Collision Detection

1. Introduction

This tutorial teaches the basics of how to implement Collison Detection in your engine.

2. Sphere/Sphere Collision Detection

One of the simplest forms of collision detection is a sphere/sphere collision detection algorithm.

You select a sphere that surrounds each object and then simply, you measure to see if two spheres have intersected. To measure an intersection, you simply calculate the distance between their two centres and compare it to the radii added together.

In this example (fig 1) the two spheres are not intersecting because their distance between their two centres is larger than the two radii added together.

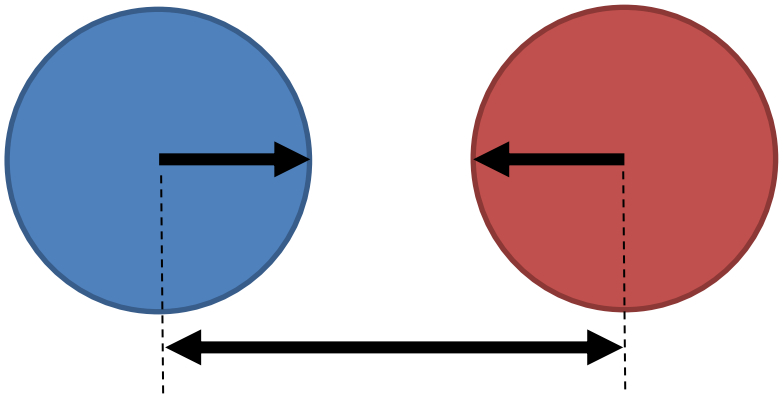


Fig 1 – the two spheres are not intersecting

In this example (fig 2) the two spheres are intersecting because their distance between their two centres is less than the two radii added together.

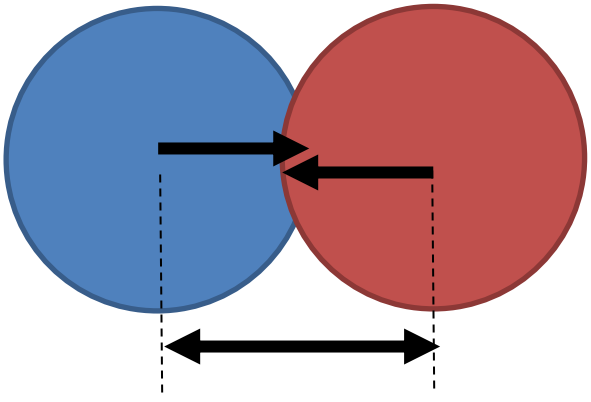


Fig 2 – the two spheres are intersecting

The sphere could be represented by a ComponentCollisionSphere and you could have a SystemCollisionSphereSphere to calculate the collisions between two ComponentCollisionSphere.

The ComponentCollisionSphere would need to store a radius. There already exists a ComponentPosition so we do not need to add that.

If you have an Entity that you want to add sphere collisions to, simply add a ComponentCollisionSphere to that Entity. Providing your Entity has a ComponentPosition then you have everything a SystemCollisionSphereSphere would need to calculate if there is a collision.

The SystemCollisionSphereSphere would need to take every two Entities and test for collisions. Therefore, you need to make sure that the SystemManager passes the list of Entities to each System instead of sending individual Entities to them as it does in the provided base code.

3. Point/Axis Aligned Bounding Box Collision Detection

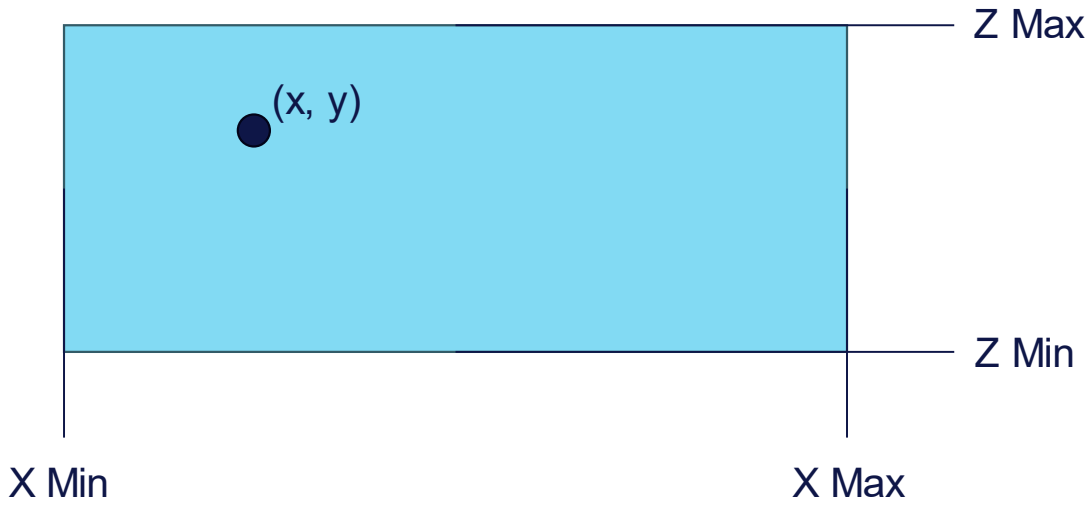
If we have an axis aligned bounding box, then we can do a simple test to see if the point is inside the box. For a point to be inside the box, the point must be within the bounds of the box.

The box could be represented by a ComponentCollisionAABB and you could have a SystemCollisionPointInAABB to calculate if a point is inside the box.

The ComponentCollisionAABB would store the bounds of the box.

If you have an Entity that you want to add AABB collisions to, simply add a ComponentCollisionAABB to that Entity.

The SystemCollisionPointInAABB would need to take a point and every Entity and test for collisions.



4. Line/Line Collision Detection

We can use vector mathematics to test whether two line segments cross each other. This can represent a wall section and the movement of the player and/or camera.

In this example (fig 3) we have a wall section (represented by a black line) and the player moving from below the wall to above the wall (represented by the green line). The wall section is a simple line since our walls are all vertical and that for the calculation we are looking directly down on the world, therefore the problem is only 2D.

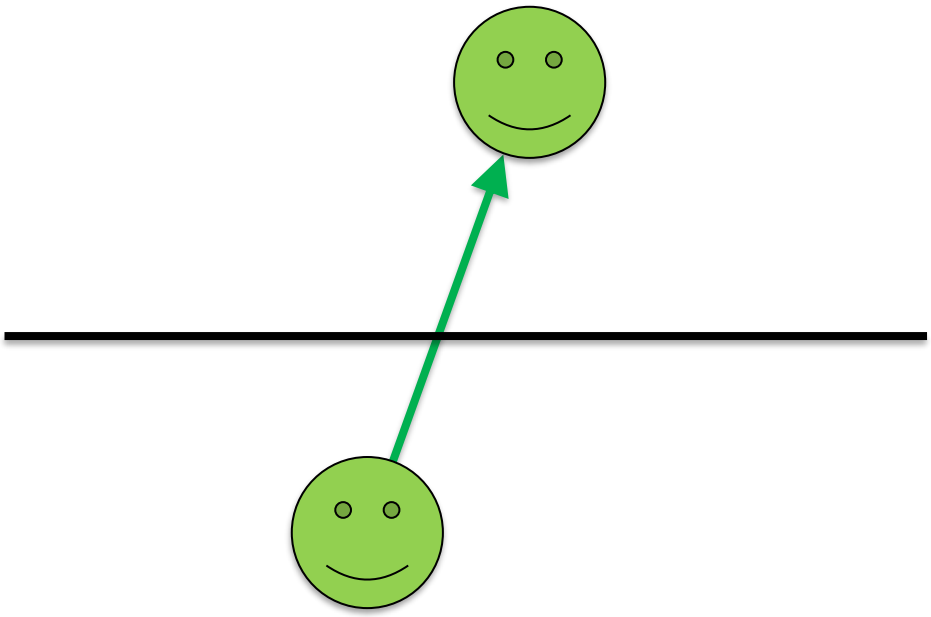


Fig 3 – player moving across a wall section

We can clearly see that the player is trying to move through the wall. However, we need an algorithm to test for this.

Part 1: The first part of the algorithm is to test if the player is moving from one side of the infinite line that the wall section makes to the other side of the infinite line (fig 4).

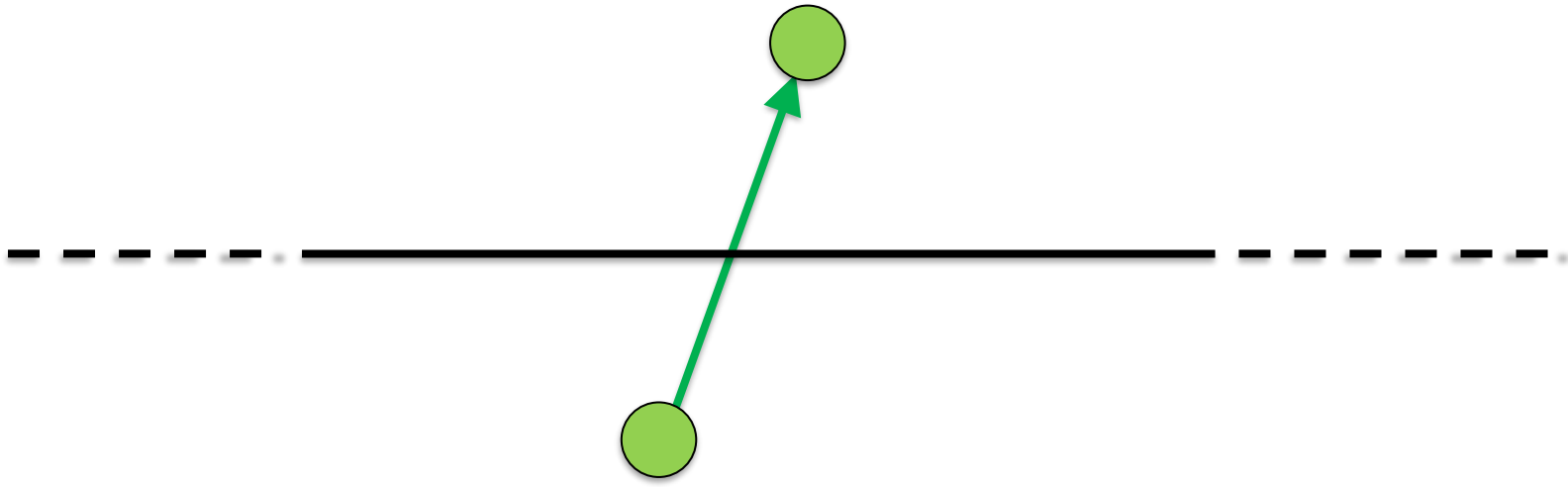


Fig 4 – player moving across an infinite line made from the wall section

To test for this movement across an infinite line we compare dot products from vectors (shown in red in fig 5) created from the player positions from the one point on the wall, using the normal from the wall using the equations in fig 5. It does not matter which point on the wall you use if you use the same point for creating the vectors.

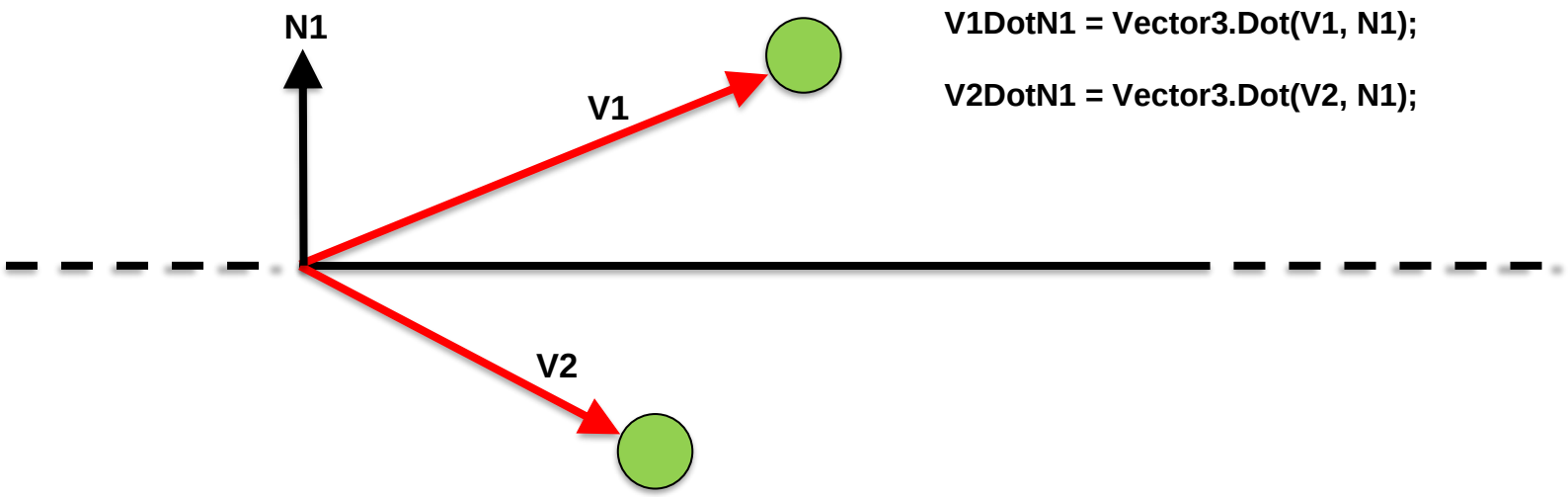


Fig 5 – vectors created from point on wall to player positions

For a possible collision, the product of V1DotN1 and V2DotN1 (i.e. V1DotN1 times V2DotN1) must be negative, i.e.:

V1DotN1 * V2DotN1 < 0

If the product of the two dot products is not negative, then there is no collision and we can stop the algorithm.

If the product of the two dot products is negative then there may be a collision so we need to continue the algorithm - what we have done is test that the player is moving across the infinite wall line, but we need to test to see if they are actually moving over the wall segment.

Part 2: The second part of the algorithm is to test if each side of the wall section are on other side of the infinite line that the player is moving through (fig 6).

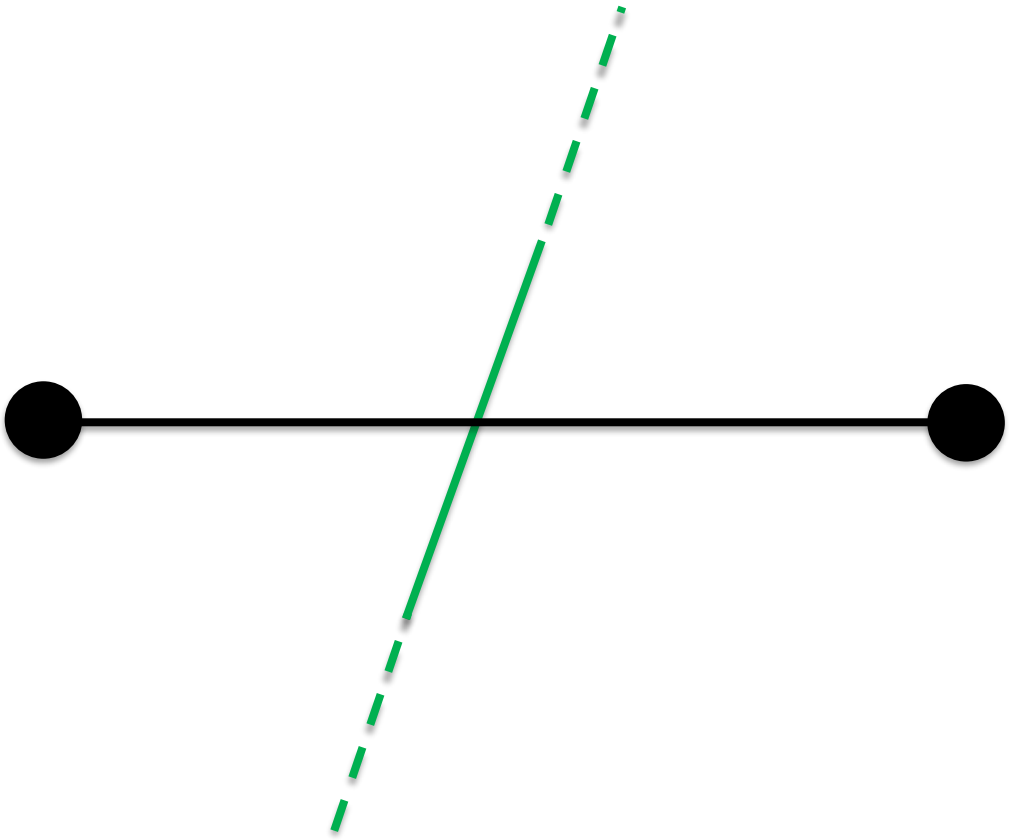


Fig 6 – player moving across an infinite line made from the wall section

To test for this we compare dot products from vectors (shown in red in fig 7) created from the wall end positions from the one of the player's positions, using the normal from the player's motion using the equations in fig 7.

