Assignment 5, Simulating Billiards

CMPT 102

03-1

1 The Problem

You will be simulating the movement of balls on a pool table. The physics involved here is fairly simple. Actually, it isn't really that simple [1], but a fairly simple subset of reality is quite close.

The problem is basically simulating inelastic collisions. A pool table is a flat rectangular surface that some pool balls can roll around on. When the balls get to any of the edges of the table, they bound off of a bumper. The balls also bounce off one another.

Our pool table is the special kind, with neither friction nor pockets. In fact, maybe it's an air hockey table.

2 The Physics

Understanding the physics that's going on here isn't the point of this assignment. But, if you don't know a little about the physics of a pool table, you can't simulate it.

We can assume that the collisions between balls are perfectly inelastic and that the rails return all of the momentum, but simply change it's direction. We will also assume that all of the balls have the same mass.

All of the physics happens in two dimensions. Vectors will be used to specify the position and velocity of the balls. For any vector $\vec{x}$, we will use x to represent the magnitude of the vector and x_x and x_y will be the two components of the vector.

2.1 Ball-Ball collisions

Collisions between two balls occur when the positions of their centres get within a diameter of each other. Calculating the velocities after the collision isn't easy. It will be handled by a library function, so you don't have to worry about it.

If you're interested in the details, see the instructor.

2.2 Ball-bank collisions

Let l and w be the length and width of the table. So each ball's x position should be between 0 and l and its y position should be between 0 and w.

When a ball bounces off of the bank, all of the momentum is returned. If a ball bounces off of one of the banks at $x = 0$ or $x = l$, its v_x should be set to $-v_x$.

If it bounces off the bank at $y = 0$ or $y = w$, its v_y should be set to $-v_y$.

3 The Simulation

Of course, understanding the physics isn't the point here, doing the simulation is. We will have to keep track of each ball on the table. We will store the position of the ball, $\vec{p} = (p_x, p_y)$, and its velocity, $\vec{v} = (v_x, v_y)$.

We will simulate the system in a series of discrete steps, each with some length, t seconds. With each step, each ball will have to change it's x position by $t \cdot v_x$ and the y position by $t \cdot v_y$.

With each step, we will have to check each ball to see if it has hit a bank or collided with another ball. If either of these has occurred, we will have to adjust the velocities as described in Section **??**.

Note that since we are simulating in steps like this, the ball might be overlapping the bank or the other ball when we notice that the collision has occurred. That's okay—we just want to look for other balls with their centres within a diameter, or a rail within a radius of the centre of our ball.

4 Program details

You should create a program called `pool.c` to implement this system. You should set a small time step, t. This will be the time difference between each

step of your simulation.

As mentioned above, with each step of the simulation, you have to (i) move the balls, (ii) check for collisions between two balls, and (iii) check for collisions between a ball and a bank.

To check for collisions between two balls, you have to check each *pair of* balls to see if their centres are within a diameter. Iterating over each pair of something is a common thing to do and is done something like this:

```
for ( i=0; i<NBALLS; i++ ) {
  for ( j=i+1; j<NBALLS; j++ ) {
```

Note that `j` starts at `i+1`, so each pair is only checked once, and `i` is never equal to `j`.

You are not required to do any input—all of the parameters and ball positions can be set in the code itself. You should make it easy to change these values if you want to explore another table setup, though. When you submit the program, it should have an interesting table-setup to demonstrate your program.

5 The library

In order to get you started and get you over some of the harder parts of the assignment, you should start with the `libpoolsim.h` library, provided by the instructor.

Before you `#include` this library, you should define the symbolic constants `LENGTH` and `WIDTH` which give the size of the table, in metres. You also should set `NBALLS`, which is the maximum number of balls to be considered and `BALLDIAM`, which is the diameter of a ball, in metres.

So, in your program, you should have something like this:

```
#define LENGTH 2.0
#define WIDTH 1.0
#define NBALLS 16
#define BALLDIAM 0.07
#include "/gfs1/CMPT/102/a5/libpoolsim.h"
```

This library contains the following type definitions:

```
typedef struct {
  double x;
  double y;
} vector;
typedef struct {
  vector pos;
  vector vel;
  short int ontable;
} ball;
```

The first type, `vector`, can be used to store any two-dimensional vector. The second, `ball` contains all of the information that we need to store about a single ball. The first two parts are `vector`s that contain the position and velocity of the ball, respectively. The last, `ontable` should be set to `1` if the ball is on the table and `0` if not.

