CSCI 441 - Lab 11
Friday, November 10, 2023
LAB IS DUE BY **FRIDAY NOVEMBER 17 11:59 PM**!!

Today, we'll look at **simple** collision detection within a bounding box and other particles in our system. This lab will fulfill the requirements for collision detection in A5. It may not be the most efficient, but it suits our needs. Feel free, if you wish, to explore other data structures to improve performance. But it is not required.

Please answer the questions as you go inside your README.txt file.

Step 0 – Copy New Library Files

If you look inside the include/ folder, you will see a new CSCI441/ file listed. There is a TextureUtils helper file. Copy it to your include/CSCI441/ files.

Step 1 – Watching Them Go

Compile and then run the lab. Enter a number of marbles to create. Everything will start. You should see a ground plane, and a bunch of colored brick spheres just sitting there. It'd be much nicer if they were moving.

Take a look at the `Marble` class. Each marble has a location and a direction. The `moveForward()` method moves and rotates the ball one step along the direction vector, updating the location in the process. `moveBackward()` takes a step the opposite direction.

These functions will take care of all the magic. Every time a frame is rendered, this represents one time step in our world. Every time step, we want to move every marble forward along their direction. Find `TODO #1`, you need to loop through every marble in scene and have them move forward.

Compile and run.

Q1: How do they roll?

And now they're rolling around. Right now, the spheres will continue to roll in the direction they are initially set to forever (forever. for ever. four. ehv. er.)

Alright cool, let's get them bouncing.

Step 2 – Keep Them Grounded

Our first collision test will be to check for intersections with the bounding box of our ground plane. The ground plane spans the space between two corner points:

```
(-_groundSize, 0, -_groundSize)
( _groundSize, 0,  _groundSize)
```

The components of those points also correspond to the limits along each dimension. Remember this plane is drawn in the XZ-plane. Goto `TODO #2`.

For every marble, we need to do four checks. One check corresponds to if the marble passes beyond the limit of each dimension or hitting a wall of our bounding box.

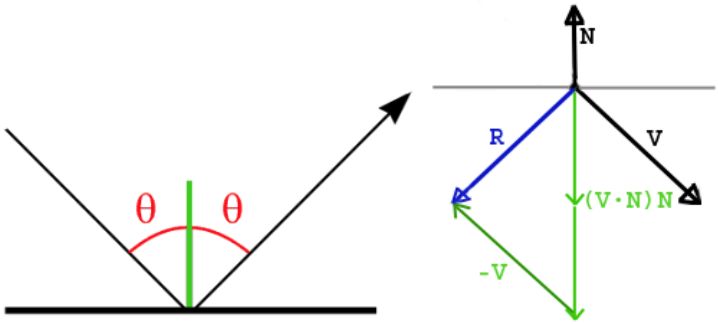
(Hint: `if( marble->getLocation().x > _groundSize ) { … })`

If the ball goes beyond the limits, then a collision has occurred, and we must handle the bounce.

Two things will happen in a bounce. First, since we already moved the marble forward to go into a collision, we must move the ball backwards to undo the collision. (*Aside: we can handle this differently by storing our moveTo point and testing the moveTo point. If the moveTo point results in no collisions, then we'll assign it as the new point to our marble.*)

Next, we need to have our marble bounce, which corresponds to changing its direction. Luckily, our brick spheres mimic the movement of billiard balls and result in perfectly elastic collisions. Hooray! This means the speed/momentum/energy going into a collision equals the speed/momentum/energy coming out of the collision. So we only need to update our direction vector.

To update our direction vector, we need to compute the new reflected vector after the collision. To handle any arbitrary wall, the angle of incidence will equal the angle of reflection respective to the wall's normal. This link (http://www.3dkingdoms.com/weekly/weekly.php?a=2) gives a good description of the math. The images below demonstrate this wall bouncing behavior.



The formula to use is:

$$Vector_{Out} = Vector_{In} - 2 * dot(Vector_{In}, Normal) * Normal;$$

Now we just need to know the normal for each of our walls in our bounding box. In the case of our bounding box, the normal points inwards towards the center (origin) of our ground plane. Don't forget we want normalized normals.

(Hint: The normal for the wall at `(_groundSize,0,z)` is `(-1, 0, 0)`)

It may be wise to add in a single collision test with one wall, compile, and run. Then when that wall is working, add a second test. And so on.

Q2: Do the marbles now stay contained within the ground plane? What four normals did you use?

Alright, the marbles bounce around the scene but go through each other. One more collision test to add in.

Step 3 – Let's Play Marbles

Lastly we need to check if after moving all of our marbles, if we now have moved any two (or potentially more) marbles into each other. TODO #3!

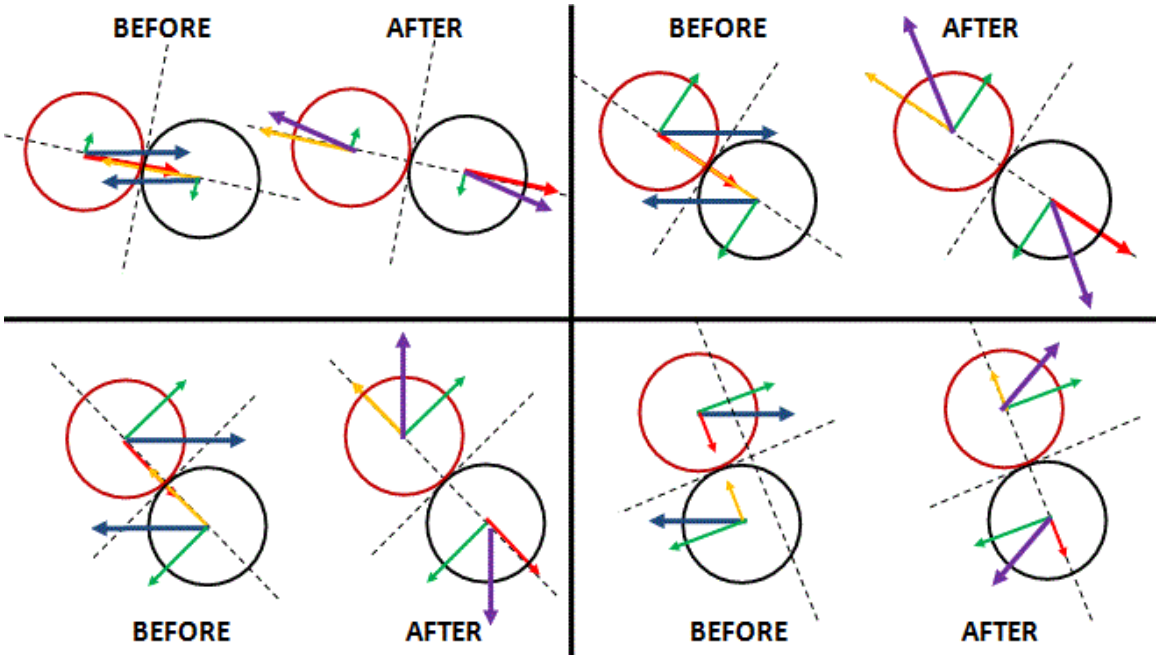
We already discussed the way to check if two spheres intersect is to test if the distance between their centers is less than the sum of their radii. Note that the spheres in our world are different sizes.

For every ball in our scene, we will compare it against every other ball in our scene.

(Hint 1: Nested for loops.)
(Hint 2: Make sure you don't compare the same marble against itself).

Now for any two marbles, we must compute the distance between them and compare that against their radii. If a collision has occurred, we now need to have both marbles bounce off each other. Like our walls, the first thing we will do is move each marble backwards slightly to unstick them.

The next image has a few examples of some of these arbitrary collisions.



The dotted lines are clues to how we can do this arbitrary collision. When two spheres intersect, their collision occurs in the plane of their movement. It is also like they are hitting an invisible wall between the two spheres. We can define any arbitrary wall, or plane, by a point and a normal. Both of these are easy to compute for two spheres.

The point, or collision point, is where the two spheres meet. This is along the line connecting their centers and occurs on the edge of the sphere. As it turns out, we don't really need to know this point. We just need the normal of the plane to reflect off of.

Hopefully it is clear from the images, that the normal for the plane is the vector from the collision point to the center of each respective ball. We will simulate two planes, each with a normal facing one of the balls.

In the images above, the red ball will be reflected around a normal pointing towards the red ball. The black ball will reflect around a normal pointing towards the black ball.

Q3: Knowing just the locations of the two spheres, how can we compute the two normals? If we have marble point L_1 and marble point L_2, what is the equation to give us our two normals N_1 and N_2?

Now that we have the two normals to reflect off of, we can use the same equation we used above to compute each sphere's new direction vector. Go ahead and update their directions.

Lastly, we want to keep our animation smooth. We had initially moved our balls backwards, so this frame they did not move. Let's move each ball forward along its new direction vector to simulate the bounce.

Compile and run.

Q4: Success?

And there you have it! We can modify our bouncing to represent particles of different masses and/or have inelastic collisions. But this is the starting point and all you need for A5 coming up. In this lab, every frame the direction vectors remained constant and the balls moved in a line (unless they collide). For A5, you will need to update the direction vector of each particle so they will begin moving in curved paths towards your Hero. You then need to check for collisions against the bounding box, other particles, and the Hero. IFF you choose to add other static objects into your scene, you will need to add collision detection against those objects. But this lab will give you the minimum required collision detection needed for A5.

If you want to read about more types of intersection tests, this paper (http://wscg.zcu.cz/wscg2004/Papers_2004_Full/B83.pdf) goes through testing arbitrary 2D polygons. This link (https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection) discusses axis-aligned bounding boxes (AABB).

Q5: Was this lab fun? 1-10 (1 least fun, 10 most fun)
Q6: How was the write-up for the lab? Too much hand holding? Too thorough? Too vague? Just right?
Q7: How long did this lab take you?
Q8: Any other comments?

To submit this lab, zip together your source code, screenshot, and README.txt with questions. Name the zip file <HeroName>_L11.zip. Upload this on to Canvas under the L11 section.

LAB IS DUE BY **FRIDAY NOVEMBER 17 11:59 PM**!!