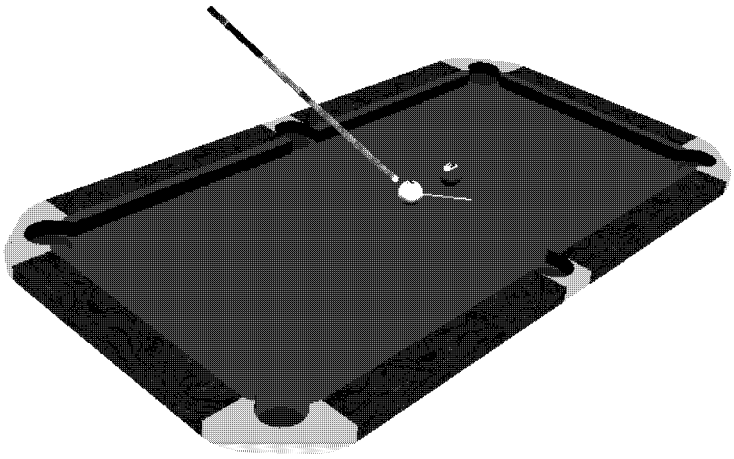
6.837 Final Project

Pierre Poignant, Esther Yoo

December 6, 2002



1 Abstract

We implemented a physic-based billiard simulation. The physics of the simulation is not exact, but to make the simulation look accurate enough to users, we approximated the main physical effects of a billiard game, such as sliding, rolling and spinning of the ball.

2 Introduction

There are many kinds of billiard games. Depending on the game, size of balls and shape and size of the table differ. We did not design our simulation for any specific game. We made it possible to change parameters for any kind of billiard game. We used Java 3D.

Java 3D

Java 3D is a client-side Java API developed at Sun Microsystems for rendering interactive 3D graphics using Java. The OpenGL API is written in the C programming language, and hence not directly callable from Java. A number of open source and independent programming efforts have provided simple Java wrappers over the OpenGL API that allow Java programmers to call OpenGL functions, which are then executed in native code that interacts with the rendering hardware. Java3D fills an important gap between VRML, which is centered on describing 3D content, and OpenGL, which is a C API for rendering points, lines, and triangles.
The foremost strength of Java 3D for Java developers is that it allows them to program in 100 percent Java, and therefore produce a portable code. The Java 3D API has a lot more to offer to the application developer. By allowing the programmer to describe the 3D scene using coarser-grained graphical objects, as well as by defining objects for elements such as appearances, transforms, materials, lights, and so forth, code is more readable, maintainable, reusable, and easier to write. Java 3D uses a higher-level scene description model, the scene graph, which allows scenes to be easily described, transformed, and reused.
The main problem of using Java 3D is performance, one particular problem, inherent in Java, is the impact of the Java garbage collector. If garbage collection occurs in the middle of a critical animation sequence, the realism of the rendered scene may be lowered for the user. An other problem is that the Java client-side API, and especially Java 3D, can be difficult to distribute to users.

Physics Simulation

First problem we encountered was to model a complex physical model. Modeling the motion of a billiard ball accurately is very complex. Therefore, we made approximated model for a credible simulation. There were three tasks needed to be done: the motion of the billiard balls, the collision detection and the collision response.

3 Goals

Our project was to produce a physic-based 3D pool simulation. One of the main goals of our project was to make the game extensible and scalable. Even though five weeks were not enough time to include all the features we wanted, we did not want to limit the possibility of adding more features. This project introduced us to Java 3D API.

4 Achievements

Aiming our project

The main obstacle we faced was to bound our project. We thought the physics would be quite straight forward. But while writing the equations of the motion was easy, solving them and reproducing the physics behind was hard. The simplest simulation and hypothesis were too easy to implement, and the most complex simulation was to lead us to a lot of debugging and to a hard time controlling the accuracy of our simulation. Therefore, we chose to simplify the equations of motion and collision response: we have decorrelated the horizontal and vertical spin and have simplified the collision response. In addition, we assumed that the motion is straight.

