30

BIT 26 (1986), 430–441

RECONSTRUCTING VISIBLE REGIONS FROM VISIBLE SEGMENTS

W. RANDOLPH FRANKLIN [1] and VAROL AKMAN

Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York 12180, USA

Department of Computer Science, University of Utrecht, Budapestlaan 6, P.O. Box 80.012, 3508 TA, Utrecht, the Netherlands

Abstract.

An algorithm is presented for reconstructing visible regions from visible edge segments in object space. This has applications in hidden surface algorithms operating on polyhedral scenes and in cartography. A special case of reconstruction can be formulated as a graph problem: "Determine the faces of a straight-edge planar graph given in terms of its edges." This is accomplished in $O(n \log n)$ time using linear space for a graph with n edges, and is worst-case optimal. The graph may have separate components but the components must not contain each other. The general problem of reconstruction is then solved by applying our algorithm to each component in the containment relation.

CR categories: I.3.5, I.3.7, F.2.2.

Keywords: hidden surface removal, polyhedra, object space, straight-edge planar graph, point-location.

1. Introduction.

The hidden surface problem in computer graphics has been an important area of study for the last two decades. For a somewhat dated but otherwise excellent survey see [19]; a bibliography is provided by Griffiths [11]. The majority of the currently used algorithms for hidden surface removal operate in *image space*, the realm of (raster) display devices. When the accuracy of the output as opposed to a particular rendering is crucial, algorithms which perform the visibility calculations at object resolution are needed. These *object space* algorithms include, but are not limited to, those by Loutrel [12], Fuchs et al. [9], Weiler and Atherton [21], Sechrest and Greenberg [18], and Franklin [1, 2]. Various analyses of the hidden surface problem from the viewpoint of computational complexity can be found in F. F. Yao [22], Schmitt [17], Ottmann and Widmayer [16], and Nurmi [15].

[1] Research of this author is supported by the National Science Foundation under grant no. ECS-8351942, and by the Schlumberger-Doll Research Labs, Ridgefield, Connecticut.

Received May 1986. Revised August 1986.

30-1

In this paper, we present an algorithm, in object space, for reconstructing the visible regions in a polyhedral scene, i.e. joining the visible edge segments output from a hidden line program to find the visible regions. Such an algorithm is used by e.g. the hidden surface algorithm of Franklin [2] as well as applications in cartography where regions are reconstructed from separately digitized boundary segments [3]. The input to the algorithm is a set of visible edge segments computed as in [2]. This will be detailed in the sequel. The regions output by the algorithm may be stored along with their intensities which must be computed for later display. Although the algorithm is based on a simple graph problem, namely, "Determine the faces of a straight-edge planar graph given in terms of its edges," this to our knowledge is the first published implementation in computer graphics. Our algorithm to solve the above problem is worst-case optimal and spends $O(n \log n)$ time and linear space for a graph with n edges. It is required that the graph has no separate components containing each other. The reconstruction is done by solving the above problem for each component in the containment relation of the graph under consideration. The algorithm has been implemented in each of Ratfor, Franz Lisp, and Prolog. The original version of this paper [7] gives the complete code in Prolog, a language well-suited for computational geometry (cf. Franklin [6], Gonzales et al. [10], and Swinson [20]).

2. Problem definition and transformation.

Given a family of polyhedra in three-dimensional Euclidean space, the following algorithm computes a hidden surface picture assuming that the viewpoint is at ∞ in the positive z-direction and orthographic projections of the polyhedra are taken in the xy-plane:

(i) Compute the intersections in the xy-plane of the projected edges of the given polyhedra to subdivide each edge into segments (hereinafter, segments). An edge with no intersections is just one segment. Each segment is completely visible or else completely hidden.
(ii) Compute the visibility status of each segment, i.e. take the midpoint of the segment and determine its status which necessarily is the segment's status.
(iii) Compute the regions in the xy-plane described by the visible segments.
(iv) For each computed region, determine the face in three-dimensional space which gave rise to it and shade the region accordingly.

