Capsule to Convex: Edge Pruning via Gauss Map

Capsule Definition

A capsule is defined by two endpoints P_A and P_B, along with a radius value *r*. The segment of a capsule is defined by $P_B - P_A$.

Minkowski Difference

The Minkowski difference (MD) is defined as the set of vectors formed by subtracting each point in convex B from convex A:

$$A - B = \{a - b \mid a \in A, b \in B\}$$

The MD can be viewed as a set of points in Minkowski space, also referred to as *configuration space*. These points form a point cloud. The convex hull of the MD contains useful information for detecting the collision between A and B. Here are some observations from the convex hull of the MD:

1. The faces of the MD define the possible axes of separation between A and B.
2. An edge pair* may or may not contribute to a face on the MD.
3. Edge pairs contribute a face (specifically a parallelogram) on the MD if and only if their images on the Gauss Map intersect.

* An edge pair is defined as two edges, one each from each convex A and B.

Pruning Edge Pairs

A naive collision detection implementation, via the Separating Axis Theorem, requires cubic time in the number of edges to test for axes of separation. Pruning unnecessary pairs is an important optimization. Furthermore pruning unnecessary edge pairs can lead to simplification of implementation.

Observation 3 allows a predicate to be constructed to cull unnecessary pairs: if a pair does not produce an intersecting pair of images on the Gauss map, the pair does not define a potential axis of separation. Such edge pairs can be safely ignored.

Convex Gaussian Image

A convex is made of vertices, faces and edges (the topology). The Gauss map of a convex is defined by the topology of a convex. The face of each convex forms a point on the Gauss map. Points on the Gauss Map are connected when two faces on the convex are adjacent to one another; edges of the convex form arcs as their Gaussian images. Vertices then map to faces on the Gauss map.

Page 75-76 of *Game Physics Pearls* contains some good example images of convex polytopes and their associated Gaussian images.

Voronoi Regions and the Gauss Map

We can define a function ***G*** to construct the Gauss map ***S*** of a convex A:

$$S = G(\,A\,)$$

One way to think of the Gauss map is: the Gauss map visualizes the Voronoi regions of a convex in a useful manner. If two Gauss maps from two different convex polytopes are overlaid upon one another, the overlap of Voronoi regions can easily be detected.

One gotcha is to overlay Gaussian images directly. Similar to the construction of the MD, the construction of S from A and B requires A to be negated. This places features of A's image, that are oriented towards B, in the same region of B's image.

$$S_A = G(\,A\,), \qquad S_B = G(\,B\,)$$

$$S_{AB} = S_B - S_A$$

Arc Overlap Test

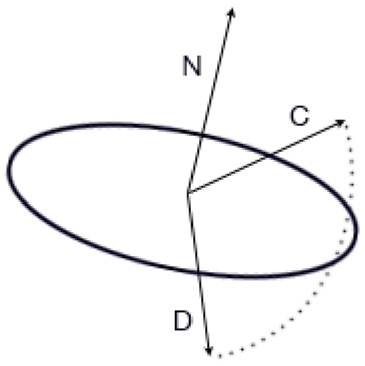
S of a capsule need not be constructed. Instead, S of the segment of a capsule need be constructed. Since S is composed of two vertices and a single edge, S is a unit circle on the unit sphere.

The pruning predicate needs to detect the intersection of a unit circle and an arc on the unit sphere. We shall treat $S_{Capsule}$ as a plane and S_{edge} as two points on the unit sphere. $S_{Capsule}$ is represented by:

$$ax + by + cz - d = 0$$

Where ***d*** is zero, since the plane is centered on the origin. ***a b*** and ***c*** are formed by the capsule's segment. By classifying the vertices of S_{edge} against the plane $S_{Capsule}$ we can form a simple intersection test.

Let P_A and P_B be the endpoints of $S_{capsule}$ where $N = P_B - P_A$. Let C and D be endpoints of the arc of S_{edge}.



The overlap predicate can be defined as:

$$\alpha = C \cdot N * D \cdot N$$

$$\alpha \begin{cases} \alpha < 0 \rightarrow intersection \\ \alpha \geq 0 \rightarrow no\ intersection \end{cases}$$

The sign of α determines if the arc composing S_{edge} intersects with the plane from $S_{Capsule}$. In C++ this may look like:

```cpp
int IsMinkowskiFace( Vec3 N, Vec3 C, Vec3 D )
{
    // Hemisphere test
    return Dot( N, C ) * Dot( N, D ) < 0;
}
```