The `ontable` part is there for two reasons. First, it allows us to only have a few balls on the table at a time, which will make testing easier. Second, it would allow us to easily add pockets to our simulation. If a ball falls into a pocket, we simply set its `ontable` to `0` and ignore it from then on. You don't have to worry about pockets for your assignment.

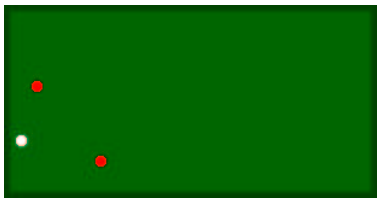
That means that *every* time you look at a ball, you should first check that it's `ontable` is `1`. If not, you should ignore it.

Also defined in the library is the function `draw_img` with the following prototype:

```
void draw_img( ball balls[NBALLS] , int generation );
```

This function creates an image file called `pool00000.ppm` in the current directory, where `00000` is replaced with the value `generation`. The array `balls` should contain the current information about each ball.

An image like this will be created:



The balls are drawn in red, except `balls[0]`, which is drawn in white. We must be playing snooker. If several of these images are appropriately com-

bined, they should form an animation of the ball movement. See Section **??** for information on working with the images.

So you don't have to worry about the particulars of the ball-ball collisions, a function `collision` is defined with this prototype:

```
void collision( ball balls[NBALLS], int i, int j );
```

This function should be called if ball `i` and `j` have collided. It will adjust their velocities in the array `balls`. So, if you detect a collision between two balls, you just have to call `collision` and it will take care of everything else.

6 Working with the images

Once you have your program working and creating images, you'll probably want to see what they look like.

6.1 In the lab

Viewing the images is far easier in the lab than from your own computer.

You can view a single image with the command `display pool00001.ppm`.

You can view all of the images, as frames of an animation with a command like this:

```
animate -delay 10 -pause 2 *.ppm
```

Here, the number after `-delay` is the 1/100-ths of a second to pause between frames and and number after `-pause` is the number of seconds to wait before repeating the animation.

If you want to convert the images into an animated GIF file, which can be viewed with a web browser and put on a web page, you can use this command:

```
convert -delay 10 -loop 0 *.ppm poolanim.gif
```

This will create a file called `poolanim.gif` containing the animation. The number after `-delay` is the 1/100-ths of a second to pause between frames. The `-loop 0` part indicates that the animation should loop after finishing.

6.2 At home

This is going to be a pain. You might be better off to get the program working and then go to the lab to play with the images.

You can transfer the frames of the image to your home computer with FTP and view them in your favourite image viewer. If you want to convert them to another format before transferring them, you can try a command like one of these, depending on the image type you want:

```
convert pool00023.ppm pool00023.bmp
convert pool00023.ppm pool00023.gif
convert pool00023.ppm pool00023.png
```

You can also create and animated GIF as described above and transfer that.

7 Hints

Your `main()` function could get quite large. You might consider breaking it up and creating a couple of functions so it is easier to understand.

You'll probably need to access your array of ball information from each of these functions. Since it is a fairly large data structure, you might want to make it a global variable instead of passing it around as a parameter.

- Remember to initialize the values in all of the variables that need it. If you create an array that needs to be initialized, you'll have to write a `for` loop to do that.
- Don't forget to check that each ball is on the table *before* you do anything with it.
- You might want to define a function to compute the distance between two vectors, since you'll be needing it in a few places:

  ```
  double distance(vector a, vector b)
  ```

- Suggested plan of attack:

  1. Create the array to hold the ball information and initialize it. Make sure you're comfortable with this much.

2. Start the simulation: get the balls moving, but don't worry if they collide with each other or the banks.
3. Get the image output working. The `draw_image` function will complain if the balls move off of the table, so it might not work perfectly until the next step.
4. Get the balls bouncing off of the banks correctly.
5. Get the balls bouncing off each other correctly.

8 References

1. Shepard, Ron, **Amateur Physics for the Amateur Pool Player**, third edition, `http://www.playpool.com/apapp/index.html`, 1997.
2. Any average first-year physics text.

9 Submitting

You have to use the submission server to submit your work. You only need to submit the file `pool.c`. You can do this by typing these commands:

```
tar cvf a5.tar pool.c
gzip a5.tar
```

Then, submit the file `a5.tar.gz`.