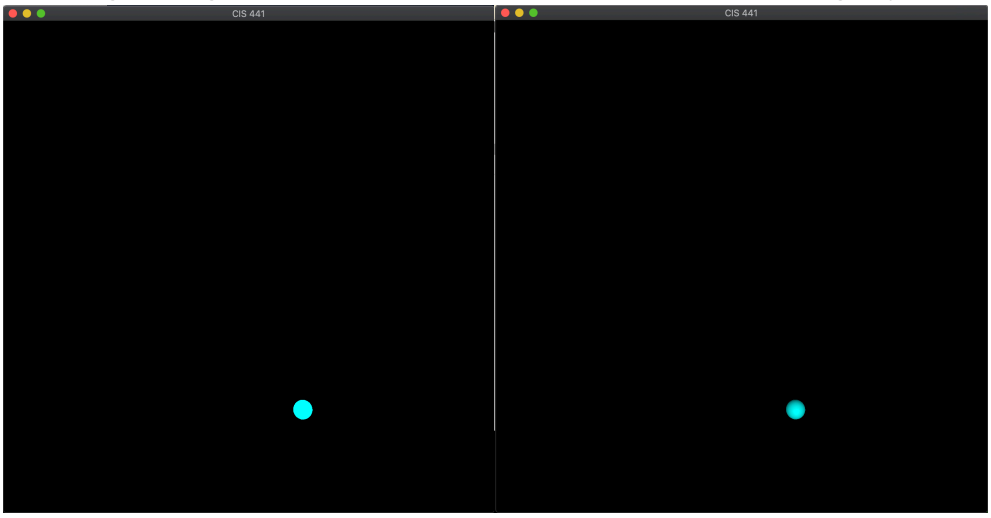
CIS 441/541: Project #3B

Goal: Extend the starter to create realistic collision responses among spheres in 3D space. In addition to these collisions, you will vary the following attributes of the spheres in response to collisions and ticks:

(1) Color
(2) Size

Steps:

1. Modify your CMakeLists.txt to work with the starter code.
2. Compile and run the code. You should see a single, unlit sphere (left). Now copy over your code for Phong shading into the vertex shader. You should now see the single sphere as lit (right).

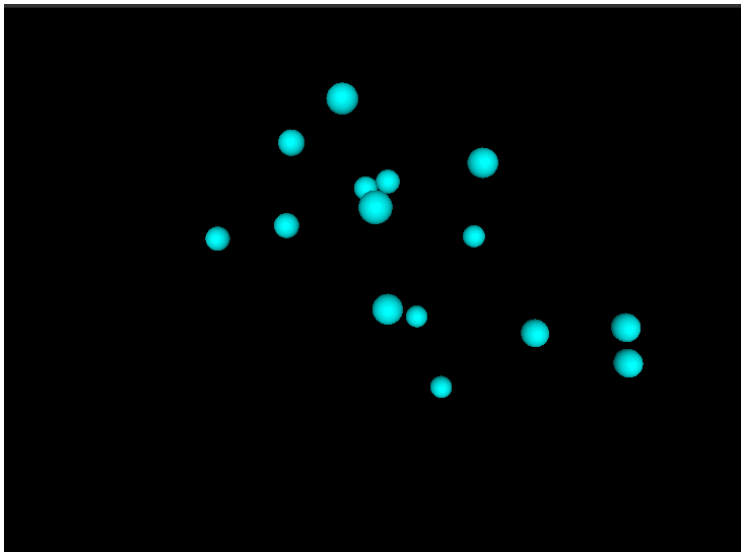


3. We're going to start by making the sphere move. The sphere has several attributes you should take note of – position, direction, color, radius, and time since collision. We can use the direction attribute to update the position each tick. Then by drawing the sphere at the new location, we can create movement. Uncomment the code labeled "STEP 3". This is basic code to keep the ball inside of a general box by "bouncing" it off the walls. We'll come back to it later but for now we will move on. Compile and run the code, and you should see your ball bounce off a wall or two.
4. Now let's get to implementing the collision. You should notice the main code already performs a rudimentary collision detection – each sphere checks every other sphere to see if a collision has happened. This is fine for our purposes, but right now it doesn't do anything.
   To properly detect the collision, update the LookForCollision and getDistance functions so we can tell if two spheres are overlapping. (HINT: distance is measured from the center of one sphere to the center of the other. What is the smallest the distance can be before they touch?)
5. Update the color of the spheres when they collide (Use whatever colors you want). If you add more spheres (edit "numSprites" in the main function) you should notice they pass through each other, but change colors when touching.
6. Now that the collision detection is working, we need to implement the collision. Call the function collideDir when a collision is detected instead of changing colors. This function should

take the position and direction of both spheres involved in the collision, and update them. To do so, we will need to implement that function.

7. Implement collideDir. This involves the following steps:
    a. Calculate the normalized normal vector N between the two balls
    b. Calculate the relative velocity vector between the two balls
    c. Calculate the relative speed along the normal vector
    d. Modify both ball's direction by the relative speed

    This is an elastic collision between two circles in 2D space generalized to spheres in 3D. For a more in-depth explanation, see the comments in the function
8. Compile this code with fifteen balls and run. You should be able to observe some bounces between the balls. Since that can be difficult to see, after 1000 ticks you should see the following:



9. Now we are going to vary color and size with collisions. After each collision, modify color according to the following formula:

```
color[0] += 0.7 * (1-color[0])
color[1] -= 0.1 * color[1]
color[2] -= 0.1 * color[2]
```

    Additionally, each 50 ticks without a collision, modify the color as follows:

```
color[0] -= 0.05*color[0]
color[1] += 0.35*(1-color[1])
color[2] += 0.35*(1-color[2])
```

    Compile and run the code. You should observe the color of the balls changing as they collide.

10. Finally, modify the size of the spheres in a similar manner. Every collision should reduce the radius of the sphere by the following formula:

```
radius -= 0.25 * radius;
```

    Additionally, each 50 ticks without a collision, modify the radius as follows:

radius += 0.01 * (1-radius)

Compile and run the code with 50 balls. After 1000 ticks with 50 balls, you should see the following: