

# CS224 Assignment 4: Animation

**Out: Thursday, 15 February**

**In: Wednesday, 21 February, 11:00 pm**

---

## Overview

In this assignment, you'll be doing two things: writing a simulator for a very simple colliding-pool-ball simulation, and applying Chenney's MCMC algorithm to search for animations on this pool table satisfying constraints. Chenney's paper is located at [course/cs224/asgn/anim/chenney.ps](#). To get you started, I'll provide a small warm-up exercise; don't bother handing it in, but *do* make sure to actually do it.

## Warm-up

Implement the MCMC algorithm on a very simple state-space. The state space in this case will be the interval  $[0, 1]$ . The desired distribution of points is proportional to

$$p(x) = \left| x - \frac{1}{2} \right|$$

Your first transition function should be “take the current value  $x$  and add to it a number  $u$  chosen from a uniform distribution on  $[-.25, .25]$ ; then take the number mod 1” (i.e., if it's less than zero, add 1; if it's greater than 1, subtract 1). Run the MCMC algorithm with this transition rule and desired density function, generate a thousand points, and plot them to see how well you did.

Your next transition function should do the same thing, except that it should add a number chosen from a uniform distribution on  $[-0.1, +0.2]$ . Note that computing the function  $q(Y | X)$  is somewhat harder.

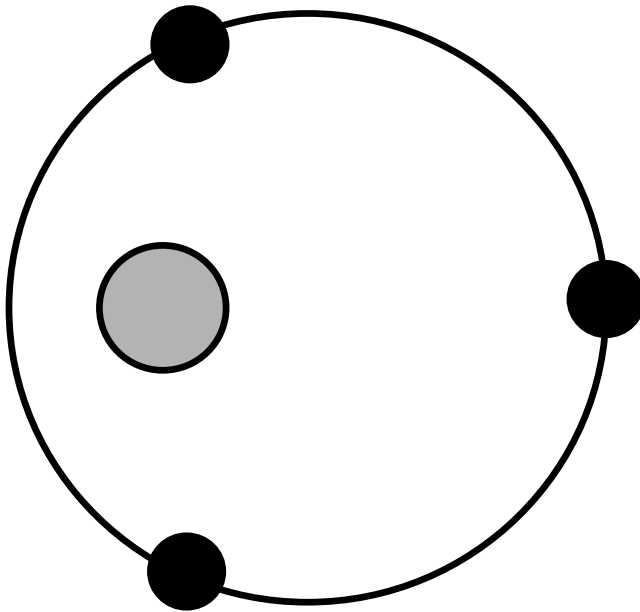
## Into the fire

Next, write a simulator. It should simulate the motions of  $n$  disks of radius 1 (“pool balls”) in a 2D world consisting of a round billiard table of radius 10 in which there are round “bumpers” placed at various locations. You may restrict to the case where the bumpers do not intersect, and are at least 2 units from each other or the edges of the billiard table. There are three “pockets” in the billiard table, each of which is a disk of radius 2, centered on the outer circle and distributed at 0, 120 degrees, and 240 degrees. The rule for collisions is easy: if a pool ball collides with a bumper or the edge of the table, it bounces off with angle of reflection = angle of incidence (although your algorithm will modify this rule slightly to provide “plausible variation”). After the collision, it moves at the same speed with which it came in.

---

If two balls collide, they behave the same way: each looks like a bumper to the other. (If you want to do something fancier, you can...but it's not necessary).

What about the pockets? If a ball travels completely into a pocket, it stops there and disappears. If it's only partly in a pocket, you should pretend that the pocket isn't there at all and make it bounce as if the wall of the table were there.



This structure for pockets is intentionally simple: you can detect whether a ball is completely in a pocket by placing an artificial “bumper” of radius 1 concentric with the pocket: if the ball touches this bumper, it's “completely in” the pocket; if not, it's not.

