gl-billiards

Dan Lapp
Taras Mychaskiw
COSC 3P98 Project

January 8th, 2014

Abstract

Contents

1 Physics Engine

1.1 Taking a Shot

Once a ball is hit by the cue, there are two ways energy is transferred to the ball. First, and more noticeable of the two is the transfer of linear momentum. This sends the ball moving around the table in straight lines. Second is the transfer of angular momentum, which applies spin to the ball. Spin can cause the ball to stop in it's tracks after a collision, bounce off cushions in strange (but predictable) ways and to any pool player, is just as if not more important an aspect to the game as the linear momentum.
To simplify the shot making process, the pool cue is assuming to act like a spring. That is, the further you pull the cue away from the cue you are going to hit, the more powerful your shot will be. Spring motion can be described using Hooke's Law,

$$\vec{F} = -k\vec{x} \tag{1}$$

where, $\vec{F}$ is the force, $\vec{x}$ is how much the spring is stretched and k is the spring constant, which describes basically how powerful (springy) the spring is.
Hooke's Law is equivalent to Newton's Second Law, which states

$$\vec{F} = m\vec{a} \tag{2}$$

where, $\vec{F}$ is the force, m is the mass of the object and $\vec{a}$ is the acceleration of the object. Setting Equation 1 equal to Equation 2 gives

$$\begin{aligned} -k\vec{x} &= m\vec{a} \\ -k\vec{x} &= m\frac{\Delta\vec{v}}{\Delta t} \\ \Delta\vec{v} &= \frac{-k\Delta t}{m}\vec{x} \end{aligned}$$

where $\vec{v}$ is the velocity of the object, and t is the time. For pool, the initial velocity of the ball being struck is always zero, so we can simplify the above equation to

$$\vec{v} = \frac{-kt}{m}\vec{x} \tag{3}$$

where $\vec{x}$ is the distance between the ball and the tip of the cue, m is the mass of the ball, t is how much time the cue is in contact with the ball and k is the spring constant of the cue. The values for t and k needed to be experimentally found though trial and error, essentially trying values until what happened in the simulation closely represented real life. The values for t and k were found to be 0.01s and 200N/m.

1.2 Event System

For the purposes of this program, an event is a ball-ball collision, or a ball banking off of the cushion. The way gl-billiards works is each time the screen is redisplayed, some physics

updater function is called with a parameter *dt*, which is how much time has passed since the last frame. The engine simulates the pool game up until a time *dt* has passed. For example, if 1s was passed, then it would simulate 1s of play before returning, then the screen would be updated as if 1s had passed.
So, the engine determines which events occur within time *dt*. Event detection then becomes *when* do events happen instead of *if* they happened. The entire update function works as follows:

```
update($dt$)

  find earliest event that occurs within time $dt$, call it $event$
  if no such event

    roll balls until $dt$

  otherwise

    roll balls until $event$ occurs
    handle $event$
    $dt$ <-- $dt$ - $event.getTime()$
    call update($dt$) again
```

In this manner, all events that happen within *dt* and handled automatically before the frame updates and the balls slow down due to friction. Note that "roll balls" simply means each ball moves at it's speed for some amount of time. Pocketing detection needs to be done there, as it is not a standard event. That is, if a ball is pocketed, it cannot possible participate in any events in time *dt*, since it would bank off the cushion or collide with another ball first.
After *update*(*dt*) is finished, each ball gets affected by friction and slows down slightly. Friction takes the form of a slight decrease in the velocity of the ball.

