Andrew Seba

Math 11573 10416 – 201601

1/31/16

## Introduction:

For most games there will be walls to move around obstacles to avoid to make the game challenging. To let the player know they are colliding with an object we must keep track of bounding boxes or spheres. Line intersections can be used to get exact intersections of collisions on objects if a greater precision is needed.

## Methods:

### Spheres:

The collisions between two spheres mathematically takes two steps. Find the distance between the two spheres and checking that distance against their combined radius. First describing a sphere is defined as

Where is the center of the sphere; and r is the radius.

Then we can find the distance between the two sphere’s centers by the following based from Pythagoras:

Then if the distance is smaller than the combined radius from both spheres the spheres will be overlapping.

For the programming side just get the first center in x and y coordinates then the radius, and then get thee second, the resulting output should show after receiving the input.

### Axially-Aligned Bounding Boxes:

The collisions between two Axially-Aligned Bounding Boxes takes a couple more steps. For each dimension the bounding box is in we need to check the distances between each axis’s center point against their combined axis radius. First we can describe an AABB (Axially-Aligned Bounding Box) by two points from opposite corners such as (0,0,0) and (1,1,1) this will be a box 1 unit in height, length, and width. To find the distance between to AABB’s we must first find the center point of each axis, to do that just get the average value for the axis we are checking the distance for. Where is the minimum value and is the max value on the same axis.

And then we can get the radius by getting the difference of the midpoint and the max value.

Now we can use the same distance formula to find out if two different AABB’s are overlapping.

Then we can check to see if the AABB’s are overlapping in the axis we found the radius and distance for.

For the application to the computer the program would just need the user’s inputted values for the extreme corners of the boxes in x, y, and z coordinates ( z if your doing 3d). Then after the given input the steps needed would to be just check the overlap in each axis that the box has. Then the output will display.

### Line Segments:

The intersection between two lines returns a point. We can describe two lines with

To find intersection coordinates of a line we can use Cramer’s Rule

There are some cases where this can lead to a divide by 0 case. In this case the lines are parallel and there will be no intersection or infinitely many points. However for this lab we are doing line segments, so lines might be perpendicular and never touch. For this lab we are given points and the line is defined between the start and end point of the line. So after we have our line equation and intersection point we can check to see if the x and y lie on the segment using:

If so then the two lines intersect.

## Results:

### Spheres:

Testing out some given test examples for two spheres we can use their center coordinates and radii by simply comparing the distance between the two centers and the sum of the radii we can see if they overlap.

Centers: (x,y,z), Radii: (m)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Sphere 1 Center | Sphere 1 Radius | Sphere 2 Center | Sphere 2 Radius | Results |
| Ex.1 | ( 2, 7, 6 ) | 4 | ( 8 , 1, 2 ) | 5 | No Overlap |
| Ex.2 | ( -6, 3, 2 ) | 10 | ( 1, -1, 6 ) | 1 | Overlap |
| Ex.3 | ( 4, 5, 7 ) | 1 | ( 3, 6, 2 ) | 5 | Overlap |

### AABB:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Box1-Point1 | Box1-Point2 | Box2-Point1 | Box2-Point2 | Results |
| Ex.1 | ( 2, 5 ) | ( 4, 7 ) | ( -3, 8 ) | ( 5, 2 ) | Overlap |
| Ex.2 | ( 3, 10 ) | ( 1, 4 ) | ( 5, 1 ) | ( 10, 3 ) | No Overlap |
| Ex.3 | ( 3, 7, 8 ) | ( 9, 2, 10 ) | ( 6, 4, 5 ) | ( 7, 8 ,6 ) | No Overlap |
| Ex.4 | ( 2, 4, 5 ) | ( 3, 6, 7 ) | ( 3, 5, 2 ) | ( 6, 6, 5 ) | No Overlap |

### Line Segments:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Line1, Point1 | Line1, Point2 | Line2, Point1 | Line2, Point2 | Results |
| Ex.1 | ( 3, 2 ) | ( 3, 9 ) | ( -1, 6 ) | ( 8, 6 ) | Intersect at (3, 6) |
| Ex.2 | ( -2, -5 ) | ( 6, 3 ) | ( 5, -5 ) | ( 5, 1 ) | No intersection between segments. |
| Ex.3 | ( -4, 4 ) | ( 4, -4 ) | ( -8, -2 ) | ( 10, 1 ) | Intersection at (0.57, -0.57) |
| Ex.4 | ( 2, 9 ) | ( 4, 6 ) | ( 3, 2 ) | ( 5, 5 ) | No intersection between segments. |

## Conclusions:

Sense these output just Booleans checking these values don’t take much time at all, the results are always solid in these cases. If we needed to start moving them around on screen some prediction models would be needed.

## Post lab Questions:

1. What adjustments would you have to make to your program that detected overlap in spheres so it could handle circles in 2D?

Just assign the point’s z value to 0.

1. Would it be better to use AABB’s or spheres in collision detection? Why?

It just depends on what you’re trying to do, For little more accuracy around oddly shaped objects the AABB’s would be more precise however the circles takes less computing. For a general purpose I would use AABB’s, then for moving objects that don’t need a lot of accuracy I would just use a sphere collider.