An edge labeling approach to concave polygon clipping

Klamer Schutte

PREPRINT submitted 7 July 1995 to ACM Transactions on Graphics

Correspondence

K. Schutte
Pattern Recognition Group
Delft University of Technology
Lorentzweg 1
2628 CJ Delft
The Netherlands
e-mail: klamer@ph.tn.tudelft.nl
phone: +31-15-786054
telefax: +31-15-626740

Abstract

This paper describes an algorithm to perform clipping of two possibly concave polygons. The approach labels the edges of the input polygons. This labeling is used in classifying the resulting polygons in the sets $A \cap B$, $A \setminus B$ and $B \setminus A$. It is shown that its worst-case time complexity is not worse than the worst-case complexity of the problem of polygon clipping itself. Suggestions are made how the average time complexity can be improved.

1 Introduction

The process of clipping two polygons is needed in applications ranging from all types of computer graphics applications to path planning and image processing[1]. The polygon clipping algorithm described in this paper calculates the analytical sets of the intersection and the difference of two input 2-D polygons that may be concave.

In literature some approaches to polygon clipping are described. Weiler and Atherton [4, 5] describe an approach which doubles the input polygons in an inside and an outside contour. Intersection points between the double contoured polygons are calculated. Using a given set of rules the output polygons are created. The disadvantage of this approach is that the set of rules given does not cater for all special

cases possible. Sechrest and Greenberg [2] describe an approach which is based on horizontal strips. A strip is bounded by events such as vertices and edge crossings. Vatti [3] uses the same notion of strips. The problem of both strip-based approaches is that special cases arise with more then one event on a strip bound. An example of such a special case is a horizontal line. Schutte [1, appendix A] gives an algorithm which is based on labeling edges. The algorithm described by Vatti can cope with polygons with holes and self-intersecting polygons. The algorithms described by Weiler and Atherton, and by Sechrest and Greenberg can deal with polygons with holes using a method similar to the one described in section 7.2.

The algorithm described in this paper is very similar to the algorithm given in [1, appendix A]. The difference is that the algorithm described in this paper can cope with background holes between the input polygons.

Initial constraints set to the input polygons are:

- The polygons should not have holes.
- The polygons should not be self-intersecting.
- The polygons are clockwise oriented.

In section 7 is shown how the algorithm can be augmented to circumvent these constraints.

2 The basic algorithm

Figure 1: Two input polygons. The area shared by the polygons is shaded with the darkest grey.

The clipping process consists of the following steps:

1. Calculate the intersections between the two input polygons A and B. This results in the same polygons, with the difference that the intersection points with the other polygon are added as vertices to the polygons.

2. Label the edges from both polygons to **Inside**, **Shared**, or **Outside**. **Inside** means that an edge is inside the other polygon. **Shared** means that both polygons share this edge. **Outside** means that an edge is outside the other polygon.
3. Find the minimal polygons which are created by the intersection.
4. Classify all minimal polygons into the output sets $A \cap B$, $A \setminus B$ and $B \setminus A$.

The next section will show how the intersections can be computed. Section 4 describes the process of labeling the edges. Subsequently, section 5 will explain how to find the minimal polygons. Section 6 finishes the description of the basic algorithm, explaining how the minimal polygons are classified.

