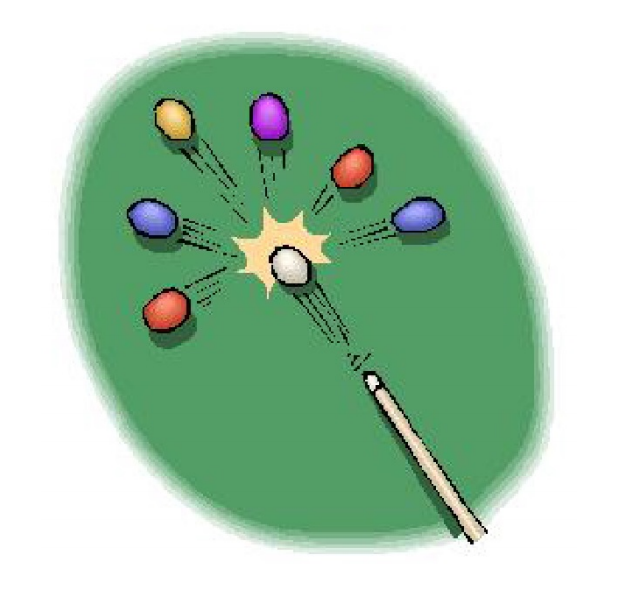
Physics Tutorial 7 – Separating Axis Theorem (Part 2)



**Summary**

In this section we will be extending our knowledge of SAT in regards to 3D space. We will be covering the edge-edge case along with discussing in detail how curves can be represented.

**New Concepts**

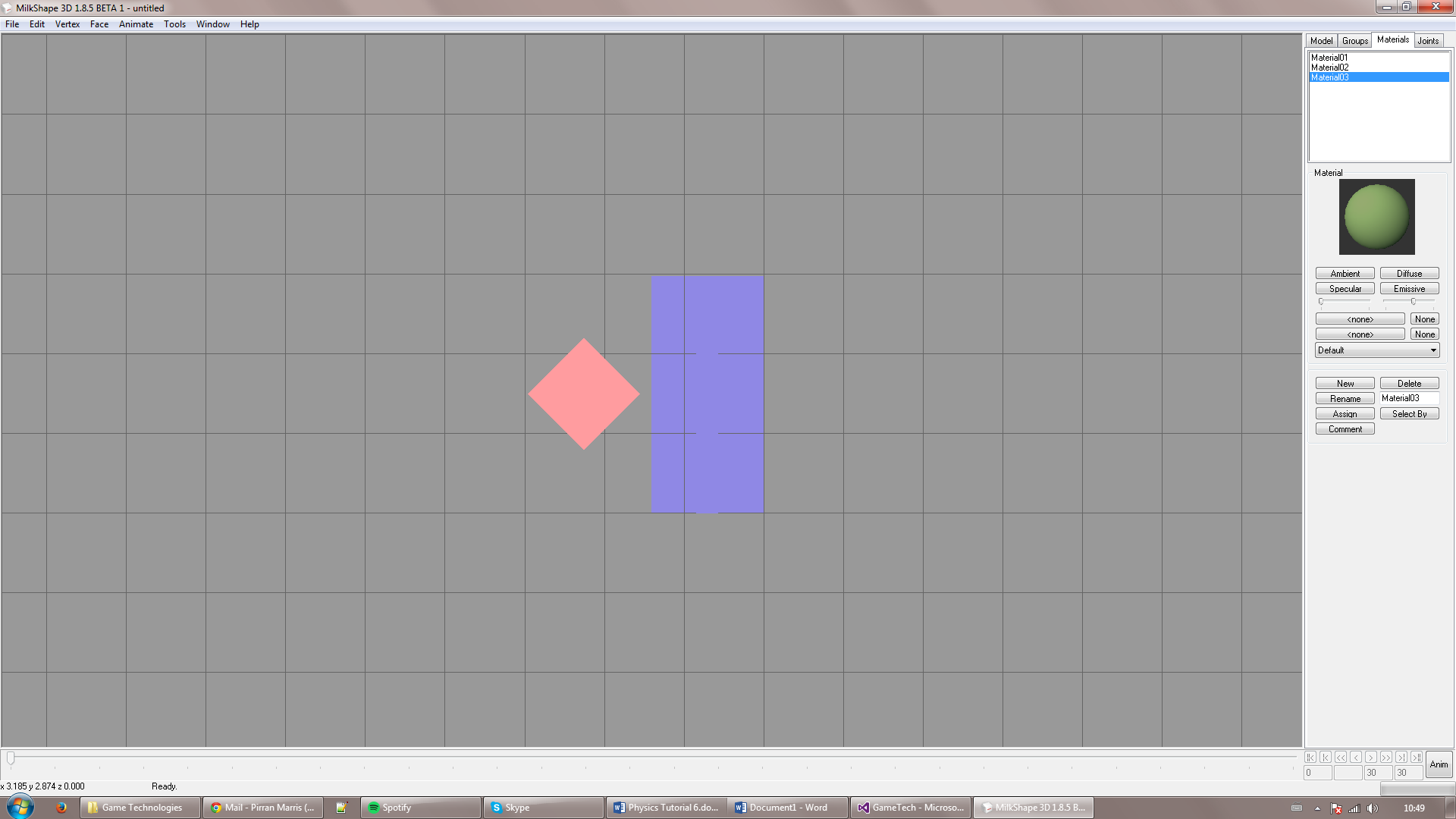
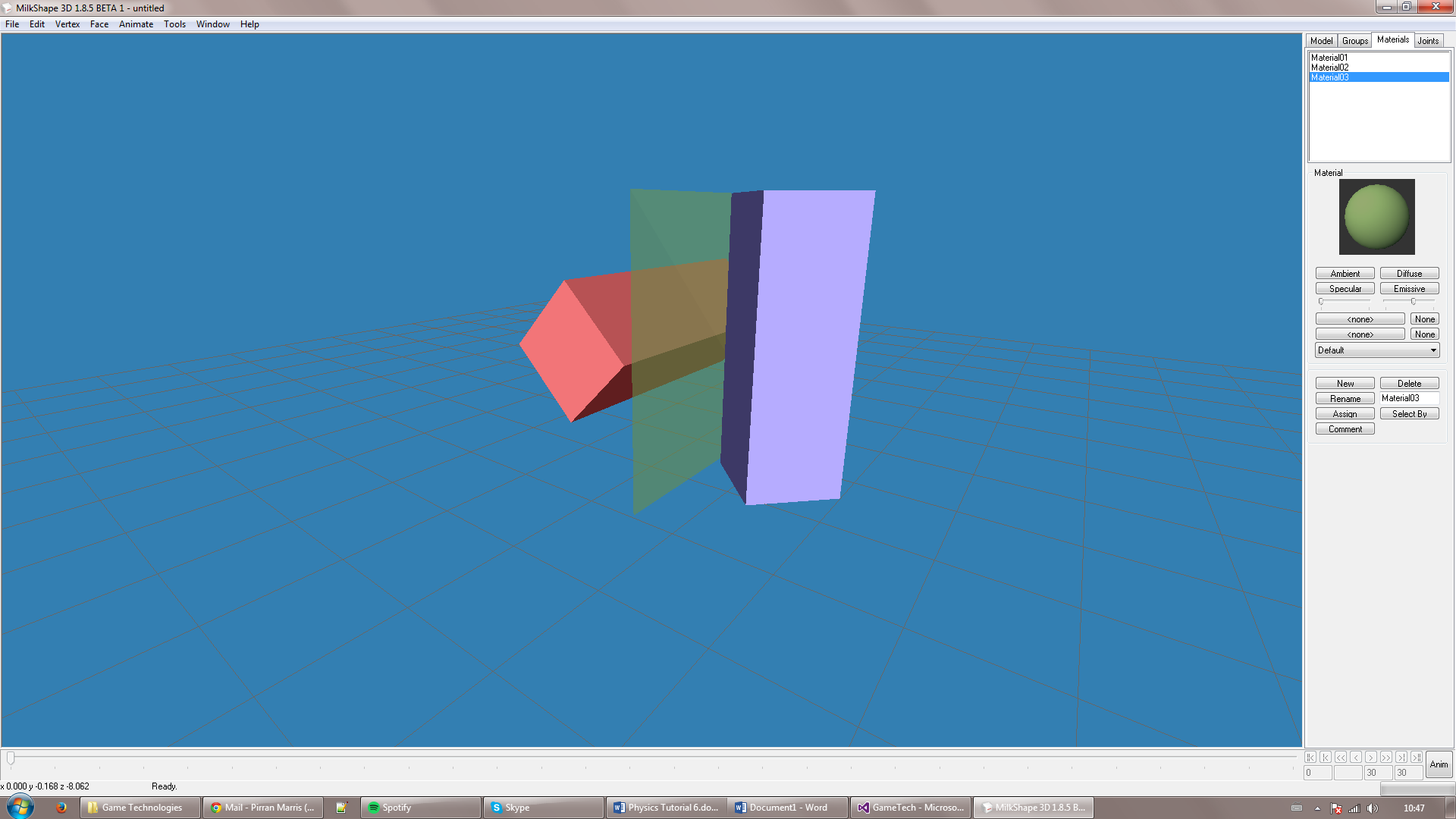
Separating Axis Theorem (3D), edge-edge cases, curve-curve cases, curve-edge cases

**Introduction**

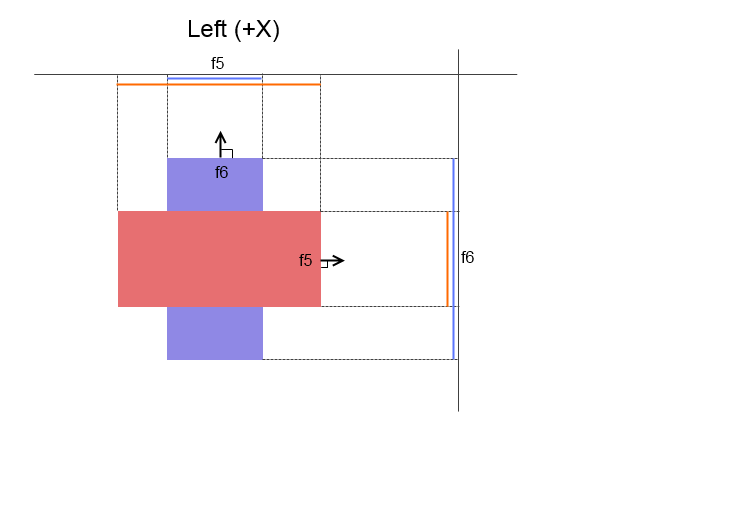
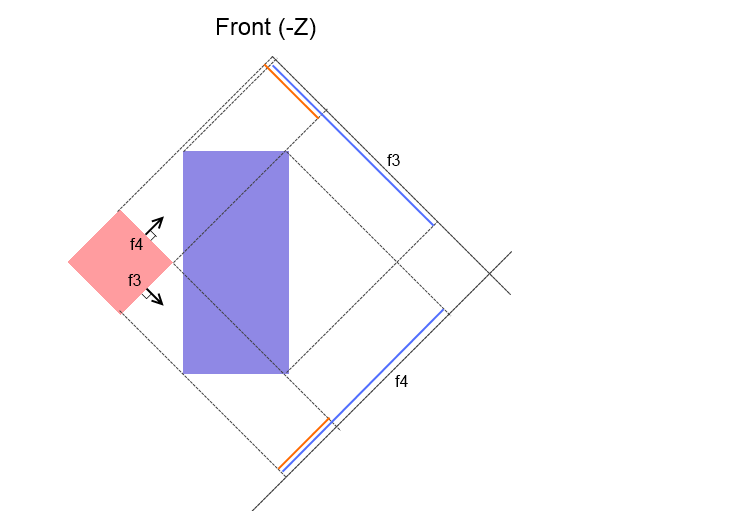
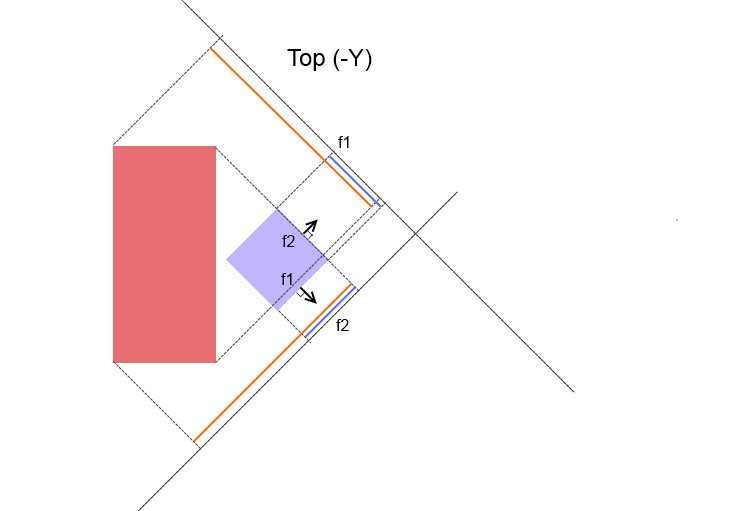
In the previous tutorial we covered the Separating Axis Theorem, focusing primarily on the 2D cases. We then extended that directly to the 3D case, and should now be able to identify when two objects are overlapping. As you may have discovered, there are certain cases where additional axes are required to be checked in the 3D case. Today we will be covering these cases, along with properly accounting for sphere-polygon and sphere-sphere cases.