1.3 Event Detection

1.3.1 Ball-Ball Collision Detection

A ball-ball collision will occur when the centers of the balls are two radii apart. In gl-billiards, the trajectory of each ball is simplified to be a straight line per frame since friction is only applied at the end of each frame. Thus, the trajectory of each ball becomes,

$$\vec{r} = \vec{r}_0 + \vec{v}t \tag{4}$$

where $\vec{r}$ is the position of the ball, $\vec{r}_0$ is the starting position, $\vec{v}$ is the velocity of the ball and t is the amount of time elapsed. Then, for each possible collision to occur, the program needs to find at what times the distance from one ball to the other is equal to twice the radius of the ball. Mathematically, this becomes

$$\begin{aligned}|\vec{r}_1 - \vec{r}_2| &= 2R \\ \sqrt{(\vec{r}_1 - \vec{r}_2)\cdot(\vec{r}_1 - \vec{r}_2)} &= 2R\end{aligned}$$

where R is the radius of a ball. Squaring both sides, and subbing in Equation 4 for both $\vec{r}_1$ and $\vec{r}_2$ (using $\vec{dr_0} = \vec{r}_{0_1} - \vec{r}_{0_2}$ and $\vec{dv} = \vec{v}_1 - \vec{v}_2$),

$$(\vec{dr_0}\cdot\vec{dr_0}) + (\vec{dr_0}\cdot\vec{dv})t + (\vec{dv}\cdot\vec{dv})t^2 = 4R^2 \tag{5}$$

This is a quadratic equation for t which can be solved easily. If there are real roots to the equation, then a collision occurs, and it occurs at the minimal possible t value which is a solution to Equation 5.

1.3.2 Ball-Wall Collision Detection

Ball-wall collision detection is similar to ball-ball, except the ball is a distance one radius away from the wall when it collides. Since the wall doesn't move, a different strategy had to be employed to detect when a ball-wall collision occurs. In this case, both the and wall positions are described using parametric equations.

$$\vec{r} = \vec{r}_0 + (dt\ \vec{v})T_b \tag{6}$$

$$\vec{w} = \vec{s} + (\vec{f} - \vec{s})T_w \tag{7}$$

Where, $\vec{r}$ is the position of the ball, $\vec{r}_0$ is the initial position of the ball, $\vec{v}$ is the velocity of the ball, dt is the amount of time that will pass this frame; $\vec{w}$ is a point on the wall, $\vec{s}$ is one endpoint on the wall, and $\vec{f}$ is the other endpoint of the wall. T_b and T_w are the parametric variables, and range between 0 and 1.
To determine when the ball hits the wall, where the ball hits must be determined first. The center of the ball hits the wall when $\vec{r} = \vec{w}$, thus the ball actually collides with the wall when $\vec{r} - R\vec{n} = \vec{w}$, where $\vec{n}$ is the normal vector to the wall. For simplicity below, we will ignore this offset value, as it is a fairly simple transformation. So, the collision occurs when

$$\begin{aligned}r_{0_x} + (dt\ v_x)T_b = s_x + (f_x - s_x)T_w \\ r_{0_y} + (dt\ v_y)T_b = s_y + (f_y - s_y)T_w\end{aligned} \tag{8}$$

Solving for T_b and T_w gives

$$\begin{aligned}T_b &= \frac{(f_x - s_x)\cdot(s_y - r_{0_y}) - (f_y - s_y)\cdot(s_y - r_{0_y})}{(dt\ v_y)\cdot(f_x - s_x) - (dt\ v_x)\cdot(f_y - s_y)} \\ T_w &= \frac{(dt\ v_x)\cdot(s_y - r_{0_y}) - (dt\ v_y)\cdot(s_y - r_{0_y})}{(dt\ v_y)\cdot(f_x - s_x) - (dt\ v_x)\cdot(f_y - s_y)}\end{aligned} \tag{9}$$

If both T_b and T_w are between 0 and 1, then a ball-wall collision occurs. The point at which can be found by subbing in T_b or T_w into Equations 6 or 7 respectfully. Then the amount of time passed can be solved for using Equations 4.

1.4 Event Handling and Velocity Updates

1.4.1 Ball-Ball Collisions

1.4.2 Ball-Wall Collisions

1.4.3 Friction