University of Ontario Institute of Technology

CSCI 3010U / ENGR 4829U

Final Project Report

Rigibo Puzzle - 2D RigidBody Game

Authors:
Mark Reventar
Taurean Scantlebury
Johnny Eid

Course Instructor:
Dr. Faisal Qureshi

April 13, 2013

Contents

1 Introduction

The final project is a prospect into the world of 2D rigid bodies. Rigibo Puzzle is a simple 2D game where the player can explore the behavior of rigid bodies and how they interact with one another. The game forces the player to predict the movements of the rigid bodies ahead of time before the click of their button.

The objective of RigiBo puzzle is to get the ball onto the wooden platform. This is done by strategically deleting the RigiBo boxes. A waiting timer is conducted when a body is destroyed to prevent the player from deleting all boxes at an instance. Finally the ball must remain on the wooden platform for three seconds to complete the level. They are currently 3 levels developed in the game.

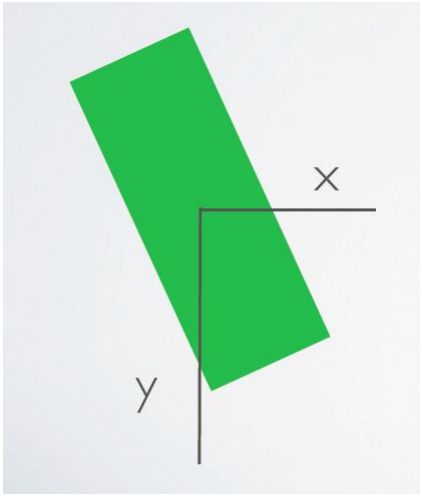
The game simulation also contains a demo section that features Crates, Wheels, Friction and Flooding. These demonstrations focus on the different physics topics that are implemented into the project.

2 Model

The model of the final project consists of the common physics of a rigid body in a world space. This includes the motion of the rigid body and its interaction with other rigid bodies.

2.1 Motion

2.1.1 State Variables



For each body, there are three state variables to specify postion [3].

- X Position, horizontal position of the center of mass

- Y Position, vertical position of the center of mass
- θ Angle of Rotation

For each body, three state variables specify its velocity

- v_x horizontal velocity of the center of mass
- v_y vertical velocity of the center of mass
- ω Angular Velocity

After these variables are determined, the total force of ($\bar{F} = [F_x, F_y]$) motion and torque can be calculated with the following equations

$$F_x = mx'' \tag{1}$$

$$F_y = my'' \tag{2}$$

$$\tau = I\theta'' \tag{3}$$

Where m represents mass and I represents the moment of intertia.

2.1.2 Gravity

In the simulation Gravity affects the center of mass in the vertical direction using the formulas [3]

$$F_{gravity} = -mg \tag{4}$$

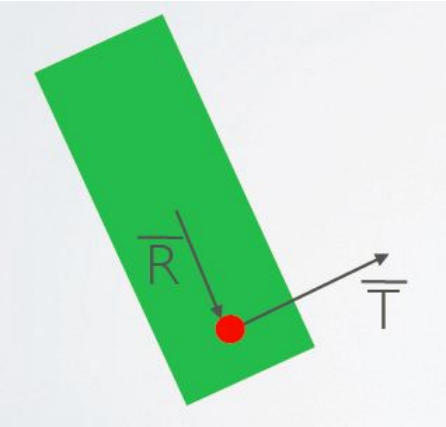
2.1.3 Friction

Friction is caused from an opposing force in the direction of which motion is flowing, in the simulation as the rigid bodies interacted with different types of materials they were affected by the equivalent friction force. A constant k value was assigned to different materials in the game and the forces in the x and y direction were calculated using the following formula

$$F_{frictionx} = kx' \tag{5}$$

$$F_{frictiony} = ky' \tag{6}$$

2.1.4 Thrust



Thurst occurs in the simulation when one body pushes ont he other in a speific direction. A point P can be calculated on any part of the body in both the x and the y direction. The bodies rotate freely around the point P, so torque τ is also calculated. Let $\bar{T}$ be the thurst force vector, which operates at point P and $\bar{R}$ the distance vector.

$$\bar{R} = (R_x, R_y) \tag{7}$$

$$\bar{T} = (T_x, T_y) \tag{8}$$

$$\bar{R} \times \bar{T} = I\theta'' \tag{9}$$

2.2 Collision Detection

At each step in the simulation, we check for the distance between the center of mass between rigid bodies. When close, the two rigid bodies are flagged and contact listeners are appointed. A collision is detected when our object enters (overlaps) another object [2].