Your simulator might work like this: take all the balls and the bumpers and the rim of the table and compute for each ball, what it will collide with if it travels in a straight line; it should then enter that collision (and the time of collision) into a priority queue. You should then extract the first collision from the priority queue, and “move everything forward to that time.” (All motion of the balls is constant speed between collisions, for simplicity, though you should try to have some balls be initially stationary as in the example). You must then respond to the collision, changing the motion of the ball(s) that collided, and removing all their future collisions from the queue. Then you need to compute all their expected future collisions and enter them in the queue, and you're ready to start over. (This doesn't address the pockets...that's for you to think out). [If you think you have a better way to write the simulator, go ahead. This is meant to encourage experimentation with MCMC, not to have you spend hours writing messy code.]

NOTE: DO NOT try to do this simulation by taking Euler steps! That is you do not have to increment time by a small delta and recalculate the position of every ball and try to detect collisions. DO write the simulator as described above and your life will be a lot easier.

## Physics

Here are the rules for the physics of the situation.

- 
1. If two balls collide, the motion of their center of mass of the two balls after the collision is the same as the motion before the collision.
  2. The change in momentum of either ball in a two-ball collision must be along the axis defined by the centers of the two balls.
  3. In the center-of-mass frame of reference, each of the balls involved in the collision exactly reverses its velocity.

From these, you can derive your simulation rules. If you're not good at physics, talk to a TA, who can help you set up the equations.

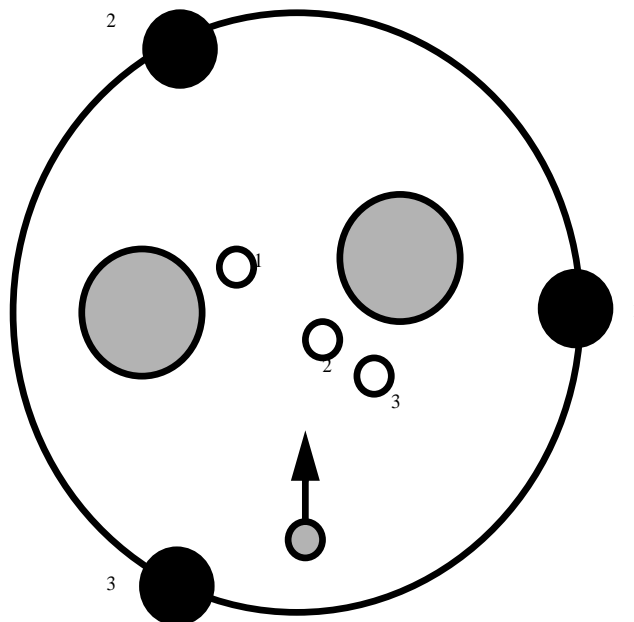
Note: you can handle an n-ball collision by treating it as a sequence of 2-ball collisions, where the order doesn't really matter --- use your priority queue to determine the order that you use.

### And the fun part...

Now take your simulator, determine some way to mutate from one animation to another, and set yourself some goals. (Don't just run a simulation and see what happens and declare that your goal...that's cheating!) Then use MCMC to try to achieve those goals in a physically realistic way.

You have to playback your top animations somehow. You can write a visualizer and share it with the rest of the class, keep it to yourself, or hope that somebody else writes a visualizer.

As an example, you might start with three balls on a table as shown, one cue-ball that arrives travelling from south to north (as shown) and have the balls end up in the numbered pockets after no more than 5 collisions. (This may be too hard; start with a simple goal and work up from there).



---

**Handin**

Hand in documentation of your work, and your working program, and some description of how to view the results, and anything you think would help us understand your work (code, Mathematica files, et cetera). You should use `/course/cs224/bin/cs224_handin anim`.

**File Structure**

For the sake of making testing easy, we recommend the following file structure to indicate the layout of a table and the initial velocities/position of the balls.

```
nbumpers // less than 6
x1 y1 r1
x2 y2 r2
...

xn yn rn
nballs
x1 y1 vx1 vy1
x2 y2 vx2 vy2
...
xk yk vxk vyk
```

where  $x_i$  and  $y_i$  are the positions of the ball or bumper centers,  $v_{xi}$  and  $v_{yi}$  are the initial velocities for the balls, and  $r_i$  is the bumper radius.

You may want to extend this format to include color of each ball and constraint information (e.g. ball  $n$  is red and should end up in pocket 1)