Although the algorithm given in [2] is an extension of this naive algorithm and uses an adaptive grid to reduce its complexity from quadratic worst-case to linear expected time, it is conceptually the same [4, 8]. In the above algorithm, step (iii) where the polygons in the xy-plane corresponding to visible parts of faces must be found from a set of visible segments is called *polygon reconstruction* and will be the subject matter of this paper. For the upcoming discussion, it is

30-2

better to cast this problem in a more abstract setting, i.e. in terms of straight-edge planar graphs. From now on, when talking about the problem in the visibility context we shall use the terms "segment" and "region" whereas in the graph context we shall employ the terms "edge" and "face," respectively.

Let E be a set of edges in the xy-plane. It is assumed that the members of E are such that they constitute a *legal* planar subdivision, i.e. one in which every face is bounded save the outer ones which are infinite. If E is obtained as a result of the above hidden surface algorithm, then the subdivision is necessarily legal under the reasonable assumption that the polyhedral input to the hidden surface algorithm is meaningful. Notice that the notion of legality is similar to constructive solid geometry in the sense that we do not allow dangling edges (Figure 1). The *polygon reconstruction* asks for a listing of the faces of the

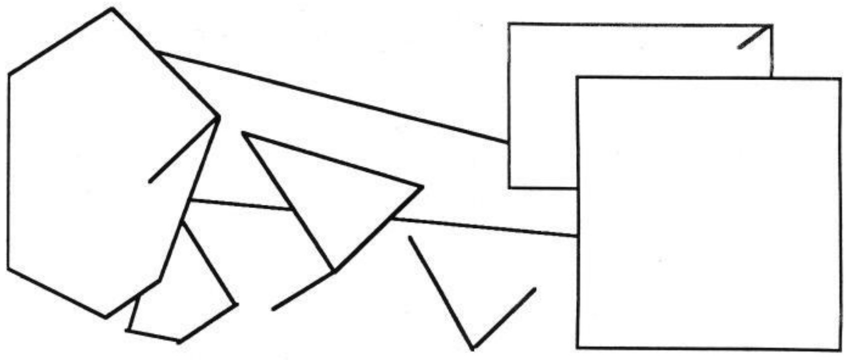
Fig. 1. Dangling edges are not allowed.

planar subdivision given in terms of its edges. By listing, we mean specifying the vertices of the faces in order (clockwise (cw) or counter-clockwise (ccw)) starting with any of them. Thus, in Figure 2(a), we are given a picture of three polyhedra in space. The visible segments have been shown in Figure 2(b). Notice that they are in no particular order, i.e. $E = \{e_1, e_2, \ldots, e_{25}\}$. In Figure 2(c) the reconstructed faces have been shown. Denoting them by F, it is seen that $F = \{f_1, f_2, \ldots, f_8\}$, excluding the two outer faces. It will shortly be clear why the vertices are labeled in Figure 2(c) in that particular way.

A crucial point in Figure 2 is that the graph has two components. In this case, the separate parts are caused by the fact that the pyramid projects individually while the cubes overlap in projection. Our algorithm can handle such separate components as long as they do not enclose each other. However, it is also possible to have separate components due to holes in the given polyhedra or due to components one of which projects inside the other. This is demonstrated in Figure 3(a) and Figure 3(b), respectively. The problem can be solved by finding the connected components of the graph using standard algorithms. As long as two connected components do not enclose each other

30-3

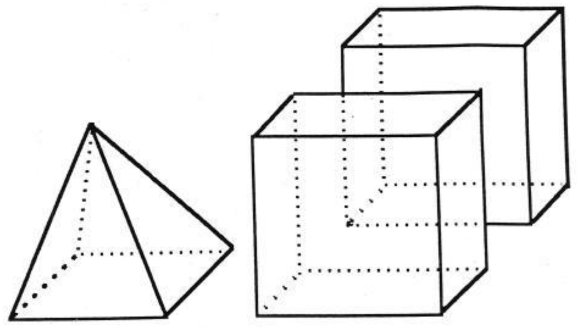

Fig. 2a.