Once a collision is detected, a binary search algorithm (as use in a previous lab) is used to back up the simulation to just shortly before the time that the collision happened. Then an approximation is made and used to find the exact time that the collision occurred. We then use our equations of motion to calculate the newly changed velocities after the collision.

2.3 Collision Handling

After a collison happens in the simulation, the bodies are affected by the impluse force. Impluse is the change of momentum of an object when a large force is applied over a brief period of time [1]. The pre and post velocities are calculated

$$\bar{v}_{a2} = \bar{v}_{a1} + j\bar{n}/m_a \tag{10}$$

$$\bar{v}_{b2} = \bar{v}_{b1} + j\bar{n}/m_b \tag{11}$$

The change in angular momentum is calculated

$$\omega_{a2} = \omega_{a1} + (\bar{r}_{ap} \times j\bar{n})/I_a \tag{12}$$

$$\omega_{b2} = \omega_{b1} + (\bar{r}_{bp} \times j\bar{n})/I_b \tag{13}$$

3 Implementation

The implementation of our game RigiBo Puzzle is written in the programming language Java. The system was designed into two parts, the Front End and the Back End.

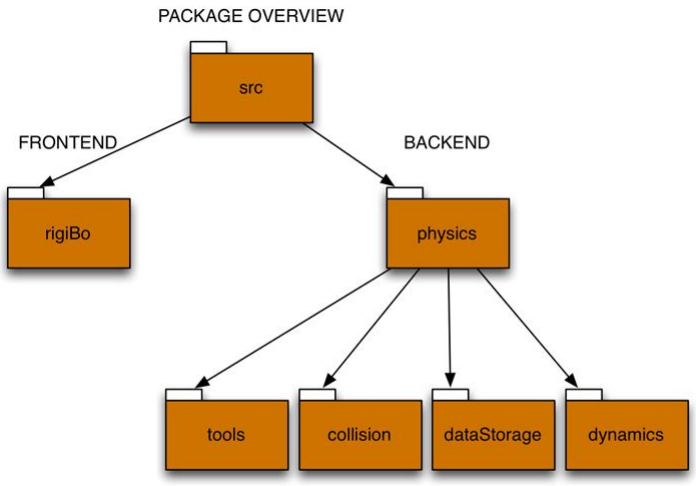
3.1 Resources

The outside resources used in the development of the RigiBo puzzle are three open source libraries. Standard Java Utility library, OpenGL and Light Weight Java Gaming Library. The openGL API is used to render the 2d graphics in our game. The Light Weight Java Gaming Library (LWJGL) provides a display window and easy to use frame rate utility. LWJGL is also integrated automatically with openGL, which made it easier for the rendering of objects into the game.

3.2 System Design

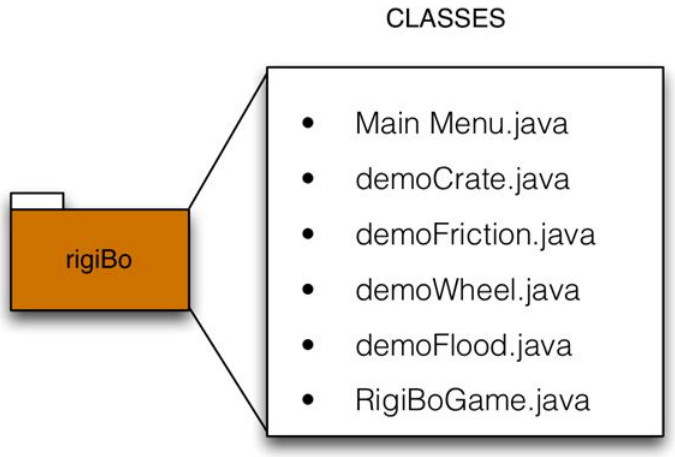
RigiBo puzzle design is split into two main parts

- *FrontEnd* Generates the layout and processing of the game. Creates the display windows and rigid bodies. Handles the user input and rendering of graphics.
- *BackEnd* Physics engine. Responsible for storing as well as calculating the Rigid Bodies positions and motions.



3.2.1 The Front End

The front end manages navigation through our system. Creates and destroys various displays depending on the users request. Displays in our system can be a demo or the RigiBo puzzle game itself.



