An Extension of Polygon Clipping To Resolve Degenerate Cases

Dae Hyun Kim[1] and Myoung-Jun Kim[2]

[1]Institute for Graphic Interfaces, cregeo@acm.org
[2]Ewha Womans University, mjkim@ewha.ac.kr

ABSTRACT

We present an extension of Greiner-Hormann polygon clipping that has been regarded as both computationally efficient and simple to implement. This extended version of the Greiner-Hormann polygon clipping elegantly handles the degenerate cases in a deterministic way, which in the previous approach is done by perturbing vertices in an indeterministic way.

Keywords: Clipping, Polygon, Degeneracy, CAD, Graphics.

1. INTRODUCTION

The algorithms described in this paper clip a polygon (referred to as the subject polygon) against a polygon (referred to as the clip polygon). Clipping 2D polygons has been regarded as a useful but also indispensable part in computer graphics and computer-aided design (CAD). In rendering, for example, it has been used to produce 3D images through hidden surface removal, and in parallel ray tracing system to split and distribute the objects of a scene to several processors. In CAD applications, it has been used to compute 2D B-Rep Boolean operations.

As far as we know, Greiner-Hormann algorithm [2] has the simplest data structure. Furthermore, it not only gives us intuitive understandings for complex cases (i.e., input polygons are concave or have self-intersections), but also outperforms the other general polygon-clipping algorithms such as [3][5][7]. However, it has one serious problem; degenerate cases that often occur in CAD applications are not handled properly. When no degenerate cases are found between a subject and clip polygon, Greiner-Hormann algorithm is regarded as a significant improvement on Weiler-Atherton algorithm [7][6][4]. Although Weiler-Atherton algorithm can handle not only degenerate cases but also inner holes, it requires a complete Boundary Representation (B-Rep), which is an overhead for some applications. In dealing with the degenerate cases, our extension outperforms the Weiler's approach in [6].

Although the predecessor of our approach is reported to handle degenerate cases by perturbing vertices, additional running of the overall algorithm is required for each perturbation, thus its running time becomes undeterminable. Moreover, the clipped results may be different depending on the perturbation direction.

In this paper we extend Greiner-Hormann algorithm in order to successfully handle the degenerate cases. In Section 2, we review the Greiner-Hormann algorithm. Section 3 shows the problems of perturbation approach to resolve degeneracy cases. In Section 4, the extended algorithm is presented.

2. REVIEW

A closed polygon P is represented by the ordered set of its vertices, $P_0, P_1, \cdots, P_n = P_0$. Therefore, the polygon consists of the line segments that consecutively connect the points P_i, i.e., $\overline{P_0P_1}$, $\overline{P_1P_2}, \cdots$, $\overline{P_{n-1}P_n} = \overline{P_{n-1}P_0}$.

For the completion of this paper, we briefly explain the concepts of the Greiner-Hormann algorithm with a simple example. The overall algorithm consists of three parts: (1) compute intersection points between two input polygons, (2) set the flags at the intersection vertices, and (3) traverse the two lists yielding the clipping. Since computing intersections between line segments has already abundant literature (e.g. see [1]), we focus only on the last two steps in this paper.

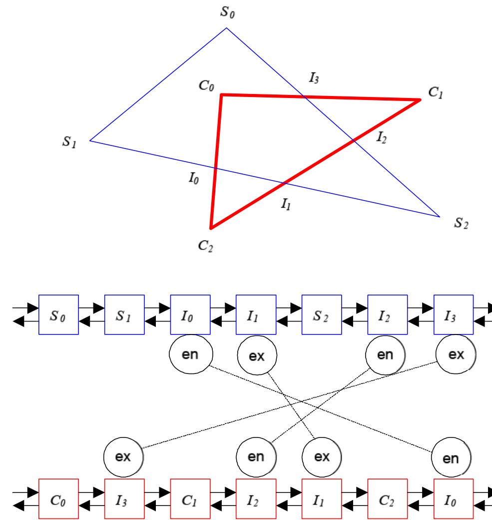

Fig. 1. Two polygons, $S = (S_1, S_2, S_3)$ and $C = (C_1, C_2, C_3)$. Bottom: After setting traversal flags.

