MANIPULATING POLYGONS

REBECCA HOBERG AND MARY SOLBRIG

Abstract. We will implement various algorithms on polygons (not necessarily convex). The user will input an ordered list of vertices. The program will then initialize a list of polygons that correspond to those vertices. If the vertices the user gives have self-intersections, more than one polygon may be created. Functions we will compute include contains (whether a point or polygon is contained in a polygon) intersection of two polygons, and union of two polygons.

Contents

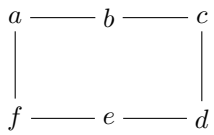
1. Our Polygons Class

The polygons class that we will implement has a list of polygons (I think we're calling it polygons), where each polygon in the list is a list of vertices that ends with its starting vertex. It will also have the following functions:

(1) Union
(2) Intersection
(3) Area
(4) Containment
(5) Perimeter

2. Examples

Here is an example of how our code works. Suppose you input the sequence of points $[a, b, c, d, e, f]$, corresponding to the following diagram.



Our code would then set the polygons list to be $[[a, c, d, f, a]]$.

Date: May 22, 2013.

If on the same diagram we were instead given the list $[a, b, e, d, c, e, f]$, our code would set the polygons list to be $[[a, b, e, f, a], [c, e, d, c]]$.

3. Implementation Details

The initialization of polygons calls the function set_corners, which does the following:

(1) Removes any colinear points.
(2) If any segment intersects a non-adjacent segment, it computes the point of intersection and splits the list into two separate lists of vertices.
(3) I haven't implemented this yet, but set_corners will also do some error-checking. For example, it needs to check it's given real points, and also that we don't have
(4) Right now set_corners allows you to give a polygon clockwise or counter-clockwise. We are planning to convert all of them to one direction, and there are a few ways we are considering doing this. One is to use the winding number, another is just to compute the area using the algorithm we currently have and see if it's positive or negative...

A helper function for the set_corners function is the function intersect(segment1,segment2) which returns (True, point) if the two segments intersect at a point, and (False, None) if they do not intersect. If their intersection is a segment, I'm currently having it return (True, None), but this may change depending on what it seems like we need.

Definition 3.1. The *winding number* of a curve in a plane around a point is, intuitively, the number of times it wraps around a point.

To determine whether or not a point is contained in a polygon, we use something like the concept of the winding number. (See definition 3.1.) If a point is contained within a polygon, its winding number will be nonzero - otherwise it will be zero. To figure out the winding number given a bunch of vertices, we consider a line extending vertically from the point in question, and, following the vertices around, we see how many times we cross the line in either direction. (yes this is super vague, we will flesh it out a lot better).

4. What is currently in Sage

Sage currently has some polygon capabilities, but most of it is done in much greater generality, considering polyhedra in n dimensions. While we were able to find the simple area function, we weren't able to find union and intersection (maybe they aren't there?) and we also didn't see Sage dealing with self-intersections. However, it is possible that we haven't looked hard enough. We will continue looking at that and include more specifics in our final draft.

5. Random

For the sake of putting in a math formula, $\cos^2(t) + \sin^2(t) = 1$. Taking the derivative of both sides, we see that

$$-2\cos(t)\sin(t) + 2\sin(t)\cos(t) = 0,$$

which makes a whole lot of sense.