Class descriptions

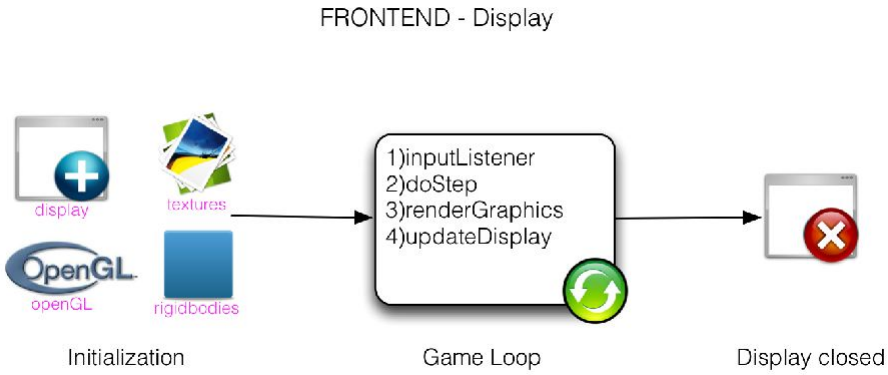
Class	Description
Main Menu	Source of navigation to proceed towards the demos or the RigiBo Puzzle game
demoCrate	Demonstrates the creation and motion of rigid bodies into the world.
demoFriction	Demonstrates the use of friction
demoWheel	Demonstrates the implementation of rotation on the rigid bodies
demoFlood	Demonstrates the maximum number of rigid bodies that can be handled by the world pool and the elasticity of the rigid bodies
RigiBo	RigiBo puzzle game. RigiBo2.java, RigiBo3.java also included as more levels for the game.

Display Window

The display window is 720 pixels in width and 800 pixels in height. It is Vertical Synced enabled to synchronize the monitors refresh rate with the application frame rate. This prevents a flickering shutter of the display. The application is run at 60 frames per second to coincide with our world delta time set at 1/60 seconds.

When the display is started, it first initializes its display properties, openGL properties, loads all the textures that will be used and creates all initial rigid bodies. After initialization setup, the program will enter the game loop. The loop will terminate when the display is closed or the escape key is pressed. Each iteration of the loop consists of 4 steps:

- *inputListener* Listen for users keyboard and mouse inputs
- *doStep* Call the backend to complete step with delta time 1/60s
- *renderDisplay* Render Rigid Body graphics to the screen
- *updateDisplay* Update frame onto screen (60fps)



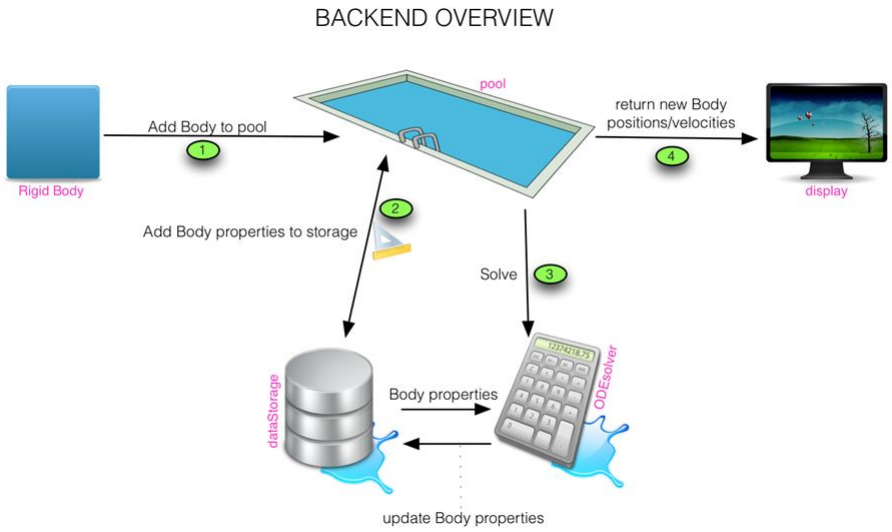
Multiple display windows are used in our system. This is because each demo and game level will have to be re rendered into the scene once the user chooses to navigate to it. Therefore to keep the code structure organized, each demo and game level will be in a separate window display. The cycle of a window display is shown above. When a user wants to run a demo or begin a game level, the current display is destroyed and closed and the new display is started.