Doubly linked list (DLL) with each node representing each vertex represents a polygon. After placing vertices into the list, intersections between each edge of the two polygons are computed and inserted into the list in order. Each intersection vertex has a reference to the coincident vertex in the other polygon; subsequently, this is called neighbor. Now comes the gist of the algorithm: set the **en** or **ex** flags at the intersection vertices, which are referenced while traversing the lists to get the clipping results. Traversing each list, mark each intersection vertex whether it is entry **en** or exit **ex** point to the other polygon's area on the traversal route. Let us explain this from the example shown in Fig. **1**. For the two polygons, S and C, whose vertices are denoted by S_i and C_i, respectively, three intersection vertices, I_0, I_1, and I_2, are computed and properly inserted into the existing DLLs. Then to mark the intersection vertices, traverse each list in the direction from S_0 to S_1. For example, at the vertex I_0 of the S, the point is the entry point to the other polygon, so mark it as **en**. In the same way, mark all the intersection vertices; note that the flag at the current intersection node can be set automatically from the previous flag by reversing the previous flag: for example, since I_0 's flag is **en** I_1 's flag becomes **ex**. Two DLLs generated after the intersection computation are shown.

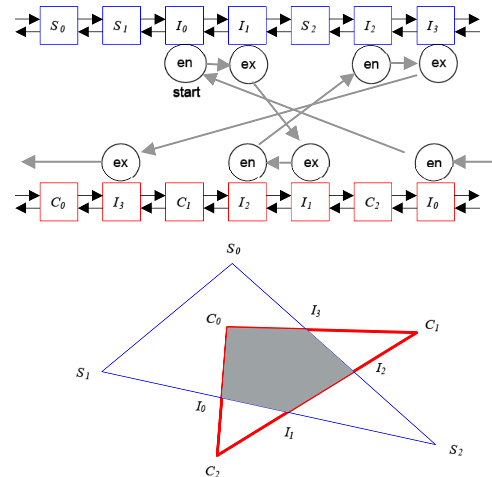

Fig. 2. Top: traversal route. Bottom: traversal result.

To traverse the graph, first locate any unused intersection vertex of a polygon and traverse the list as its traversal flag says: (1) **en** corresponds to the forward direction and (2) **ex** to backward. Unless another intersection vertex is met, the traversal continues on the same list. The flag of all the vertices on the traversal is deleted. On arrival at the next intersection vertex, the algorithm changes the list to the other; it jumps to the neighbor of the vertex. It traverses the list in the same manner as above. This process continues unless the traversal makes a cycle, thus yielding a polygon. When it consumes all the intersection points, the algorithm stops. Fig. 2 shows how to traverse the graph.

By reversing the polygon region (inside as outside and outside as inside) when setting traversal flags, different boolean operations can be achieved. For example, for the difference operation, set traversal flags for C against S the same as above, but set traversal flags for S against C regarding the outside of C as inside. For more detail, please refer to [2].

So far, it has been tacitly assumed that there are no degeneracies, i.e., each vertex of one polygon does not lie on an edge of the other polygon. To partly resolve degeneracy, [2]suggested to use vertex perturbation. Details on the problem of perturbing degenerate vertices follow in the next section.

3. PROBLEMS

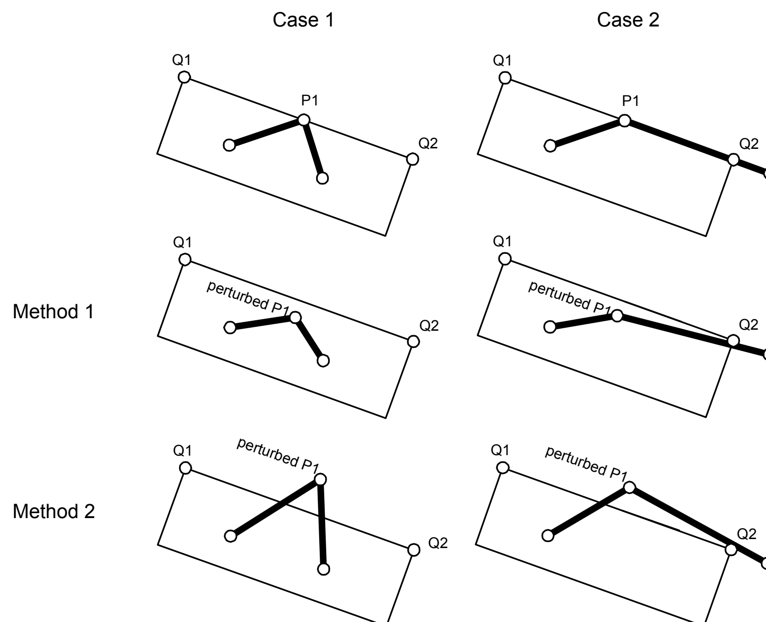

Fig. 3. Top Row: Two kinds of degeneracy. Middle Row: Perturbing.

