Coursework Report

Roderick Ewles
40330977@napier.ac.uk
Edinburgh Napier University - Physics Based Animation (SET09119)

Abstract

The aim of this project was to model the physics of a pool table focusing on collision detection and friction. The project is split into three parts. Part 1 considers sphere-sphere collisions without friction. Part 2 considers how efficiently collisions can be modelled, again without friction. Part 3 considers how friction affects the pool balls on the table.

1 Sphere Collisions

The first part of the project was to get a small number of pool balls to bounce off each other correctly. Collision response was calculated using conservation of momentum. The main equations used were:

$$V1' = V1 - ((2*(s1-s2))/(m1+m2))*m2*n$$

$$V2' = V2 + ((2*(s1-s2))/(m1+m2))*m1*n$$

Where s1 is the component of ball 1's velocity acting along the collision normal, s2 is the same but for ball 2, m1 and m2 are ball 1 and 2's masses respectively, n is the collision normal and V1 and V2 are the initial velocities for the pool balls.

This works on the assumption that the pool balls are perfect rigid spheres and as such there is only one point of contact between the two spheres. As the second circle will gain any momentum lost by the first this change in momentum can be represented by a vector(the final version of the equation shown above calculates this vector and converts it to a velocity that can be added or subtracted from the ball's initial velocities).

This method of collision response is applied throughout all parts of this project. However, it is moved to be a method for an object in part 2 and is modified for part 3.

2 Scalability

The goal of part 2 was to optimise the implementation of the code to get as many pool balls on the table as possible while maintaining a smooth simulation at a consistent frame rate. Several techniques were implemented to achieve this and a final count of 5750 pool balls were successfully simulated.

The main technique used to allow the model to scale was the application of a uniform grid [1]. This is where the space being considered has a uniform grid of cells overlaid. this effectively splits the space up into smaller areas. Collision tests are only carried out on spheres that are within the same cell. This allows fewer collision detection tests to be carried out which in turn improves the efficiency of the simulation allowing more objects to be animated. The grid size implemented was 20x20. Experiments were carried out with a larger number of cells (50x50), but this was found to reduce performance.

The sphere objects had methods added to determine which column and row the sphere is currently in. The edge bounds go beyond the edge of the table so that pool balls with a high enough velocity to pass the edge of the table are still allocated to these edge cells to allow collisions with the cushions, otherwise these fast moving balls would escape the table.

Once the spheres have been determined to be in a cell or cells, the collision tests are carried out. The collision detection and response is implemented in the cell object. Once the collision test has been carried out and the spheres are determined to be colliding the response is then applied.

In the part 1 implementation every ball is tested to see is it comes into contact with the cushions so that a collision response can be applied. However, in part 2 there are too many pool balls on the table for this to be feasible. Instead only the cells that are on a particular edge carry out the test for that cushion, further improving efficiency.

The grid itself was initially stored in a vector of vectors. However, the vectors would re size themselves until the vectors became very large. Instead of this a two dimensional array is used to store the grid objects as they are of fixed size they do not suffer this problem.

Similar to part 1 the initial implementation manipulated the objects themselves which proved to be very memory intensive. To solve this problem, pointers were stored to objects rather than the objects themselves. For example the two dimensional array that represents the grid stores pointers to the respective cells rather than the cells themselves. The same is true of the vector that stores the spheres and the vectors of spheres in the grid objects. This allows the simulation to handle far more objects than would otherwise be possible.

3 Pool Physics

The final part of the project looks at the effects of friction and rotation on the behaviour of the modelled pool balls. Before any implementations were considered some experiments were carried out with a real pool table to see if any useful observations could be made. One interesting observation was that if any angular component is transferred between the cue ball and the target ball, it is imperceptible to the human eye. Even when a lot of angular velocity was imparted to the cue ball the target ball would gain only a linear velocity. The angular component of the cue ball is extremely important in pool as it allows a good pool player to position the ball in such a way that they control the game. Therefore the angular component applied to the cue ball and its interaction with the felt of the table is crucial to how the game works. Following the design philosophy "if it looks right it is right", these real world observations are what was attempted to be implemented.