3.2.2 The Back End

The backend acts as the physics engine for RigiBo puzzle. It has two main responsibilities

- Data Storage Keep track of all Rigid bodies definitions and properties
- ODE Solver Calculate rigid bodies new positions and velocities using our equations of motion

The main central class is the Pool class. This class manages all physics entities, dynamic simulating and asynchronous queries. In our world coordinate there will be a pool of rigid bodies created by the front end. As the front end creates the Rigid Bodies and defines its properties, the bodies are sent to the Pool. The Pool stores all RigidBodies properties and definitions into the data-handling package, dataStorage. With all the rigid body properties stored inside our stacks, the pool will now be able to use this data to determine any collisions between bodies and solve for final velocities and positioning after collision. The collision listeners in our system are only triggered when the center of masses between two rigid bodies are within each others size fixtures. Size fixtures are a set square trace around each object. This helps with speed metrics as well memory efficiency. Having to listen and calculate for any contacts between all rigid bodies for every time step can become very costly.



A Rigid Body object contains definitions and properties that is set by the front end.

RigidBody

Definition

- Position
- Velocity
- Angle of Rotation
- Angular Momentum

Properties

- Type(Dynamic, Static, Kinematic)
- Shape (vertices, edges, etc.)
- Mass
- Elasticity Constant
- Friction Constant

3.2.3 Integration

The backend and front end work together to apply realistic movement of rigid bodies throughout the game. When the game is first started, the front end will create a Pool object from the backend. The front end is then free to use and

create Rigid Bodies into the pool by declaring a RigidBody object, and defining its properties. The rigidbodies are then given a new position and velocity via every pool.step() call (doStep). The front end can then go into the back end and retrieve the RigidBody objects new position and velocities which will be used to render the objects back into the display.

4 Results

Since our simulation is based on rigid body collisions, the results after running our simulation describes to us how rigid bodies collide in 2 dimensions. When two rigid bodies collide, the resulting effect of the collision can be a number of things. These would vary from the bodys simply bouncing off each other to the rotation and even just stopping their movement.

An example of one of the collisions observed in the simulation is that when an object is dropped on a flat surface (with the bottom of the object parallel to the surface and greater than 50% of the objects mass lands on the surface) the object will stop its motion when it hits the surface. However, if less than 50% of its mass hits the surface, it will roll off and fall to whatever is below. The image below demonstrates these two scenarios.

Another example is when an object is dropped with a velocity in the x direction (left and right). As shown below, the object will not just hit the surface and not move; it will also slide or roll in the case of a round object, on the surface. How much it rolls depends on the x velocity as well as the shape of the object. For example a round object would be more likely to roll in the x direction after the collision, where an object with flat sides is less likely to move. In the case of an object with flat side, the size of the sides also make a difference, the longer the sides the less likely it is to roll and the shorter it is, the more likely it is to roll.

Forces also have a big effect on the results of collisions, after all, an object wouldnt move without forces. For example, a surface with a low amount of friction (i.e. ice) would allow an object to slide on it, but an object with higher friction, (such as concrete or rubber) will not let many objects to slide on it, however, if the object has enough force on it, it could slide or even role due to the large amount of friction. Gravity, wind, as well as other forces would also effect collision since they could cause the objects to be pushed in different directions.

Elasticity of an object that has a collision also could have a big effect. A collision in which the elasticity if 1 (perfect) will not lose any energy in the collision (this is shown in our lab with the bouncing ball. The ball would never stop bouncing and always return to the same height). With a value lower than 1, the objects will lose energy (once again shown in a lab, the bouncing ball this would eventually stop bouncing). In the case where it is 0, the objects that collide will stay stuck together (the bouncing ball will not bounce once it hits the floor).

5 Conclusions

Working on this project, more valuable knowledge of how rigid bodies interact was obtained and also observed. We started out with the idea of creating the a 2D interactive game with rigid bodies, integrating Motion for the rigid bodies including State Variables, Collision Detection and Collision Handling.

Having a clear Model in place, we were able to smooth implement and develop the game using the programming language Java with openGL. Physics in the model was implemented to integrate with the game. We encountered a few problems while including the physics into the game, but were able to debug the errors successfully.

Future extensions of the project would be improving the textures used in the game simulation for the rigid bodies. Also development of the game could be done to release the game unto mobile platforms such as iPhone or Android platforms.

References

[1] Dwaipayan Chakrabarti and David J. Wales. Simulations of rigid bodies in an angle-axis framework. *Phys. Chem. Chem. Phys.*, 11:1970–1976, 2009.

[2] Michel Frmond. Rigid bodies collisions. *Physics Letters A*, 204(1):33 – 41, 1995.

[3] Erik Neumann. Myphysicslab rigid body collisions http://www.myphysicslab.com/collision.html, 2004.