To handle degenerate cases, perturbing vertices has been suggested in [2]. Two possible perturbations are shown in Fig. 3: moving the degenerate vertex into the inside of the other polygon or the outside. Let us assume that the line segment $\overline{P_1P_2}$ intersects $\overline{Q_1Q_2}$. For the intersection points, P_i and Q_i, we can extract their relative position with respect to each line segment: $0 \le \alpha \le 1$ and $0 \le \beta \le 1$. By perturbation, α and β can be fixed not to be zero or one. However, for the perturbed polygons we need to perform line/line intersections again; of course, there still exists possibility to find another degeneracy from the resulting polygons. This makes the overall algorithm indeterministic. The approach does not guarantee what result it can produce, for example, as in Fig. 4; for the difference operation, four triangles, one rectangle with a hole, or sometimes more results can be yielded.

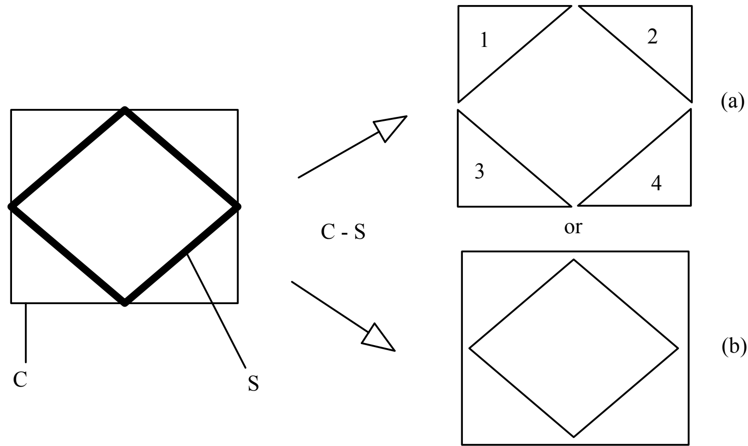

Fig. 4. The same operation can produce different results.

4. EXTENSION OF GREINER-HORMANN ALGORITHM

This paper addresses the degeneracy problem by extending the simple traversal rules of the Greiner-Hormann algorithm. We introduce two new traversal flags: **en/ex** and **ex/en**; however note that they are not new but a composite flag of **en** and **ex**. As implied from the left side of
Fig. **7** the **en/ex** flag means that **en**--that tells the event of entering the other polygon--and **ex**--that tells the event of exiting from the polygon-occurs at one vertex simultaneously.

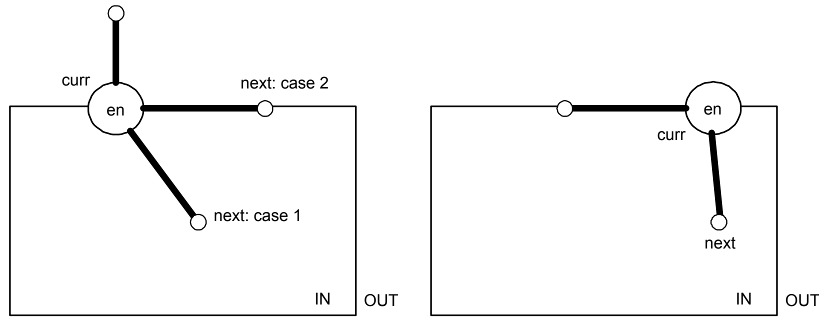

Fig. 5. Setting en flag.