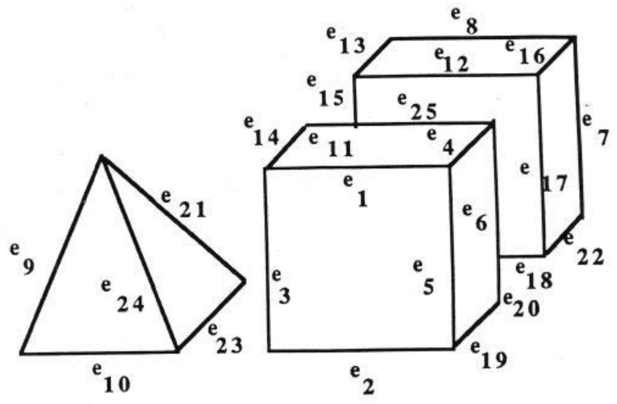

Fig. 2b.

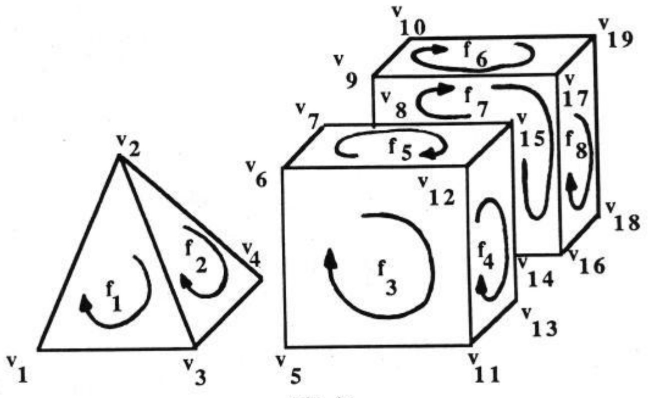

Fig. 2c.

Fig. 2. The polygon reconstruction problem.

30-4

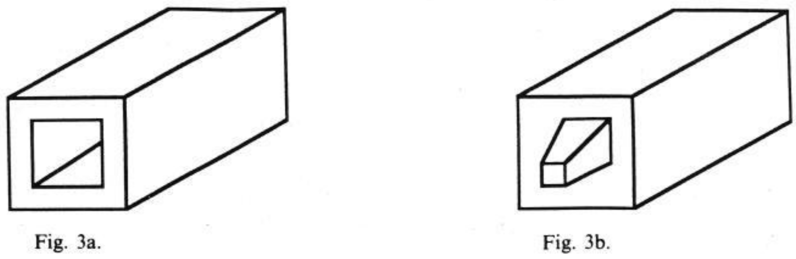
Fig. 3a. Fig. 3b.

Fig. 3. Two separate components with containment.

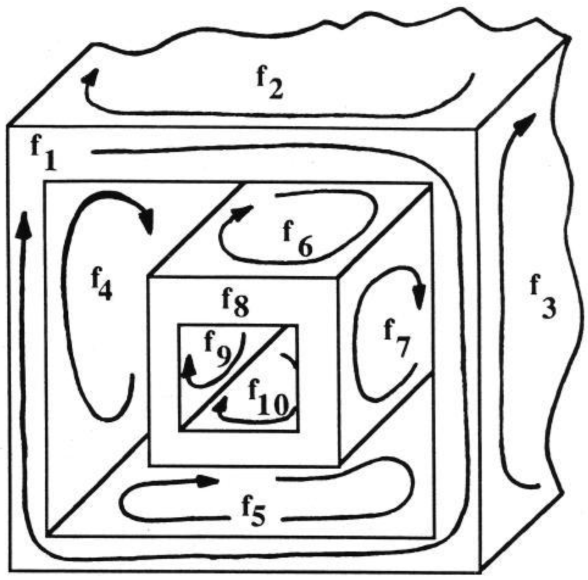

Fig. 4. A tower of separate components enclosing each other.

