# Algorithm

In this section, I will explain my algorithm for each key feature.

**Circular arc of edge**

In the original sweep algorithm, we monitor the events, “Insert”, “Remove”, and “Swap” to maintain the order of line segments. The first challenge to apply this algorithm to circle-circle intersections is how to define those events for circles. For this purpose, I split each circle into four quarters as shown below. As a result, we can define the “Insert” and “Remove” events similar to the line segments.

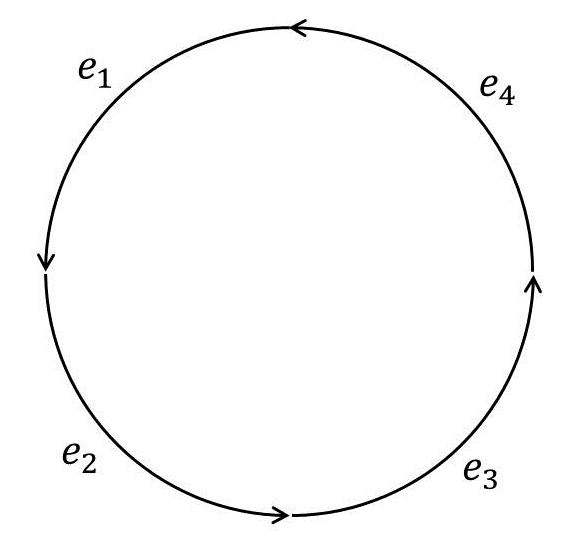


Fig 1. Each circle is split into four circular arcs to compute the “Insert” and “Remove” events correctly.

There are several benefits to use four circular arcs instead of two semicircles. First, we can use the same algorithm to compute the left most edge which is necessary to find the outer boundary for the inner loops. Second, we can use the same notion of *next* edge. If we use the two semicircles instead, there will be two edges that connect the same pair of vertices. Thus, we need to come up with a new idea to define which edge is the next in terms of the counter clockwise order, which might be cumbersome.

**X coordinate ordering of edges on the sweepline**

When an “Insert” event occurs, the sweepline algorithm adds the new edge into the binary search tree so that we can keep tracking the adjacent edges. To deal with the circles, we have to modify the algorithm for judging which edge is left or right. For the case of Fig. 2 (a), when the new edge is added, the edge is right of the edge . On the other hand, for the case of Fig. 2 (b), the edge is left of the edge .

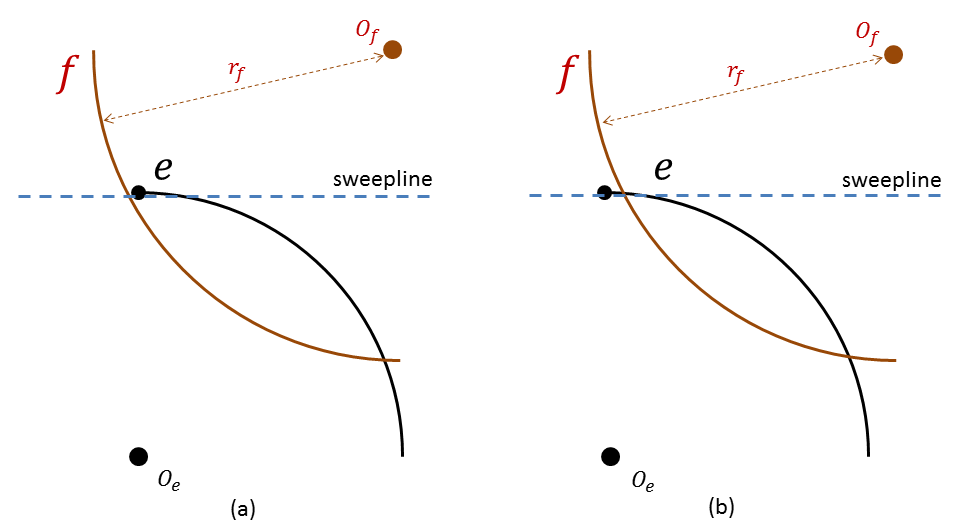


Fig 2. How can we judge that the edge is right of the edge in the case (a), while the edge is left of the edge in the case (b)?