As:

$$Ff = u * Fn$$

An approximation for the effect of the friction on the pool ball's velocity was:

$$V' = V - V * U * dt$$

The same approach was applied to calculate the angular component of the pool ball. This gives quite realistic looking behaviour in pool balls without spin, and allows spinning balls to slow down at a natural looking rate. Friction will also cause a ball with only a linear velocity to roll which means that the ball should rotate at a rate that is in proportion to its linear velocity. This is implemented in the integrator and as stated in Coulomb's law, can be treated separately to the friction based effects of the ball spinning.

The next challenge to be considered is the pool ball spinning and how that effects the ball's trajectory. As the model is based off of conservation of momentum, the work done by the frictional force is considered instead of the force itself. Another important factor that was considered is that the spin will have a different effect on the trajectory of the ball depending on which axis the ball is spinning on. In effect, the x and z axis spin cause the top and bottom spin effects, and the y axis spin is responsible for causing the pool ball's trajectory to curve (called "english").(It is possible to create a curved trajectory using a velocity on the z axis and spin on the z axis but this type of shot would be physically impossible to make, therefore this ca is not considered.)

Friction causes both angular and linear components to decrease. Most would become heat, but some of the angular component will be converted into a change in linear velocity. This change in linear velocity is what causes the effects observed in shots with top spin, back spin and curved trajectories. This is done by calculating the tangential velocity of the object at the given time with respect to the axis the ball is spinning on. Tangential velocity is calculated in the following way:

$$Vt = rW$$

Where r is the radius and W is the angular velocity.

In top spin and back spin the spin causes the pool ball to either accelerate or decelerate linearly on the tangential axis to the spin, i.e. positive x axis spin will accelerate the ball in the z direction. This is in effect a change in velocity at each time step. This velocity is calculated as follows(the example is x spin resulting in a linear velocity being added in the z direction):

$$V = r * W * u$$

This calculation is carried out for both the x and z components to allow diagonal trajectories with top spin and backspin to work to allow the model to look realistic. This is necessary as in reality the top spin or back spin should act in the same direction as the shot. Real pool players prefer to make positional shots using only top spin or back spin without any english as english causes the ball's trajectory to change from a straight line making the likelihood of missing the shot greater. The first executable shows a shot with an extremely large amount of back spin stopping the ball and causing it to roll backwards a small amount. This shot probably wouldn't be seen in a real game of pool and is more in the realm of a "trick shot".

Y axis spin or english is a little trickier to implement as the balls current direction of travel must be taken into account. Then depending on whether the spin is clockwise (negative) or anticlockwise. The resulting curve or tangential velocity will act in the same direction as the spin. In effect the spin should result in a change in the angle the linear velocity is acting. This is approximated by using the above tangential velocity calculation and reducing one axis' linear velocity while increasing the other axis' linear velocity. With lower angular velocities this curvature is hard to see, so the part two executable also has a large spin applied to show this working. This implementation is necessary to allow the modelling of shots like a back spin (or draw shot) with english.

The final two executables show examples of the shot described above from different angles. This allows the effects of both axes of spin on the final trajectory of the pool ball to be seen. Another consideration is the fact that in most cases the curved trajectory of the cue ball in these shots is not only caused by y axis spin but also the way the cue bends during these shots acting like a cantilever and applying an additional tangential velocity.

4 Conclusion

In Conclusion, parts 1, 2, and 3 were attempted, the model produced was based on real world observations and research consisting of watching pool players explain the different effects of these shots and how they are used as well as reading books and websites [2] [3]. Part 3 could be expanded, particularly when dealing with collisions with the cushions and the effect y axis spin has as in this case there will be an even greater friction force for a brief instant changing the angle of the bounce. This is currently only implemented in the normal friction function.

References

[1] C. Ericson, *Real Time Collision Detection*. San Francisco, CA, USA: Elsevier, 2005.

[2] D. Eberly, *Game Physics 2nd edition*. Burlington, MA, USA: Elsevier, 2010.

[3] M. e. a. Jackson.