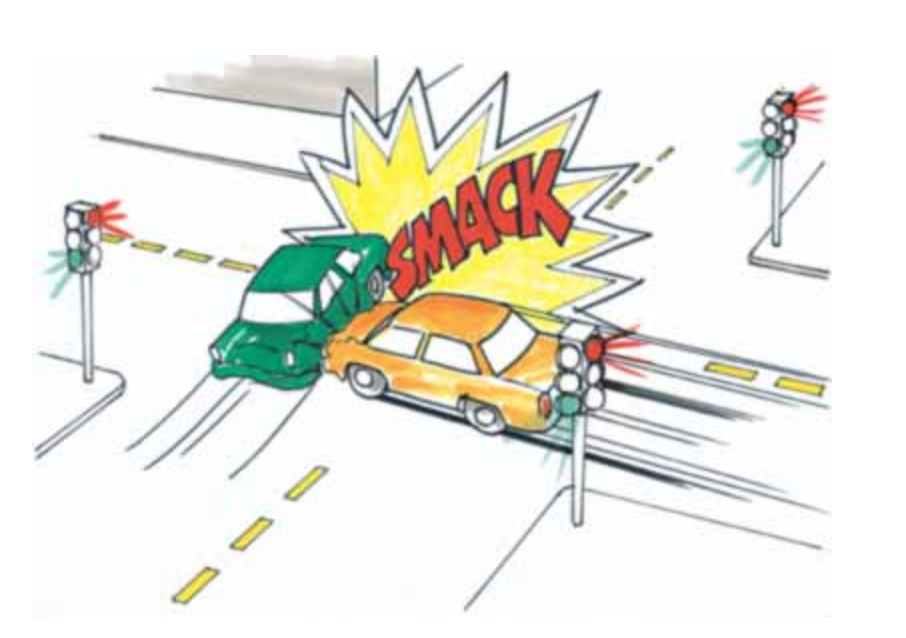
Physics Tutorial 6 – Separating Axis Theorem (Part 1)



**Summary**

In this tutorial we will begin looking into generic collision detection algorithms. We will be going into depth with the separating axis theorem specifically. Looking into the theory of how it works and how that can be translated into code.

**New Concepts**

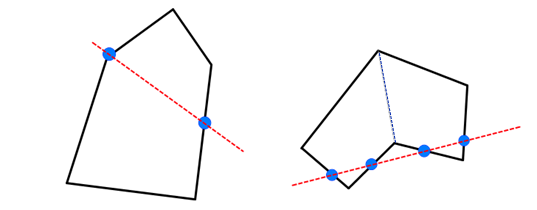
Convex and concave shapes, generic collision detection algorithms, 2D separating axis theorem

**Introduction**

A key part of the physics engine is being able to detect collisions between arbitrary shapes. In the previous tutorial we looked at ways of detecting collisions between known shapes such as sphere-sphere and sphere-plane. However, as the number of possible collision shapes increases so too does the number of bespoke collision detection functions. In today’s tutorial we will be looking into the Separating Axis Theorem (SAT), one specific example of a generic collision detection algorithm.

**Convex or Concave?**

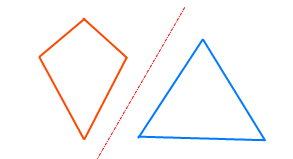
The first thing to note is that this algorithm only works with convex shapes. You can identify whether a shape is convex of concave by how many times a line intersects that shape. A convex shape will only ever have two points of intersection. This can be seen in left hand shape below. On the other hand a concave shape will have at least one instance where an intersecting line will have more than two points of contact.



Although concave shapes cannot themselves be easily collided, any concave shape can be split into a number smaller of convex shapes. This can be seen by splitting the right hand shape in two separate convex shapes along the blue dotted line.

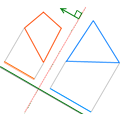
**Separating Axis Theorem**

The Separating Axis Theorem states that if two objects are NOT colliding, then a line (or plane in 3D) can be drawn between them. This is in fact the same proof we used for the sphere-sphere collisions in the previous tutorial.