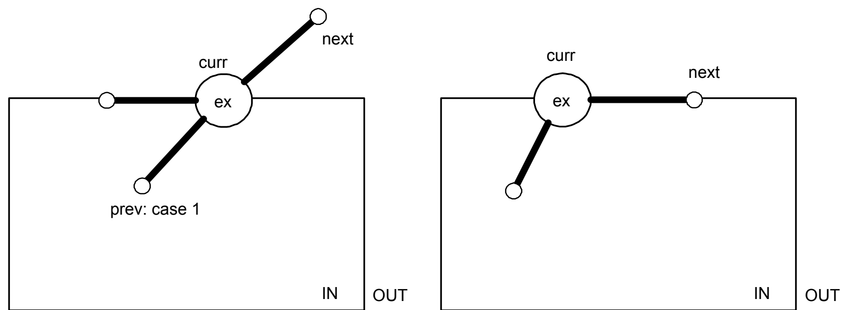

Fig. 6. Setting ex flag.

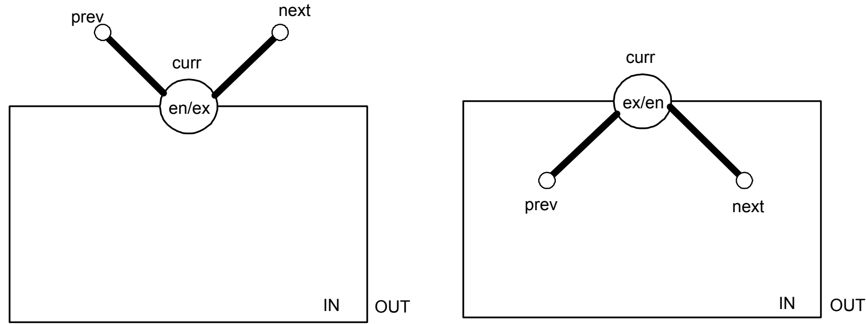

Fig. 7. Setting en/ex flag or ex/en.

As in [2], for the extended polygon clipping we need three processing stages; (1) compute intersections between the two input polygons, (2) set up traversal flags of the four kinds at each intersecting vertex, and (3) traverse the lists to yield the clipping results.

Degeneracy should be detected and handled during the intersection computation. For simplicity's sake, here we classify the degenerate cases into two primitive cases occurring between two half open line segments:

- One segment touches the other whilst they are not parallel to each other. See Fig. 8(a).
- Two segments are parallel to each other but coincide in part. See Fig. 8(b).

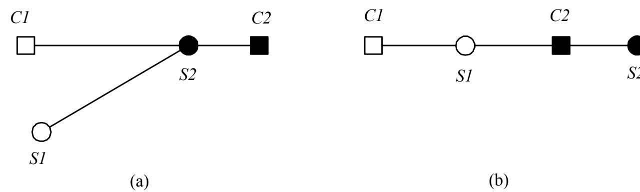

Fig. 8. Degeneracies.

In Fig. 8 the circles represent the vertex of the subject polygon, and the rectangles of the clipping polygon. To represent half closed interval one vertex of each line segment has been drawn filled; for example, $\overline{C_1C_2}$ has a parameter domain $(a,b]$, $a<b$, of half closed interval. Such distinction between closed and half closed interval helps not to produce redundant intersection points. Doubly linked lists before and after intersection computation for the situation of Fig. 8(a) are shown in Fig. 9, respectively.

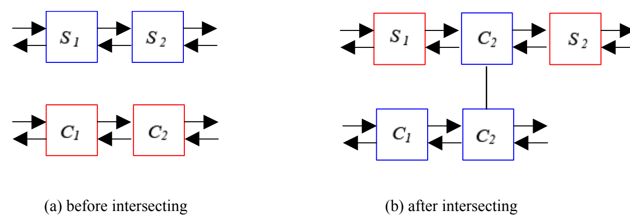

Fig. 9. Linked lists before and after intersection computation.

Data structure for a vertex contains eight fields as shown in Fig. 10: vertex position, pointers to the previous and next vertex, tag for intersecting vertex, traversal flag, link to the coincident intersection vertex (what is called neighbor) on the other polygon, and two additional data to run the clipping algorithm for degenerate cases. Two more fields have been added to [2]. We now explain how to set traversal flags in the extended polygon clipping. The algorithm scans one list tracking status of the two edges emanating from the current vertex. They consist of three status:

on: the edge is on the boundary of the other polygon.

in: the edge is inside the other polygon.

out: the edge is outside the other polygon.

```
vertex = {
        real x, y;                /* coordinates */
        vertex *next, *prev;      /* links */
        bool  isect;              /* intersection? */
        int   flag;               /* none, en, ex, en/ex. ex/en */
        vertex* neighbor;
        vertex* couple;           /* additional information 1 */
        bool  cross_change;       /* additional information 2 */
       }
```

Fig. 10. Vertex data structure.

We can think nine possible pairs from the above status to tell whether the current vertex is the entry point (**en**) to the other polygon or the exit point (**ex**); more importantly, **en/ex** and **ex/en**. For example, a pair (on, out) tells that the current vertex is exiting from the other polygon. The table below shows the nine possible pairs to determine the traversal flag for a vertex--note the degenerate cases have been already depicted at from Fig. 5 to Fig. 7.

(prev, next)	Traversal flag
(on, on)	none
(on, out)	**ex**
(on, in)	**en**
(out, on)	**en**
(in, on)	**ex**
(in, out)	**ex**
(out, in)	**en**
(in, in)	**ex/en**
(out, out)	**en/ex**

Let us assume that m intersection vertices, $I_0,\cdots,I_m$, where $I_0 = I_m$, have been found between two input polygons S and C. In case of no degenerate intersections, the following properties are preserved:

- The number of the intersection vertices, m, is always even.
- The traversal flag of I_j differs from that of I_{j-1}.

This assumption has been exploited in Greiner-Hormann algorithm to set the traversal flags; once the initial traversal flag at the first intersection vertex is known, the rest can be set automatically without further examination. However, this assumption does not always approve practical uses; for example, **en** flag often follows **en** flag, as shown in Fig. **11**. Our approach to comply the assumption even in degenerate cases is to use **ex/en**(or **en/ex**) flags or to couple two vertices of the same flag, using the *couple* field in the vertex data structure.

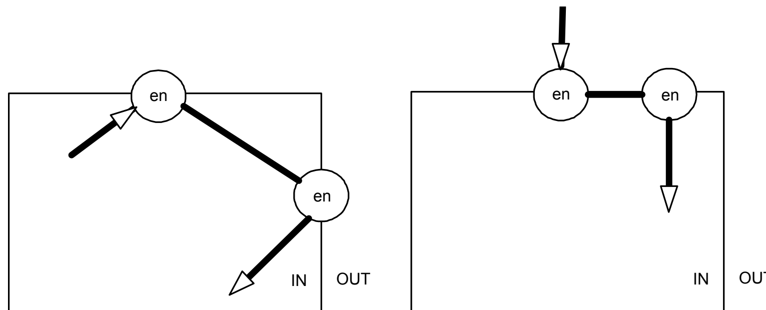

Fig. 11. Degenerate vertex flags.

Now we explain the extension of Greiner-Hormann algorithm. Similarly to the above, it consists of two steps:

(1) Set traversal flags for C.
(2) Select starting vertex for each traversal.
    (2-1) Traverse the list from the starting point yielding

4.1 Traversal Flags

At step (1) above, each intersection vertex C_i of the clipping polygon C is examined. If the edge $\overline{C_{i-1}C_i}$ is on S the previous edge status γ_{i-1} is assigned *on*. Otherwise, it is tested whether the midpoint of the edge falls inside or outside S; accordingly, the previous edge stat will have either *in* or *out*. Analogously, for the next edge $\overline{C_iC_{i+1}}$ the edge stat γ_{i+1} is assigned. The pair ($\gamma_{i-1}, \gamma_{i+1}$) determines the traversal flag of the intersection vertex C_i according to the table presented before. Let us note that whether an edge is on the other polygon can be efficiently tested by checking the following condition: Two vertices, $C_i \to prev \to neighbor$ and $C_i \to neighbor$, form an edge of the other input polygon S.

Subsequently, intersection vertex with the traversal flag *none* is no longer considered intersecting. At (1), we set the traversal flags for C because the traversal flags for the other polygon can be set nearly automatically referencing its neighbor. The first intersection vertex of S will determine the rest; for example, if the first intersection vertex of C and its coincident vertex of S are respectively **ex** and **en**, then the traversal flags for the remaining vertices of S will be exclusive to C. If the first flags are the same, the traversal flags of S will be the same as C.

As seen from the data structure in Fig. 10, we have not yet treated *couple* and *cross_change* fields. The *couple* field of the current vertex on the traversal route is set to reference the closest intersection vertex that has the same single flag (i.e., **ex** or **en**) as the current vertex. Therefore, the coupled vertices behave as if they are one, satisfying the degeneracy assumption. We explain the use of the *cross_change* field by illustrating how the **en/ex** flags have been conceived.

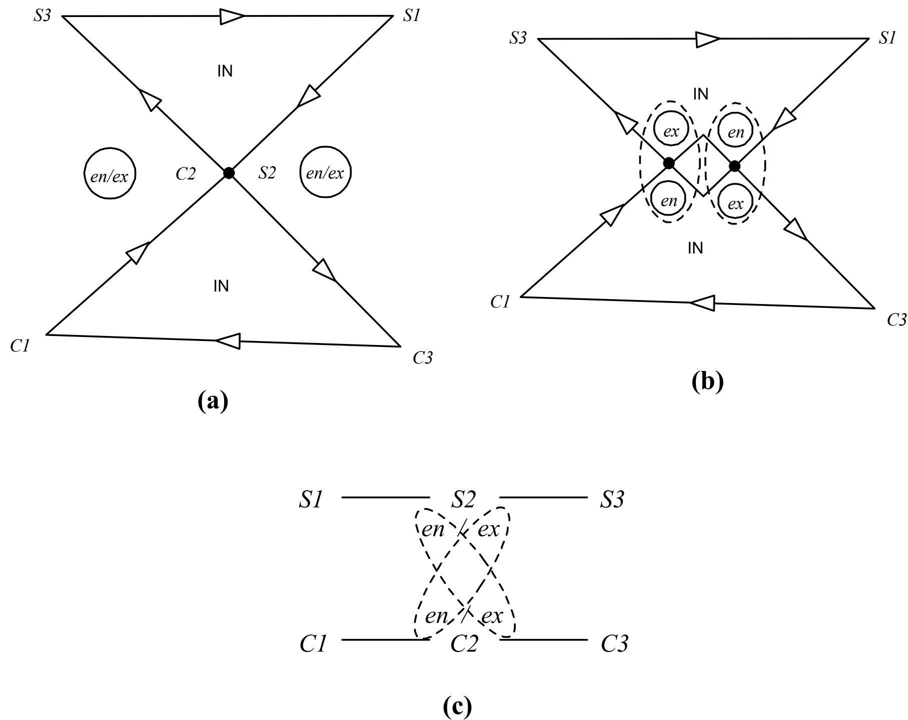

Fig. 12. Conceptualization of the en/ex flag.

The role of the **en/ex** flag shown in Fig. 12(a) has been conceptualized from Fig. 12(b); it symbolically perturbs one polygon to fix the degeneracy. However, doubly linked list for vertices, which describes only one dimension, cannot correctly convey the (**en**, **ex**) pair appearing in (b); see (c). Therefore, we need an additional flag, *cross_change* to indicate the pair. In the example of Fig. 12, *cross_change* should be set *true*. Meanwhile, when S_1 and S_3 is exchanged, *cross_change* does not need to be set. This can be determined by computing orientation of the two triangles:

$$T_1(S_{i-1}, C_{i-1}, S_{i+1}), T_2(S_{i-1}, C_{i-1}, C_{i+1})$$

When the two triangles have different orientation, *cross_change* is set *true*. Using vertex information introduced so far, we can now traverse the vertex lists, yielding clipped polygons.

4.2 Clipping

The algorithm traverses the list starting from the intersection vertex selected among those with any traversal flag. The visited vertex on the traversal deletes its flag one by one. Since a vertex can have two flags at the same time it needs to place one flag according to the traversal route to the vertex.

To select the starting vertex, we apply three following rules.

RULE 1. Once a flag of a couple has been deleted, both of the vertices can no longer be used as a starting vertex.

RULE 2. If the couple with each flag still set have (**en**, **en**), the second vertex can be selected as a starting vertex; if the couple have (**ex**, **ex**) the first vertex is selected.

RULE 3. Any intersection vertex with any traversal flag except for the above can be chosen as a starting vertex.