**Edge-Edge Collisions**

In 2D, edges can be considered the same as faces, however in 3D this is an incorrect assumption. Thus we now have to compensate for edge-edge collisions as well. An example of a 3D case where purely using the normal of each face does not provide the correct result can be visualised below.



The two cuboids (Red and Blue) are not colliding, as we know that a plane exists in which they do not collide. If work through our current version of SAT then all possible separating axes return true. This is can be shown progressively below.



This means that were going to have to produce some new axes to check more possible planes between the two 3D objects.

The simplest way, and the way we cover in these tutorials, is to take every edge of both objects and use a cross product of each permutation to produce additional axes. This list will cover all possible edge-edge cases, including the green plane shown in the above figure.

If you recall from the graphics tutorials, the cross product between two non-parrallel vectors will result in a vector that is orthogonal to both of the previous vectors. So in simplest example, the x/y/z axes, if we cross x,y we will get z, if we cross y/z we will get x etc. So by using the cross product of an edge on object 1 and an edge on object 2 we will produce a possible axis that is orthogonal to both of those edges and a possible plane in which the two objects do not overlap.

We will now have a robust polygon-polygon collision detection algorithm that will work for any 3D polygons.

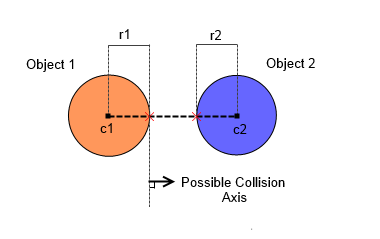
One thing that has changed in the above code is the additional check for previously found axes. As we now have to check all possible permutations between edges the number of possible collision axes has vastly increased. So being able to remove duplicate axes will save us a lot of time when we have to iterate through each axis later on.

**Spheres and Curves**

One very important side of collision detection we have ignored thus far is that of curved surfaces, such as spheres. A curve has infinite edges and infinite normals so being able to handle all possible permutations is no longer a viable option. Instead we will have to come up with another solution to handle these cases and produce a list of possible collision axes.

However, because we know that all our shapes are convex, there will only ever be one point of contact. So all we have to do is find and check a single axis to know whether the curved shape is colliding.

Looking at the simplest case (sphere-sphere), all we have to check is whether the distance between the two centre points is less than their combined radii to prove they are colliding. So going back to SAT, the only axis we need to check in this case is the direction between the two centre points.



In the sphere-sphere case what we are actually checking here is the closest point of object 1 to the closest point of object 2. Though in the sphere-sphere case that direction vector results in the same vector as the one produced by the two centre points.

In the sphere-polygon case this is not always true. If we look at the first figure below, the possible collision axis produced by the two centre points will give us a false positive. Therefore we must calculate the closest point of object two to the centre point of object 1. This is accomplished by iterating through each face in turn and deducing the closest point to the first object.