Code Structure

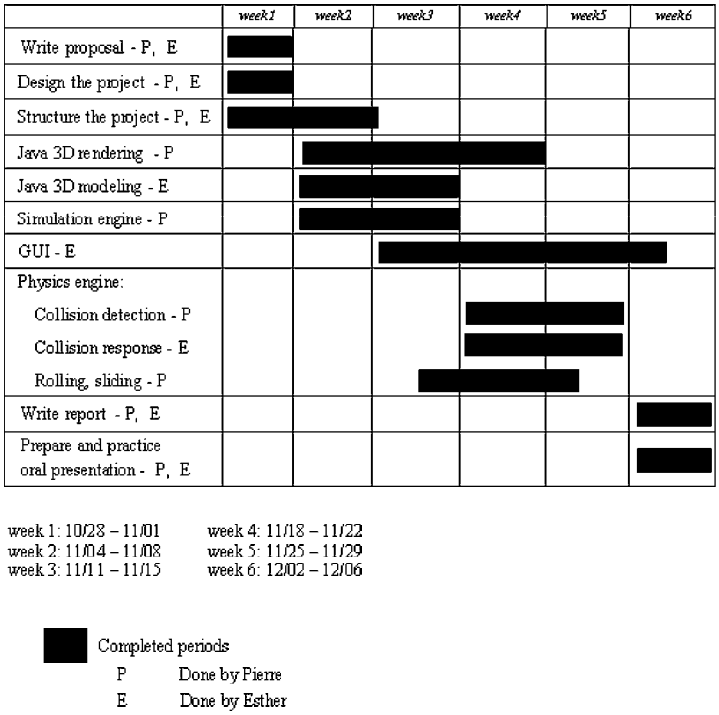
We focused on the structure of the project and we spent a good amount of time together in order to define a structure that both of us would be confident to work with. This resulted in belated schedule in the beginning. After the first two weeks, we only displayed the table and one ball. But by that time, we had settled all the underlying structure of the project. This enabled us to concentrate on the simulation.
The main idea of our structure is that there are three main classes that handle all the others : World.java, View3D.java and PoolGui.java. World.java takes care of the simulation, View3D.java takes care of all the Java 3D structure, and PoolGUI.java takes care of the GUI. Therefore, when we create a simulation, we first create those 3 main classes, then we create PoolElement that we had to the world. Depending on the class the Object belongs to and of the interface the object implements the world will perform some actions. Our main work has been to specify this contract : what is going to be done by the world if the objects implements the interface HasNode3d ... Once we had all the scenarios settled we have been able to work very efficiently.

Multiple Collisions

We have not been able to explore multiple collision response. The multiple collision occurs when the player breaks the balls in the beginning of a game. If we use our current algorithm to solve this problem, it will never end. However, if we decide to stop the algorithm after a fixed amount of iterations, the result will not be correct because of energy losses. And this will result in not accurate visualization. Therefore, we wanted to explore some global approach of the problem, but we did not have enough time to figure this problem out.

5 Individual Contributions

Here are detailed the schedule of the different contributions :



6 Lessons Learned

Java 3D

This project has introduced us to the Java 3D API. This API might not be as efficient and as flexible as a C-OpenGL code, but it enabled us to get the result very fast because it was easy to use Java programming language. Another benefit of using Java3D was that we were able to produce a scalable application.

Team work

The team coordination has been easier, because we were a two people team. But we still had to come up with way to work together. To understand the way each of us would code and structure the project, we decided to work and code together for the first week. Once the structure and the methods were clear to both of us, then it was much easier to communicate and to work separately, and

