divided into groups of two and these pairs implemented and discussed code together which made the work hours efficient.

During the conversion to C++, a couple of obstacles in terms of different operative systems and the different versions of OpenGL caused troubles. This took much time and focus from the project itself. OpenGL 3.3 has some new implementations when drawing objects, different from what we have used before.

The Euler explicit method was first used as the numerical integration method. However, Euler gives a larger numerical error which grows for each iteration. This is noticeable when using a larger step size since the simulation gets unstable. To minimize this problem the Runge-Kutta 4 method was implemented, which represent a more accurate model. This method considers four following steps, each dependent on the previous one, from which a weighted average representing the next step of the slope could be calculated.

Another difficulty with the simulation was to handle time as a parameter. In this project the simulation was performed as fast as the current computer could compute the code, which brought troubles with different computers. Some computers calculated the code too fast and in that case the simulation could look unrealistic.

One idea was to implement a simple model so that the balls could roll off each other to avoid the scenario that weights gets stuck on top of each other at low velocities. However, this was not implemented due to lack of time.

5 References and links

Nonlinear pendulum:
http://www.math24.net/nonlinear-pendulum.html

Eulers explicit method:
http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx

Air resistance:
http://hyperphysics.phy-astr.gsu.edu/hbase/airfri.html

OpenGL tutorials:
http://www.opengl-tutorial.org/

Vector-Collision:
http://simonpstevens.com/Articles/VectorCollisionPhysics

Appendix 1

Table 1: Constants for the weights

CONSTANTS		
Radius	0.02	(m)
Volume	$1.68 * 10^{-5}$	(m^3)
Density	11340	(kg/m^3)
Mass	0.19	(kg)
Air Coefficient	0.47	