UNIVERSITY OF WATERLOO

CS 759 Report: Killing Time

Theo Belaire
20415730
Bryan Coutts
20428420

April 6, 2015

Some terminology, I use the term *included* point to refer to points that are to be included inside the blob, and *excluded* points to refer to points that must be on the exterior of the blob. Unless specified, polygon can mean non-convex polygon.

1. GENERATION OF THE CONVEX HULL

Our code uses the giftwrap algorithm to generate the convex hull. This runs in (i^2) time, where i is the number of points included in the set.
It works by picking the leftmost point, and calculating the angle that is formed with each other point in the set, and picking the point that forms the angle closest to a straight line with the previous point.
At the conclusion of this step, we have a list of included points in clockwise order, which forms our polygon. Currently it's convex.

2. CHUNKIFICATION

To test if a point is within any polygon, we can use the Jordon Curve Theorem. This states that if we trace a ray from the point in any direction, it will cross an edge of the polygon an even number of times if and only if the point is outside the polygon.

This is computationally simple to check, and we use it as a subroutine.
For each excluded point p, we check if it's in the polygon, and if it is, we attempt to find the edge that it is closest to. We can calculate the normal to an edge uv easily enough, since there are only two possible unit vectors orthogonal to the vector $\vec{uv}$. Since we know our points are ordered by clockwise order, there is exactly one outwards pointing unit vector, call it $\vec{n}$.
To check the distance between a point p and the edge uv, we can consider the value

$$\vec{pu} \cdot \vec{n}$$

which is just the distance projected onto the normal.
We also need to check that the point p is closest to a point on the line segment, not just the infinite line, so we can compare

$$\vec{pu} \cdot \vec{vu}$$

and

$$\vec{pv} \cdot \vec{uv}$$

which should both be positive if p is between the two points.
Once we've found the unique closest line segment to an exterior point, we insert it into the list between the two.

3. Calculating Radii

The *radius* of a point determines how large a circle is used when drawing the blob around it. If all the radii were very small, it would not look very different from just the polygon. However, we cannot increase the radii without bound, since we require that the circles for included and excluded lists cannot intersect. It looks better if they don't even meet, so using exactly half the distance to the nearest point of the opposite type is not quite the best. Instead, we have a tunable parameter, which is around 2.5.
Also, we don't want to create a scenario where we close the "neck" on an exterior point that was inside the convex hull, trapping it inside. This could happen we increased the radii of two interior points
Words

4. Absorbing points near lines

Once we have all the radii, we almost know the final shape of the blob. Ho

5. Uncrossing

6. Arc computation

Once we have a final boundary, and radii, we can draw the blob.
For each edge uv, we consider a pair of circles centered at u and v, with the radii r_u and r_v.

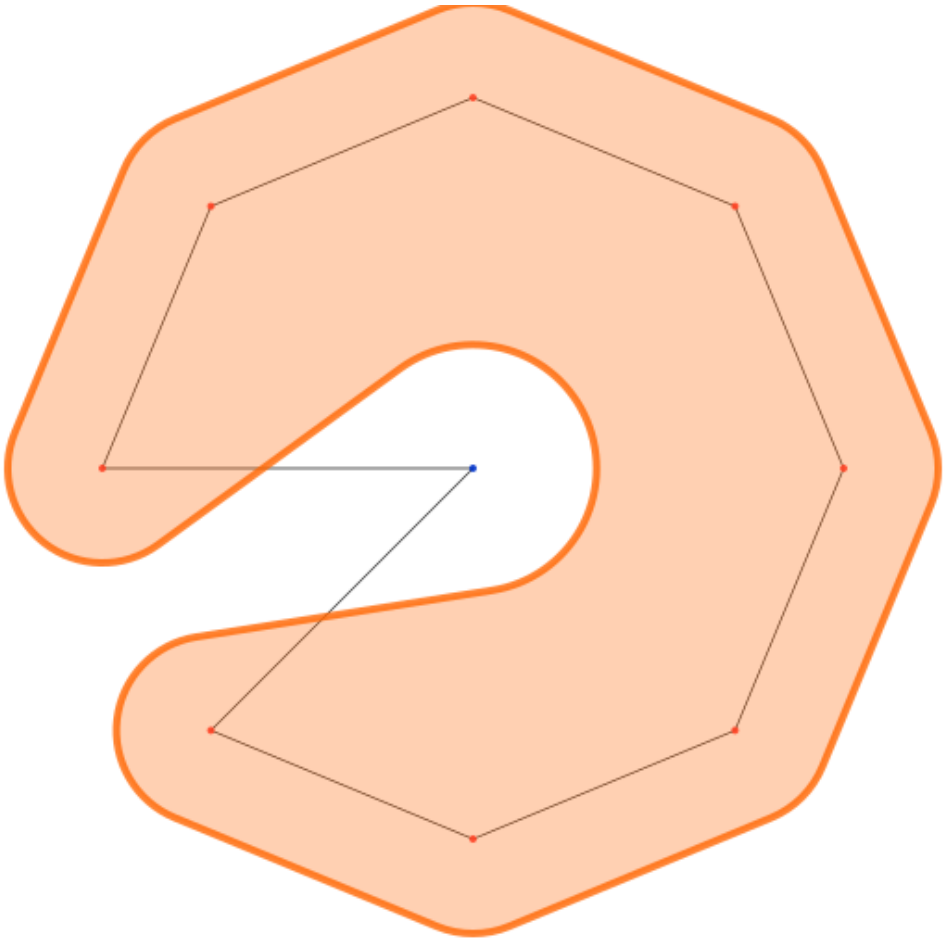

Figure 3.1: “An example of why we need the second rule”

Consider as if we were to take a piece of string and wrap it around the outside of these circles. It would have three segments, first when it was in contact with the first circle, then it would leave and travel along a line tangent to both circles, then it would arrive at the second circle and travel along it before leaving somewhere on the other side.