**Collision Detection by John Ying**

Introduction:

In games, one of the most important physics aspects in today’s society is collision detection. The ability for the game to know when one object runs into another to stop it from “going through” it. If a car runs into a tree, the car should stop due to colliding with it. With this program, basic algorithms will allow the user to detect if objects collide together, or aren’t even touching.

Methods:

After determining the beginning variables with which to work, like the x and y coordinates of a point, radius, etc. through user input, basic math computations are used. Beginning with the sphere, the equation used is as follows:

Radius Distance = Radius 1 + Radius 2

This determines the distance needing to be overcome if the sphere’s collide, or if the actual distance of the spheres is less than the radius distance, then they do not. To determine the actual distance of the two sphere’s, this equation is used when the points (0, 1) radius 4 (3,4) radius 3

Distance = SquareRoot ( (3 – 0)^2 + (4-1)^2) )

If the distance is greater than the radius distance, then the two spheres do not touch, otherwise they do.

In solving an axially aligned bounding box equation, variables for the points of box 1 and box 2 must be given. Likewise all points radius MUST be greater than or equal to the distance between the two points of the same point coordinate position.

In Box 1 (8, -3, 0), (20, 5, 7) and Box 2 (0,0,4),(10,1,10) are given. By solving for the center and radius of each one:

Center = (20 + 8) / 2

Radius = 20 – center

Solving for each x, y, and z coordinates, the actual distance is subtracted as such:

Distance = Center1 – Center2

Distance <= Radius1 + Radius2

The only rule is that Center1 must be the larger of the two numbers.

In determining the line connections, Cramers Rule is key. By turning the points into Standard Form first:

Ans1 = (( x2 – x1 )\*y1) – (( y2 – y1 )\*x1);

xValue1 = y1 – y2

yValue1 = x2 – x1

Repeat for the second line.

Then solve for the solutions.

xSol = ( ((yValue1)\*(ans2)) – ((ans1)\*yValue2)) ) / ( ((yValue1)\*(xValue2)) – ((xValue1)\*(yValue2)) )

Conclusion:

In conclusion, this solves the equations for collision detection between lines (pathing), spheres (collision), and boxes (collision).