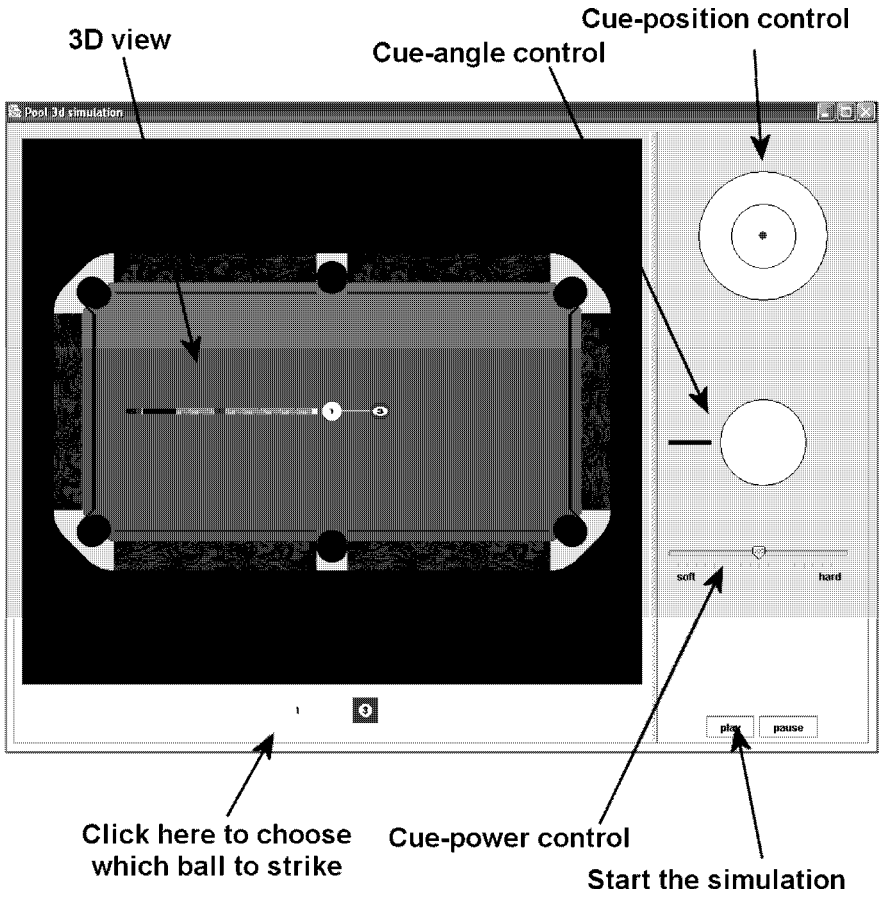
therefore we have been to improve our productivity through the project. We think this method was successful!

Milestones benefits

We thought the weekly meetings were very helpful. First it obliged us to work regularly, we were a little bit late schedule at the beginning of the project, and the meeting forced work more to stay in track with our initial schedule. In a way we had the project rush in the middle of the project, and not at the end of it ! Second having an exterior point of view at what we were doing led us to come up with answers to some questions that we had not thought about in the first place. In our next software engineering projects, we will try to learn from this experience and apply it to our methods.

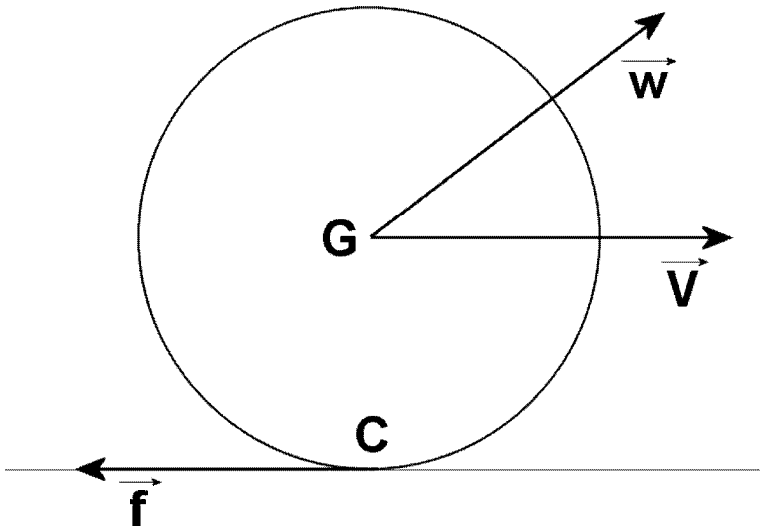
The GUI

We wanted to pay a lot of attentions to the user-interface, but on the other hand we knew that coding a nice GUI is hairy, therefore we decided to keep it simple, efficient and flexible. We had to create our own layout manager, that would stand separately from the main classes. If the object needs to display something (a control, a panel) on the GUI, it just provides to the GUI manager a JComponent object.


3D view
Cue-angle control
Cue-position control
Pool 3d simulation
soft
hard
play
pause
Click here to choose which ball to strike
Cue-power control
Start the simulation

7 Description of Deliverables

7.1 Physical Model



7.1.1 Rolling and Sliding

When a billiard ball is hit with the cue stick, the ball starts moving across the table. If the ball is struck along its center of mass, the ball is not initially rotating but only sliding.
However, soon the ball starts rolling along. The friction between the ball and the table causes this rolling motion to occur. The force of friction applied to a body sliding on a surface is calculated by the following formula :

$$f = \mu_s mg$$

The friction force is applied in the direction opposite the velocity. Since this force is applied to the surface of the ball and not its center of mass, the friction force causes angular acceleration in the ball. As the ball rolls across the table, the angular velocity increases because of this sliding friction force. This continues until a time of equilibrium is reached, where the velocity of the point contacting the table equals the velocity of the center of mass. At this time, the ball is no longer sliding and is now rolling on the table. This situation happens when :

$$v = Rw$$

The equations for the speed and the moment are :

$$\begin{aligned} \frac{dv}{dt} &= -\mu_s g \\ \frac{dw}{dt} &= \frac{5}{2R}\mu_s g \end{aligned}$$

We can therefore deduce the time until natural roll :

$$t = \frac{2(v_i - RW_i)}{7\mu_s g}$$

we get as well the speed at the time of natural roll :

$$v_c = \frac{5}{7}v_i + \frac{2}{7}Rw_i$$

7.1.2 Collision Response

As a starter we derive the collision equations assuming that balls do not rotate. Then we calculate the angular impact equations as well.
First we can write the incoming and outgoing center of mass velocities under the influence of the unknown impulse.

$$\begin{aligned} \boldsymbol{v}_2^A &= \boldsymbol{v}_1^A + \frac{j}{m^A}\boldsymbol{n} \\ \boldsymbol{v}_2^B &= \boldsymbol{v}_1^B - \frac{j}{m^B}\boldsymbol{n} \end{aligned}$$

Where $\boldsymbol{n}$ is the normal vector of the collision and j is an unknown scalar.
These equations assume that there is no friction during the collision.

The other equation is the Newton's Law of Restitution :

$$\boldsymbol{v}_2^{AB} \cdot \boldsymbol{n} = -e\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}$$

Where is the coefficient of restitution : $e = 0$ for a total plastic collision and $e = 1$ for a total elastic collision,
and $\boldsymbol{v}_i^{AB} = \boldsymbol{v}_i^A - \boldsymbol{v}_i^B$.
We use the 3 previous equations to solve get the expression of j :

$$j = \frac{-(1+e)\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}(\frac{1}{m^A} + \frac{1}{m^B})} \tag{1}$$

We can notice that $\boldsymbol{n}$ does not need to be a normalized vector (avoiding the square root). The second thing to notice is that these equations handle the ball-ball collision as well as the ball-table collision: we just need to take an infinite

mass for the static object (the table).

We can now write the equations in the general case with both linear and angular velocities :

$$\begin{aligned} \boldsymbol{v}_2^A &= \boldsymbol{v}_1^A + \frac{j}{m^A}\boldsymbol{n} \\ \boldsymbol{w}_2^A &= \boldsymbol{w}_1^A + \frac{1}{I^A}\boldsymbol{r} \wedge \boldsymbol{n} \end{aligned}$$

were $\boldsymbol{r}$ is the vector from the center of mass to the point of collision.
The resolution of these equation yields to :

$$j = \frac{-(1+e)\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}(\frac{1}{m^A} + \frac{1}{m^B}) + [(I_A^{-1}\boldsymbol{r}^A \wedge \boldsymbol{n}) \wedge \boldsymbol{r}^A + (I_B^{-1}\boldsymbol{r}^B \wedge \boldsymbol{n}) \wedge \boldsymbol{r}^B] \cdot \boldsymbol{n}}$$