Jumping onto a vertex during the traversal can have four routes according to the flag of the currently visited vertex and its position within the lists, as shown in Fig. 13: (D1) To the right side of the target vertex, (D2) to the left side, (D3) from the other list to the left side, (D4) from the other list to the right side.

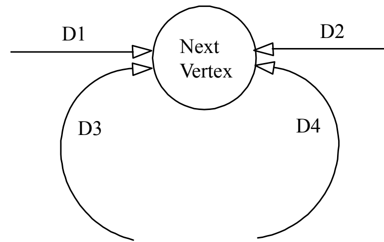

Fig. 13. Four routes to the next vertex.

According to the route chosen to reach the current vertex, delete the traversal flag(s) of the vertex. For example, suppose that a vertex with **ex/en** flag is being visited along D3 route as shown in **Fig. 14**. Then **ex** is deleted but **en** remains intact. After the deletion a new traversal status, D2, is created because the traversal is now directed to the left (i.e. the previous vertex on the list).

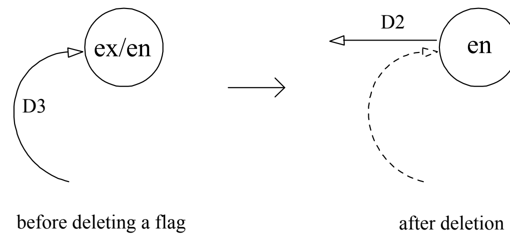

Fig. 14. Before and after deleting a flag.

When both vertices of a couple are met on one way traversal (i.e., to the right in case of (en, en) and to the left in case of (ex, ex)), the later visited vertex does not change the traversal direction. For better understanding, see traversal route on I_1 and I_2 in

Fig. **15**.

Fig. **15** demonstrates the traversal of the lists constructed from the two input polygons.

Fig. **17** shows some examples with different degenerate intersections output from our implementation.

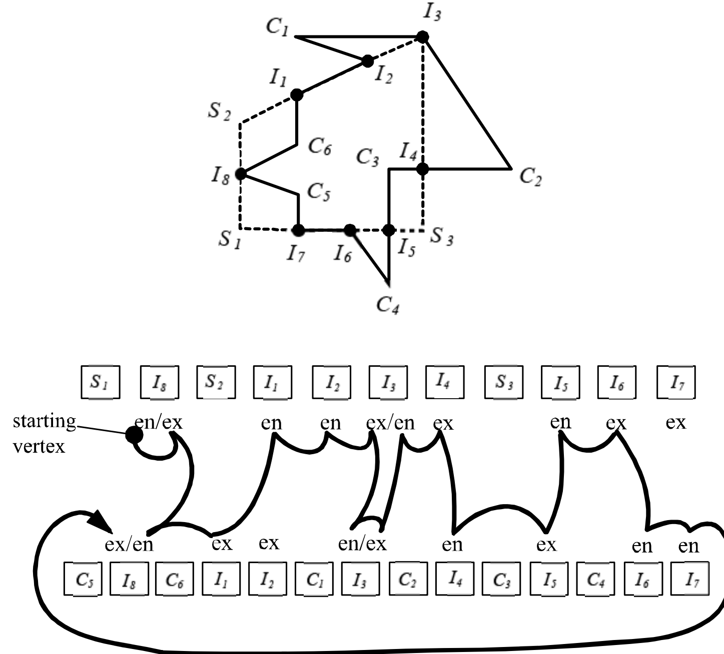

Fig. 15. An example that shows the list traversal by a curve for the operation intersection. Upper: Two input polygons after intersection computation. Lower: The traversal route is denoted by a curve.

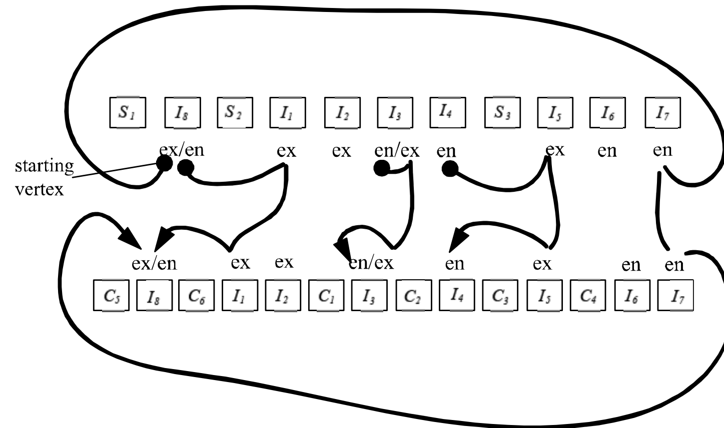

Fig. 16. For the polygons of Fig. 15, this time, subtraction operation, S - C, has been done.