3 Calculating the intersections

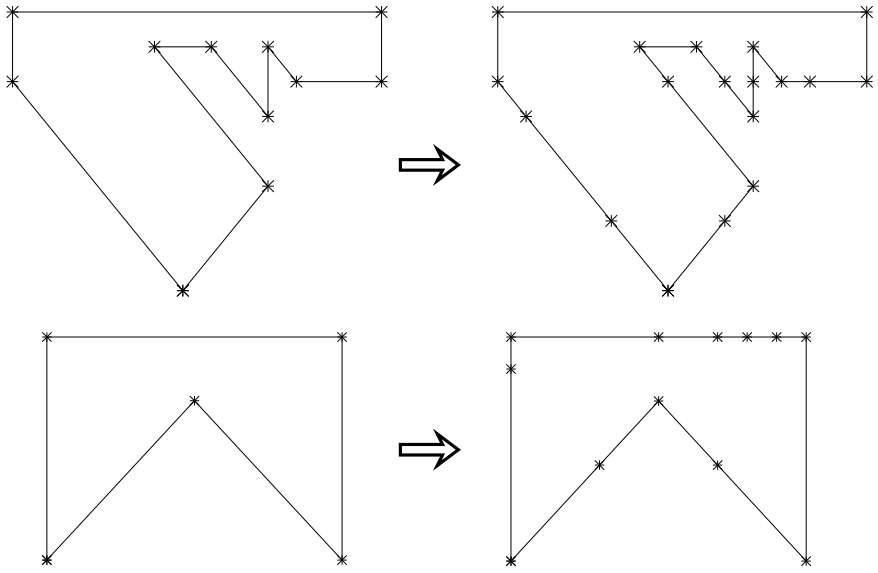

Figure 2: The polygons before and after intersection calculation.

The intersection points between two polygons are calculated by intersecting all the edges of the first polygon with all the vertices of the second polygon. These intersection points are added to the polygons as vertices. The new created vertices do contain a link to the corresponding vertices on the other polygon. We call the resulting polygons augmented polygons.

We have a special case if we do not find any intersections. This means that we have one of following situations:

1. Polygon A is inside B. This is the case when any vertex of A is inside B, which easily can be tested. If A is inside B, we arbitrarily split B in two polygons, with A somewhere on the split. We have to do this, because we do not want polygons with holes.

2. Polygon B is inside B. Similar handled to the case above.

3. The polygons A and B are disjunct.

In all these three cases we are finished.

4 Labeling edges

All edges in the augmented polygons should be labeled **Inside**, **Shared**, or **Outside**. An edge is **Shared** if both vertices of this edge are connected to the other polygon and if the two vertices they are connected to on the other polygon are the two vertices of one edge.

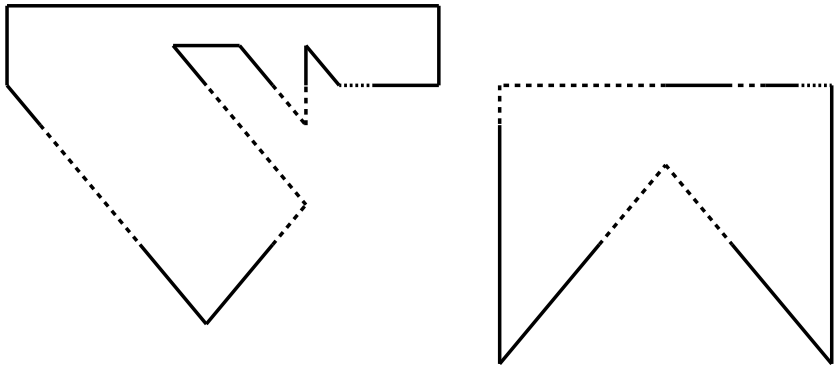

Figure 3: The labeling of the edges of the augmented polygons. The continues lines are **Outside**, the large stipples **Inside** and the short stipple **Shared**.

The resulting problem is to see whether the edge is **Inside** or **Outside** the other polygon. A tentative algorithm seems:

```
if is_inside(other_polygon, edge.some_point) then
  edge.label = Inside
else
  edge.label = Outside
```

Algorithm 1: A naive algorithm

The major problem in this approach is how to calculate edge.somepoint, this a point on the edge. The vertices themselves are not usable, as they can coincide with vertices on the other polygon. Any other point on an edge (such as the middle) results in numerical errors due to rounding. Such numerical errors can dislocate such a point from just beside an edge of the other polygon to on a position on that other edge – for which position it is not possible to decide whether the point is inside or outside the other polygon. A more stable algorithm is the following:

```
if edge.first_vertex.is_connected() then
  label_angle(edge, edge.first_vertex.connect)
```

```
else if edge.second_vertex.is_connected() then
  label_angle(edge, edge.second_vertex.connect)
else if is_inside(other_polygon, edge.a_vertex) then
  edge.label = Inside
else
  edge.label = Outside
```