Here as the two objects are spheres, $\boldsymbol{r}$ is parallel to $\boldsymbol{n}$ the angular velocities have no influence on the collision.
There fore we need to model some friction during the collision if we want to incorporate the effects of rotation.
We are going to assume that the response is of the form:

$$\begin{aligned} \boldsymbol{v}_2^A &= \boldsymbol{v}_1^A + \frac{1}{m^A}\boldsymbol{P} \\ \boldsymbol{v}_2^B &= \boldsymbol{v}_1^B - \frac{1}{m^B}\boldsymbol{P} \end{aligned}$$

where

$$\boldsymbol{P} = j(\boldsymbol{n} - f_c\frac{\boldsymbol{v}_S}{\|\boldsymbol{v}_S\|})$$

Where j is the first j computed before, f_c a new friction coefficient and $\boldsymbol{v}_S$ is the relative surface velocity.
To get the speed of the balls we need to project these equations on the plane.
We model as well the collision response for angular momentum

$$\begin{aligned} \boldsymbol{w}_2^A &= \boldsymbol{w}_1^A + \frac{1}{I^A}\boldsymbol{r} \wedge \boldsymbol{P} \\ \boldsymbol{w}_2^B &= \boldsymbol{w}_1^B - \frac{1}{I^B}\boldsymbol{r} \wedge \boldsymbol{P} \end{aligned}$$

Here for the simulation we choose $e = 1$ for ball to ball collision, and $e = 0.5$ for ball-table collision.

7.2 Algorithm

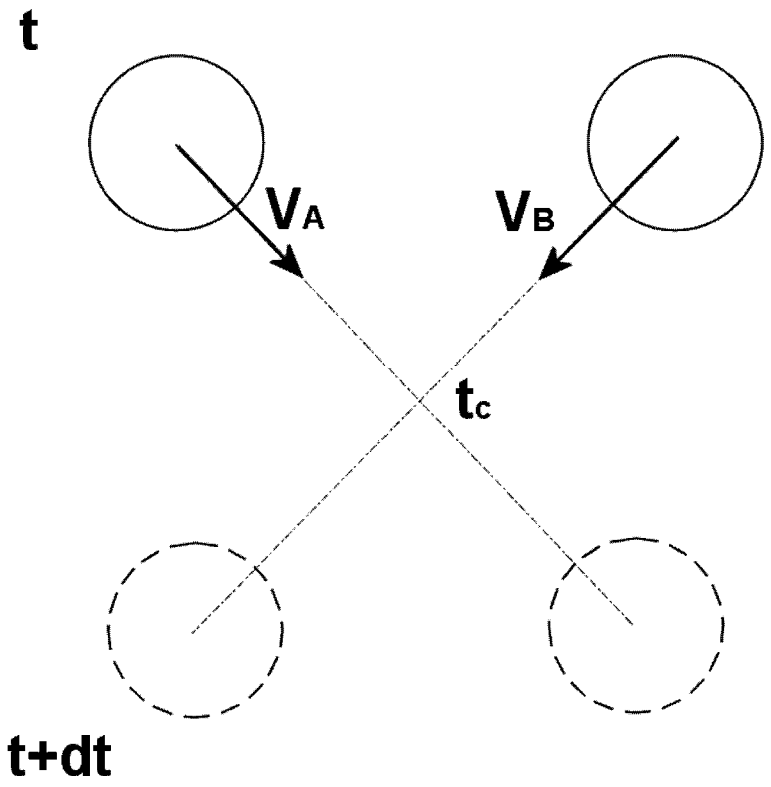
7.2.1 The loop

The algorithm is quite straightforward. We let users interact with the pool table and choose their parameters for the ball. They can decide where and with what

strength they want to hit the ball with the cue-stick. They can also decide the direction of the ball. When users have chosen their parameters, they start the simulation. The program iterates through the following loop, until all the balls on the table stop
check collision in the next step
solve collisions that will occur
move the balls to the next step $(t + dt)$
render