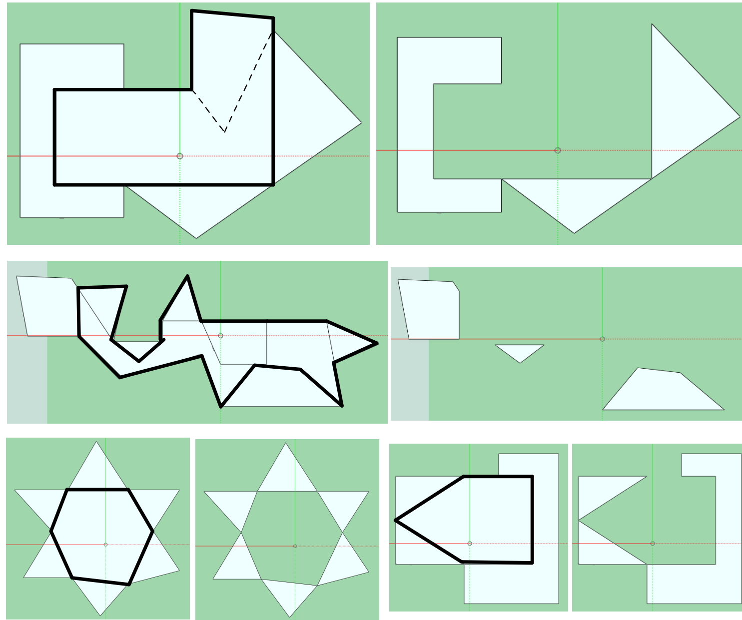

Fig. 17. Running examples of polygon clipping with degenerate cases.

5. ACKNOWLEDGEMENT

This research has been done as a part of IT Leading R&D support project, “Development of realistic virtual engineering technology” (KITA 4300-1000-1823).

6. CONCLUSIONS

We extended Greiner-Hormann algorithm to cope with the degenerate cases. For the case as in Fig. 4, it guarantees the result: if we use **en/ex** and **ex/en** flags Fig. 4(a) is obtained, otherwise Fig. 4(b) is obtained. Even without implementing a full boundary representation of [7], thus only with the vertex lists, the degenerate cases could be handled elegantly.

However, our extension shows clearly one limitation. It does not guarantee correct result in a certain situation: for example, when one self intersection point of an input polygon is crossed by the other polygon. This is mainly because the vertex data has only one neighbor pointer to the other polygon. To deal with such problem the self-intersecting polygon has to be separated into none self-intersecting polygons as has been done in [5].

To the complexity of Greiner-Hormann algorithm, our extension adds more complexity to perform inside test while setting traversal flags for each intersection vertices on one polygon. As noted in Sect.4.1, the inside test is done only for the edges that have the intersection vertices. Thus our extension outperforms Weiler's approach [6], which performs inside test for all the edges of both input polygons.

6. REFERENCES

[1] Berg, M., Kreveld, M. and Overmars, M., *Computational Geometry*, Springer-Verlag, *2000*.

[2] Greiner, G. and Hormann, K., Efficient clipping of arbitrary polygons, *ACM Transactions on Graphics*, Vol. 17, 1998, pp 71-83.

[3] Rappaport, A., An efficient algorithm for line and polygon clipping, *Visual Computer*, Vol. 7, 1991, pp 19-28.

[4] Rogers, D., *Procedural elements for computer graphics*, Mc Graw Hill, 1985.

[5] Vatti, B. R., A generic solution to polygon clipping, *Communication of ACM*, Vol. 37, No. 7, 1992, pp 56-63.

[6] Weiler, K., Polygon comparison using a graph representation, In SIGGRAPH 80, ACM, 1980, pp 10-18.

[7] Weiler, K. and Atherton, P., Hidden surface removal using polygon area sorting, In SIGGRAPH 77, 1977, pp 214-222.