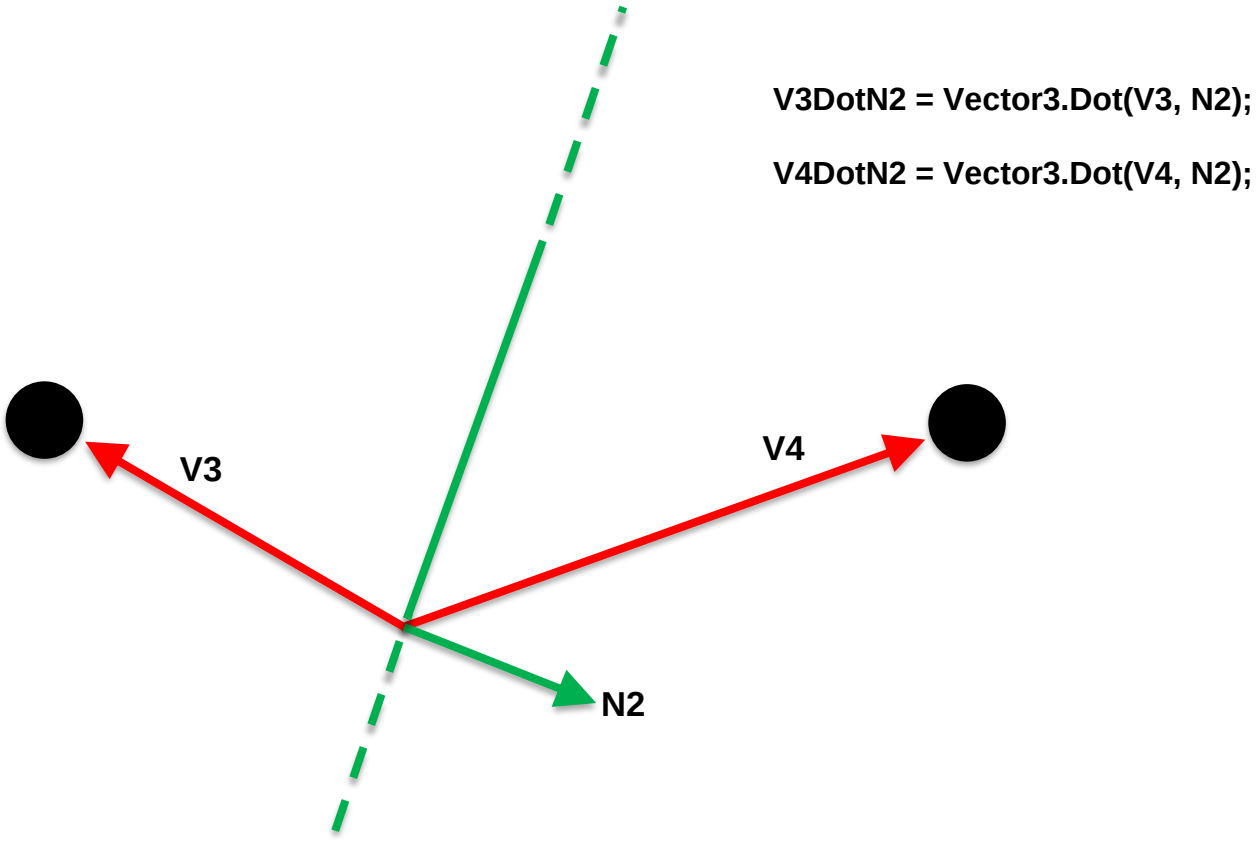


Fig 7 – vectors created from player's position to wall end positions

For a collision, the product of V3DotN2 and V4DotN2 (i.e. V3DotN2 times V4DotN2) must be negative, i.e.:

V3DotN2 * V4DotN2 < 0

If the product of the two dot products is not negative, then there is no collision and we can stop the algorithm.

If the product of the two dot products is negative, then there is a collision.

5. Exercises

Please attempt these exercises, but if you get stuck or you are confused then ask for help during the scheduled lab times.

1. Look at the sphere/sphere collision detection routine in the 600098-Inheritance_Basic_Example.zip that is provided in lab 1 on Canvas.

   There is a collision when the distance between the spheres is less than the sum of the radii of the spheres.

   Consider how you would add this functionality to your ECS architecture.

   Look back through the lecture video/slides and your lecture notes to implement collision detection between the Camera and any Entity that have an appropriate Sphere Collision Component.

2. Consider how you can extend your collision code to allow for any two Entity/Entity Sphere/Sphere collisions.

3. Consider how you can extend your collision code to allow for point in AABB collisions.