Let be the newly added circular arc, be one of the existing circular arc, be the center of its circle, be its radius, and be the vertex of which has the higher Y coordinate than the other side vertex of . When an edge is added, and is compared with the existing edge , the edge is left of the edge if and only if one of the following conditions are satisfied:

1. The edge is on the left side of its circle, the X coordinate of is less than the X coordinate of , and the distance between and is greater than ,
2. The edge is on the right side of its circle, and the X coordinate of is less than the X coordinate of or the distance between and is less than .

**Circle-circle intersection**

Even though there are four circular arcs for each circle, we do not want to compute the intersections of circles for four times. Instead, I compute the intersections of each pair of circles at most once, and store them in hash table so that the computed intersections can be retrieved later in the constant time. For the computation of the circle-circle intersections, we can use the following equation:

Where and are the radii of the two circles, respectively and is the distance between two centers of the circles. Once is computed, can be easily computed by Pythagorean theorem. Thus, given the coordinate of the centers of two circles and their radii, we can compute the two intersections between them.

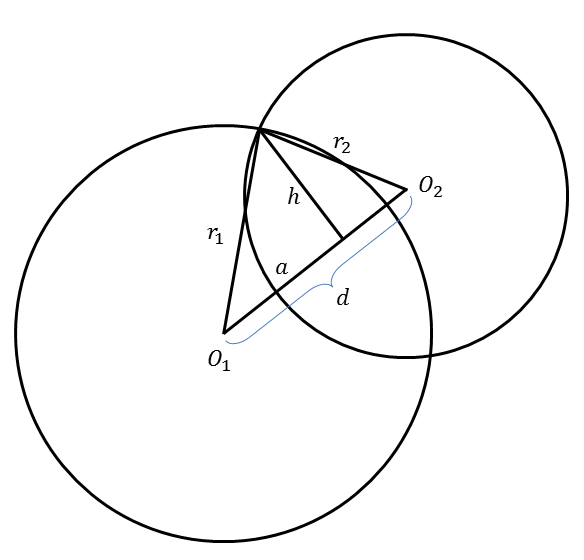


Fig 3. The two intersections of two circles are computed by first calculating the length of , then calculating the length of .

**Swap of edges in case of two intersections**

If there is only one intersection between two edges, we just need to swap two edges and in the binary search tree to update the adjacency. However, if there are two intersections, we have to be a little more careful to take care of this. In the case of Fig. 4, when the edge is added, it is left of the edge . Then, when the sweepline reaches , the first “Swap” event occurs, and the order of and is swapped. Then, when the sweepline reaches , the second “Swap” event occurs, and since the order of and is already swapped by that time, it swaps their order back.

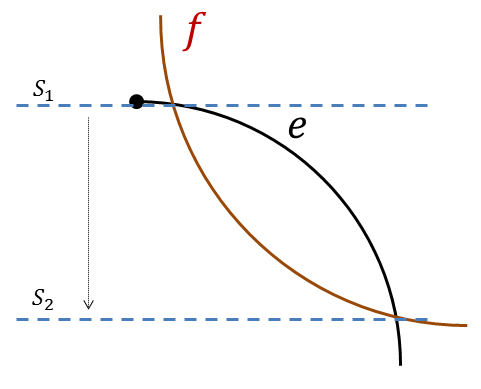


Fig 4. In the case that there are two intersections between two edges, the first “Swap” event swaps the order of these two edges, but the second “Swap” event swaps back their order.

Therefore, for the second “Swap” event, we have to put the edges in the swapped order.

# Implementation

In this section, I will explain how I modified each function of ACP library to implement the aforementioned algorithm.

**Point.h**

I changed the constructor InputPoint(const PV2 &ip) to a public method, because when I compute the coordinate of the vertices of the circular arcs, I get PV2 object for them, and I need to

**Edge::leftOf()**

Aeaf

**Edge::intersects()**

Fae

**Edge::withinArc()**

**Arrangement::swap()**

**Arrangement::check()**

**Arrangement::computeNumComponent()**

**Arangement::overlay()**