7.2.2 Collision Detection

The naive algorithm is the following: at some initial instant t we know the positions and velocities of all the balls on the table. We plug those numbers into the equation of motion and find the positions and velocities at $t + \delta t$. To make the ball bounce, we have to encumber the algorithm with a check for collision after every time step. That is, whenever we move a ball, to a new position, we have to calculate its distance from every other ball on the table. If the distance is smaller or equal to the sum of the two balls radius, a collision has occurred, and we need to do something about it.
Several aspects of this algorithm are not optimal. First this algorithm has a complexity of $O(n)$ ($\frac{n^2-n}{2}$ collision-tests) but for a pool table we will assume that it is OK (we could partition the table for example). The most important remark is that by the time a collision is detected, the balls have already penetrated each other's volume. Furthermore, this algorithm might not even detect collision for some values at all.
Therefore, we need to test at each step if a collision will occur in the next step and if so, we compute the collision of the closest time and move the balls to this collision point in order to compute collision response.


t
VA
VB
tc
t+dt

7.3 Code description

7.3.1 package pool3d.world

class World.java Main class of the simulation, the world controls the main algorithm and all the objects used by the simulation.

interface HasWorld.java Every objects that has a pointer to the world implements this interface.

interface PoolElement.java Every object that gets added to the world implements this interface.

7.3.2 package pool3d.objects

class DirectionVector A DirectionVector gets displayed on the 3d scene and represents a direction (can be a moment, a speed, the cue direction).

class PoolCue The PoolCue is the 3d representation of the cue in the 3d scene.

class PoolTable The PoolTable gathers all the information about the table (walls, holes...), and holds as well a pointer to its 3d representation.

abstract class PoolSegment The holes and the walls on the table can be treated the same way, and therefore extends this abstract class.

class PoolHole The holes of the table implement this class.

class PoolWall The walls of the table implements this class.

class PoolAngle When walls and holes get added to the table, the table contains some gaps, the PoolAngle class enables to fill those gaps.

class PoolBall Every ball on the table implements this class.

7.3.3 package pool3d.gui

class HasJPanel Every pool-element that gets displayed one the screen implements HasJPanel.

class PoolGUI The PoolGUI is the frame that gets displayed. It contains all the other graphical components. When a PoolElement gets added to the GUI, the PoolGUI calls the method getJPanel(), and displays the panel depending on the type of the PoolElement.

class WorldControls This class contains the buttons that stop and start the world.

class CueBallInteraction This class contains the panels that enable the cue-ball interaction.

class BallPane Class used by CueBallInteraction for the position of the strike on the ball.

class CuePane Class used by CueBallInteraction for the angle of the cue with the ball.

7.3.4 package pool3d.graphics

In this package we put all the java classes that are supports for the Java3D scene representation.

class FrameCheck This class extends a Java3D behavior and gets called every time a frame is rendered by Java3D. Then this class notifies the world that Java3D has rendered a frame.

class HasNode3d Every PoolElement that gets displayed in the Java3D scene implements this class.

class Node3d This class extends the Java3D class BranchGroup that is an object that can be displayed on the Java3D scene. It contains method to rotate, translate the object, to apply texture and to change the color, geometry of the Java3D shape.

class Pool3dKeyBehavior This class handles the key events generated by the Java3D scene.

class Pool3dOrbitBehavior This class handles the mouse events generated by the Java3D scene.

class View3d Main Java3D class: it contains pointer to the scene and to the 3D utility class. When this class gets implemented it will generate a Java3D universe for the display of the pool simulation.

7.3.5 package pool3d.physics

In this package, we have put the collision detection and response algorithms.

8 Acknowledgments

We would like here to thanks our TA Jingyi Yu for his advice and the way he helped us to target and schedule the coding and the modeling of our project.
We would like as well to thank our "significant ones" Eeleen and Seongmoo who have been very understanding and always let us cancel our dates to meet each other to make the project go.
Special thanks to Alexandre for discovering the online book about Java3D, and to Jae-Ho for cooking us nice Korean dinner during the last coding rush.

9 Bibliography

Although lots of resources about pool simulation are available on the internet, only a few web-site give good physic resources. We currently have found two interesting articles by Miles Jackson on gamasutra.com.
Therefore, we made intensive use of books on the subject, like "Amateur Physics For The Amateur Pool Player" by Rom Shepard or "The Physics of Pocket Billiards" by W.C. Marlow.
The Java3D API is very well documented on internet, also the sun tutorials and API specifications have been very useful. But we have mostly relied on the book "Java 3D Programming" by Daniel Selman available online for free on http://www.manning.com/selman/onlinebook/.