Algorithm 2: Stable version of naive algorithm

A vertex is connected if it is an intersection point with the other polygon. The routine label_angle() given below:

```
label_angle( edge, vertex )
begin
  Vector prev = vertex - vertex.prev
  Vector next = vertex.next - vertex

  if angle(prev,next) ≥ π then
    if (angle(prev,edge) ≥ π) and (angle(next,edge) ≥ π) then
      edge.label = Outside
    else
      edge.label = Inside
  else
    if (angle(prev,edge) ≥ π) or (angle(next,edge ) ≥ π) then
      edge.label = Outside
    else
      edge.label = Inside
end
```

Algorithm 3: Computing label_angle

The idea behind this algorithm is that if one of the vertices on the edge is connected to the other polygon we can check for that vertex whether it lies on the inside or the outside of the other polygon. If neither of the vertices of the edge lay on an edge of the other polygon we can use both vertices of the edge in the test whether they are inside the other polygon. The calculation of angle(a,b) can be calculated using the cross product (here we assume that the z component of a and b are 0): $(a \times b)_z = a_x b_y - a_y b_x = |a||b|sin\phi$. If this quantity is positive the angle ϕ is smaller then π else ϕ is bigger then π.

5 Finding minimal polygons

All the edges from both augmented polygons are doubled in a forward and a backward edge, which we will call directed edges. **Shared** edges are doubled only once. Figure 4 shows the directed edges. For all these directed edges we search for the smallest clockwise oriented polygons it is part of. This is performed by following the directed edges such that at every intersection we proceed with that directed edge which starts at that intersection, and which has the smallest angle with the previous directed edge. A directed edge can only be part of one polygon. This also leads to one counter clockwise polygon, which is the outer contour of the union. This polygon should be deleted from the set of minimal polygons.

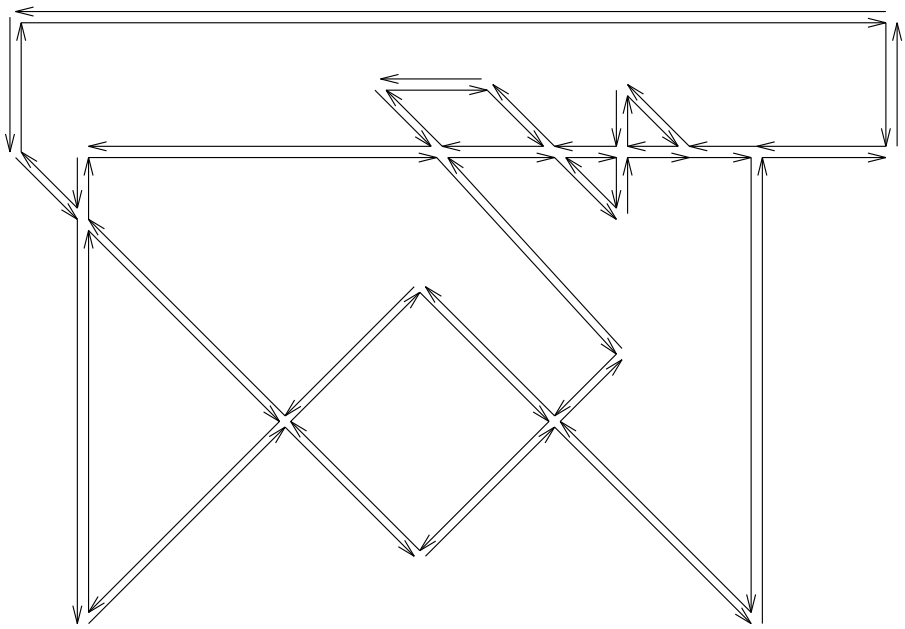

Figure 4: The forward and backward edges.

6 Classifying minimal polygons

The set of minimal polygons found covers the whole area of the union of the two input polygons. Now we have to classify each of the polygons to be member one of the sets $A \cap B$, $A \setminus B$ or $B \setminus A$. To perform this classification we introduce the term parenthood. Parenthood is defined as a relation between edges and input polygons and between minimal polygons and input polygons. If the parenthood is true it is sure that the object is part of the input polygon. If the parenthood is false it is sure that the object is not part of the input polygon. Besides the values true and false, a parenthood can also be unknown and mixed. Unknown means that we do not have conclusive evidence. Mixed means that contradictory evidence is present. This is only possible for minimal polygons, and means that the minimal polygon is a hole between the two input polygons.

