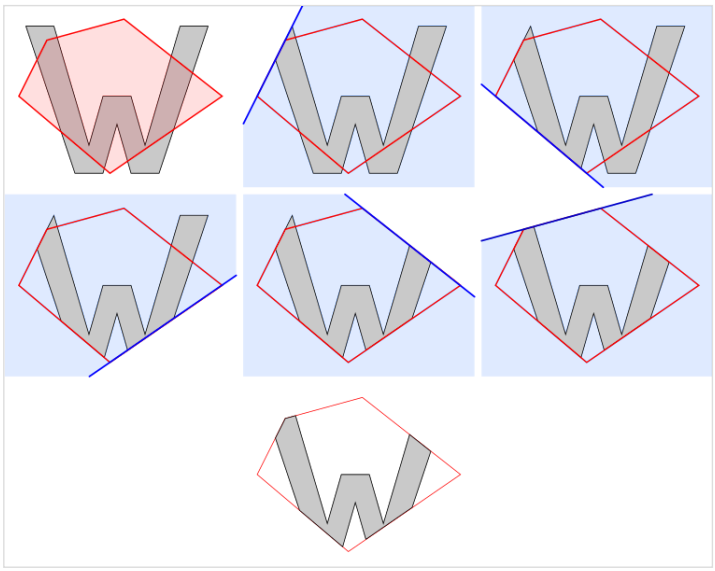
Sutherland–Hodgman algorithm

The **Sutherland–Hodgman algorithm** is an algorithm used for clipping polygons. It works by extending each line of the convex *clip polygon* in turn and selecting only vertices from the *subject polygon* that are on the visible side.

Description

The algorithm begins with an input list of all vertices in the subject polygon. Next, one side of the clip polygon is extended infinitely in both directions, and the path of the subject polygon is traversed. Vertices from the input list are inserted into an output list if they lie on the visible side of the extended clip polygon line, and new vertices are added to the output list where the subject polygon path crosses the extended clip polygon line.

This process is repeated iteratively for each clip polygon side, using the output list from one stage as the input list for the next. Once all sides of the clip polygon have been processed, the final generated list of vertices defines a new single polygon that is entirely visible. Note that if the subject polygon was concave at vertices outside the clipping polygon, the new polygon may have coincident (i.e., overlapping) edges – this is acceptable for rendering, but not for other applications such as computing shadows.



All steps for clipping concave polygon 'W' with a 5-sided convex polygon

The Weiler–Atherton algorithm overcomes this by returning a set of divided polygons, but is more complex and computationally more expensive, so Sutherland–Hodgman is used for many rendering applications. Sutherland–Hodgman can also be extended into 3D space by clipping the polygon paths based on the boundaries of planes defined by the viewing space.

Pseudocode

Given a list of edges in a clip polygon, and a list of vertices in a subject polygon, the following procedure clips the subject polygon against the clip polygon.

```
List outputList = subjectPolygon;

for (Edge clipEdge in clipPolygon) do
    List inputList = outputList;
    outputList.clear();

    for (int i = 0; i < inputList.count; i += 1) do
        Point current_point = inputList[i];
        Point prev_point = inputList[(i - 1) % inputList.count];

        Point Intersecting_point = ComputeIntersection(prev_point, current_point, clipEdge)

        if (current_point inside clipEdge) then
            if (prev_point not inside clipEdge) then
                outputList.add(Intersecting_point);
            end if
            outputList.add(current_point);

        else if (prev_point inside clipEdge) then
            outputList.add(Intersecting_point);
        end if

    done
done
```

The vertices of the clipped polygon are to be found in *outputList* when the algorithm terminates. Note that a point is defined as being *inside* an edge if it lies on the same side of the edge as the remainder of the polygon. If the vertices of the clip polygon are consistently listed in a counter-clockwise direction, then this is equivalent to testing whether the point lies to the left of the line (left means *inside*, while right means *outside*), and can be implemented simply by using a cross product.

ComputeIntersection is a function, omitted here for clarity, which returns the intersection of a line segment and an infinite edge. Note that the intersecting point is only added to the output list when the intersection is known to exist, therefore both lines can always be treated as being infinitely long.

Implementations

A Python implementation of the Sutherland-Hodgman can be found here (https://github.com/mdabdk/sutherland-hodgman).

See also

Other polygon clipping algorithms:

- Weiler–Atherton clipping algorithm
- Vatti clipping algorithm

On the subject of clipping:

- Clipping (computer graphics)
- Clipping (in rasterisation)
- Line clipping algorithms

References

- Mel Slater, Anthony Steed, Yiorgos Chrysanthou: *Computer Graphics and Virtual Environments: From Realism to Real-Time.* Addison Wesley, 2002. ISBN 0-201-62420-6.

- Ivan Sutherland, Gary W. Hodgman: *Reentrant Polygon Clipping.* Communications of the ACM, vol. 17, pp. 32–42, 1974

External links

- Polygon clipping and filling (http://www.cs.drexel.edu/~david/Classes/CS430/Lectures/L-05_Polygons.6.pdf) Describes the algorithm using images that are easy to understand.
- Rosetta Code example (https://rosettacode.org/wiki/Sutherland-Hodgman_polygon_clipping)