UNIVERSITY OF WATERLOO

CS 759 Report: Killing Time

Theo Belaire
20415730
Bryan Coutts
20428420

April 7, 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.

Our algorithm works as follows:

1. Take the convex hull of the included points.
2. Fix the convex hull, so that no excluded points are in its interior.
3. Compute the radii for each point.
4. Add nearby points to the polygon.
5. Remove "crossover" points.
6. Draw blob around the vertices of the polygon.

1. GENERATION OF THE CONVEX HULL

Our code uses the giftwrap algorithm to generate the convex hull. This runs in $\emptyset(i^2)$ 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. Point Exclusion

In this step, we will fix our polygon, so that no excluded point is in its interior. We will do so by removing triangles from our polytope; this will cause it to no longer be convex.

To test if an excluded point is within our polygon, we use the Jordan Curve Theorem. One of its consequences is 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.

For each excluded point p in the interior of our polygon, we find the polygon edge closest to p. We consider only the edges whose endpoints p is between. We then insert p into our list of polygon vertices, between the endpoints of this edge. This removes the triangle formed by the edge and p from our polygon.

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

4. Absorbing points near lines

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

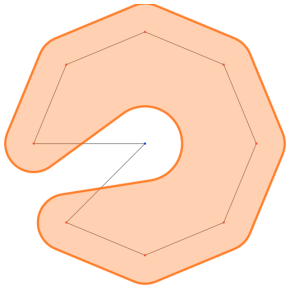

Figure 3.1: “An example of why we need the second rule”

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.
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.