The classification of the minimal polygons is performed in a two-step process. First we check the parenthood of each minimal polygon. After that, we decide to which set the minimal polygon belongs, based on its parenthood.

We are interested whether a minimal polygon is a child of each of the input polygons. This question can be answered by using the labeling of the edges. For each edge of a minimal polygon we compute the parenthood regarding the input polygons. This means that for both input polygons we decide the parenthood of the edges of the minimal polygons depending on their labeling:

Shared	the parenthood of the edge is **ParentUnknown**.
Outside	the parenthood is **ParentTrue** if this edge resulted from the input polygon currently considered. Otherwise it is **ParentFalse**.

Inside the parenthood is **ParentTrue** if the edge does not result from the input polygon currently considered (and thus is an edge from the other input polygon.) Otherwise we can not conclude anything about the parenthood, and the parenthood is **ParentUnknown**.

This can result in the following parenthoods for the polygons:

ParentUnknown the parenthood of all the edges of this polygon were **ParentUnknown**.

ParentTrue the parenthood of at least one of its edges was **ParentTrue**, and none was **ParentFalse**.

ParentFalse the parenthood of at least one of its edges was **ParentFalse**, and none was **ParentTrue**.

ParentMixed the parenthood of at least one of its edges was **ParentTrue** and the parenthood of at least one of its edges was **ParentFalse**.

Now we can classify the polygons:

1. If one of the parenthoods is **ParentMixed**, the polygon is a hole between the two input polygons, and thus not a member of any of the sets.
2. If both parenthoods are not **ParentFalse** the polygon is member of $A \cap B$.
3. Else one of the parenthoods is **ParentTrue**. The polygon is part of that polygon, and not of the other.

7 Extensions of the algorithm

In the introduction some constraints were posed which the input polygons should satisfy. However, with some preprocessing of the input these constraints can be circumvented.

7.1 Handling counter-clockwise polygons

The detection whether a polygon is clockwise or counter-clockwise oriented can be done by calculation of its area. If the area is positive the polygon is clockwise oriented. If the area is negative the polygon is counter-clockwise oriented. By reversing the order of the vertices, a counter-clockwise oriented polygon will become clockwise oriented. Such a check, and possible reversing, can be done at the very beginning of the process.

7.2 Handling polygons with holes

A polygon with a hole A can seen as polygon A_c with a hole A_h such that $A = A_c \setminus A_h$. We can use the algorithm described above to calculate $A_c \setminus A_h$, which is a set of polygons.

Thus to clip a polygon B with a polygon with a hole A we have to clip B with every polygon in the set $A_c \setminus A_h$.

7.3 Handling self-intersecting polygons

With a self-intersecting polygon we can calculate the intersection points with itself similar to section 3. This results in an augmented polygon, of which minimal polygons can be calculated similar to the procedure given in section 5.

The set of polygons retrieved by this procedure are not self intersecting, and does cover the input polygon including implicit holes. No easy way exists (to the knowledge of the author) to detect which of these polygons are implicit holes.

This means that input polygons which are self intersecting but do not include implicit holes can be used as input to the algorithm.

8 Computational expense

The worst case complexity of the polygon clipping problem is $O(nm)$, with n the number of vertices in polygon A and m the number of vertices in polygon B. This complexity is reached when every edge of the first polygon intersects with every edge of the other polygon. An example of this can be seen in figure 5, where the number of polygons in the set $A \cap B$ equals $nm/4$. However, for other cases the number of output polygons is smaller, with a minimum of 1.

The complexity of the algorithm described in this paper always is $O(nm)$ for the calculations of the intersections and the labeling of the edges, even in cases when the number of output polygons is much smaller then nm. The other parts of the algorithm have lower complexity.

8.1 Complexity of intersection calculation

Every edge in the first input polygon can intersect with every edge in the second input polygon. This results in $O(nm)$ complexity. If space partitioning techniques are used not every edge of one polygon has be checked to all the edges of the other polygon, but only those who are located in the same area element. This could result in $O(n + m)$ complexity for appropriate space partitioning techniques and simple problems.