So if we can find a single case where we can draw a line/plane between the two shapes without intersecting either of them then we can prove that the two shapes are not colliding.

This can be achieved by projecting all the points of each shape along the axis being tested. This will give us a single value describing the distance of that point along that given axis (This is the same projection used in the previous sphere-plane test). It is this value that can be used to check whether the two objects overlap. As can be seen below, if we project the two shapes along the separating axis (shown by extending the green normal infinitely in either direction) then we end up with two lines (orange and blue) that can easily be tested for overlap.



The actual code to find the minimum and maximum projections along a vector are already provided in SphereCollisionShape and CuboidCollisionShape. In the cuboid example, which is the generic example of a convex polygon in this tutorial, all vertices of the cuboid are projected along the axis and the closest and furthest distances are returned as the min/max points. In the sphere example, which is the example of a shape with potentially infinite points, the min/max points are calculated directly from what we know about the shape, in this case it is just *±radius \* axis.*

*(Clarify what a projection is)*

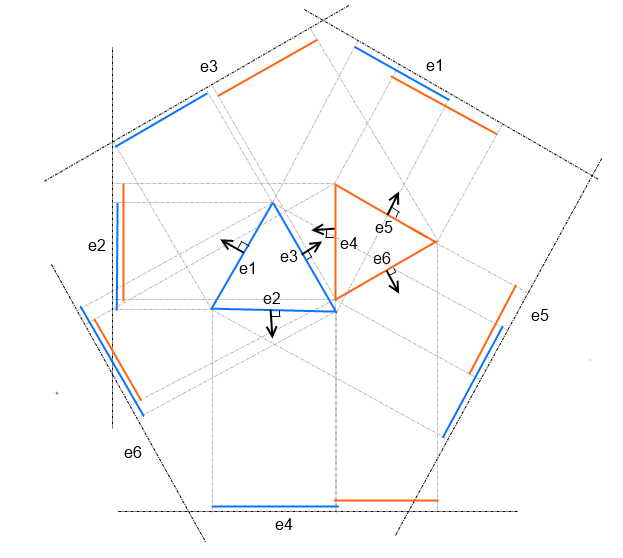
To project a point along an axis we use vector projection, defined as such.

Due to (our axis) being normalised can be simplified into , which gives us the physical location of the vertex along the given axis. For the purpose of identifying distance however, we use the single value produced by as the ‘distance’ on vector along our axis .

**Potential Separating Axes**

Now we have seen how we can prove two objects are NOT colliding in a given axis. The problem, and the main issue of the SAT algorithm, will be identifying all possible axes that will need to be evaluated.

The easiest way to do this is first to assume the shapes are not curved, and are in fact made up of a series of lines/faces. This will mean that the number of possible collision points is limited to each flat face. In this case, we can use the normals of the faces of each object as a possible collision axis.



The above diagram shows all possible collision axes between two 2D triangles. Each test (e1-e6) has been formed by the normal of the corresponding edge. As can be seen on all but the third test (e3) all tests return that the two triangles are in fact colliding. Such that every normal could potentially be a separating axis, and all normals must therefore be evaluated. However one benefit of the separating axis theorem is that as soon as a single separating axis is identified, the algorithm can exit early in knowledge that the two objects are not colliding. This can be seen in the e3 test above, where programmatically the algorithm could exit early without testing e4-e6.

It is also worth noting that parallel axes do not need to be checked multiple times, in an un-rotated square example this is equivalent to saying that checking the +Y normal produced from the top face is the same as the –Y normal produced by the bottom face. This is because we are only concerned with the distance to that plane from each point and not on which side of the plane each point lies (see sphere-plane example from the previous tutorial).

**The Algorithm**

So to summarise what we have learnt about the Separating Axis Theorem:-

* If there exists an axis in which the two objects do **not** overlap, then we can prove that they do not collide.
* If no axis exists in which they do not overlap, then we can safely assume that they **do** collide.
* The number of possible axes is the same as the number of faces on the objects combined.
* Parallel axes can be ignored.

<CODE>