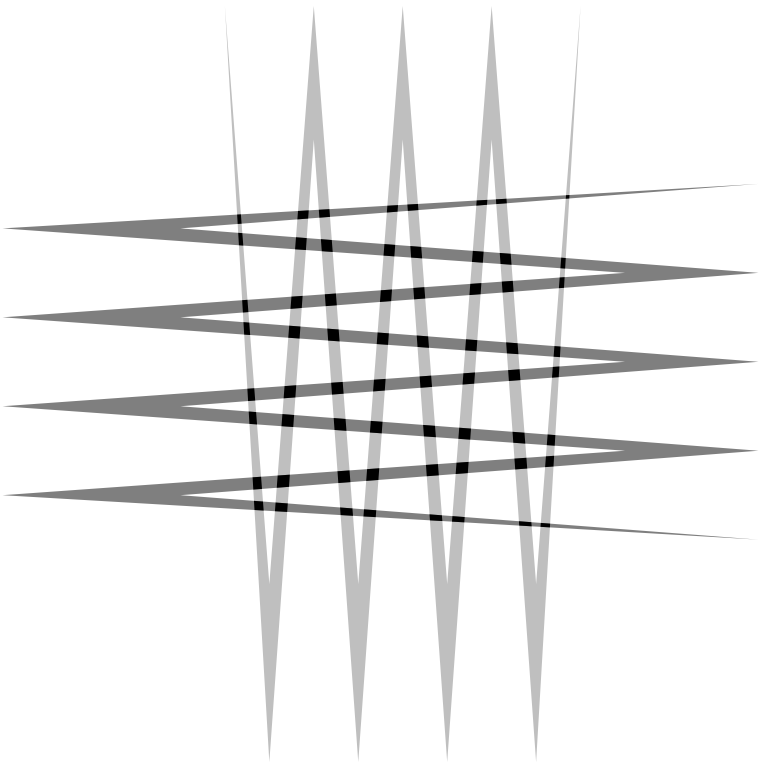

Figure 5: An example with complexity $O(nm)$: the number of polygons in $A \cap B$ (black in the figure) equals $nm/4$. This example scales up by adding more zig-zags.

8.2 Complexity of labeling edges

The algorithm proposed in section 4 makes use of is_inside() to label edges which are either completely outside or completely inside the other polygon. This check does have to test against all edges of the other polygon for concave polygons.

However, this check is not strictly necessary. We already know that the polygon is not completely inside or outside the other polygon, so there must be at least one intersection point with the other polygon. If we classify the edges connected to this vertex, we do not need to consider the edges completely inside or outside the other polygon. We simply can label them as **Unknown**, and relying on the edges connected to the intersection points to get the correct polygon labeling. This will result in the following algorithm instead of algorithm 2:

```
if edge.first_vertex.is_connected() then
  label_angle(edge, edge.first_vertex.connect)
else if edge.second_vertex.is_connected() then
  label_angle(edge, edge.second_vertex.connect)
else
  edge.label = Unknown
```

Algorithm 4: Improved version of stable algorithm

When making decisions about the parenthood of the edges as described in section 6 we use the rule that an edge label **Unknown** gives **ParentUnknown** as parenthood for this edge. This leads to an algorithm with complexity $O(m+n)$

Acknowledgments

Support from SION and NWO is gratefully acknowledged. Much of the work has been performed at the Measurement laboratory, Department of Electrical Engineering, University of Twente.

References

1. Klamer Schutte, Knowledge Based Recognition of Man-Made Objects, *PhD Thesis*, University of Twente, ISBN90-9006902-X, 1994

2. Stuart Sechrest and Donald P. Greenberg, A visible polygon reconstruction algorithm, *ACM: Computer Graphics*, **15**, 1981, 17-27

3. Bala R. Vatti, A Generic Solution to Polygon Clipping, *Communications of the ACM*, **25**, 1992, 58-63

4. Kevin Weiler and Peter Atherton, Hidden surface removal using polygon area sorting, *Proc. of the 4th ann. conf. on computer graphics and interactive techniques, 1977*, pp. 214-222

5. Kevin Weiler, Polygon Comparison using a Graph Representation, *SIGGRAPH 80, Computer Graphics*, **14**, 1980, 10-18