they can be included in the same input set to our algorithm. The question of containment can be decided by repeated applications of standard point-location algorithms. In particular, consider two separate components C_1 and C_2. If one takes any point p of C_1 and finds out that C_2 encloses p then it is seen that C_2 contains C_1. Note, however, that we are still left with the problem of specifying an order of the computed faces which must eventually be painted in a hidden surface display. Consider the faces in Figure 4. Here we have a "tower" of three connected components enclosing each other. The right approach is to start the painting with the "outer" component and progress toward the "innermost" component. In other words here we must paint the computed faces in the following order: $f_1, f_2, \ldots, f_{10}$. Within each component, the order of painting its faces can be made arbitrary. With these explanations, we assume

30-5

that the *general* problem of reconstruction is solved as soon as we give our algorithm for reconstructing a planar graph with no containment. Accordingly, in the sequel we shall assume that we are dealing with a legal straight-edge planar subdivision with no "holes." Separate components are on the other hand allowed as long as they do not violate this requirement.

Another important issue to be resolved is that of real coordinates. The fact that one is in fact dealing with numbers output by a line intersection algorithm makes it necessary that we take care of the small discrepancies introduced in the way the intersections were computed and the set of visible segments were obtained. We refer the reader to Franklin [5] for an account of such numerical errors caused by the underlying algebra. Thus, a preconditioning step prior to reconstruction is needed. This process will take as input a value δ and will output a new set of segments so that if there are vertices in the segment set within δ-neighborhood of each other then they are "reduced" to a single common vertex. The proper value of δ is dependent on the underlying numerical errors. This preconditioning process is assumed in the following description. In fact, we shall, without loss of generality, describe our polygon reconstruction algorithm assuming that the graph vertices all have integer coordinates.

3. The algorithm

The reader may follow the example in the next section while reading this section. The input to the algorithm consists of n edges in the xy-plane which are specified by their endpoint coordinates

$$E = \{((x_{i1}, y_{i1}), (x_{i2}, y_{i2})): \quad i = 1, \dots, n\}.$$

It is emphasized that no particular order is assumed in E. As a matter of fact, the operations of our algorithm can be carried out on the abstract data type "set" in an environment supporting set operations efficiently.

Given E, we first obtain the graph specified by E. To do this, we create $E'' = E \cup E'$ where E' is the "reverse" of E, that is

$$E' = \{((x_{i2}, y_{i2}), (x_{i1}, y_{i1})): ((xi_1, y_{i1}), (x_{i2}, y_{i2})) \in E\}.$$

E'' now corresponds to the edges of the *directed* graph specified by E. Next we sort E'' by the first key in lexicographic order in x and y values. Specifically, consider an element $((x1, y1), (x2, y2))$ of E''. Then the sort takes place on key $(x1, y1)$ and the lexicographic order $\leq$ is such that

$$((x1, y1), (x2, y2)) \leq ((X1, Y1), (X2, Y2)) \textit{ iff } (x1 < X1) \textit{ or } (x1 = X1 \textit{ and } y1 \leq Y1)$$

for another element $((X1, Y1), (X2, Y2))$ of E''. Now, consider the elements of

30-6

E'' with the same key. (For convenience, we assume that we renamed the vertices so that the graph is now represented by vertex names and not vertex coordinates.) These are the edges of the planar graph which have an endpoint (vertex) equal to their common key. We shall call the totality of the other endpoints (vertices) of these edges a "row" and the common vertex a "pivot" for descriptive purposes in the following algorithm. The pivot is not included in the row. Thus the planar graph can be visualized as a table made of a family of rows each having a different pivot. Finally, we sort the elements of each row about their pivots in angular order. (The angle θ of a vertex satisfies $0 \leq \theta < 2\pi$.) If the vertices are sorted in cw order then the final faces output by the algorithm will be in ccw order (save the infinite face which will be cw). If the vertices are sorted in ccw order then the final faces will be cw (with a ccw infinite face). In the sequel we assume that the latter approach is taken.

We shall refer to this final table as the "navigation" table since the algorithm to be given below will navigate through this table to obtain the faces one after another. The navigation table is similar to a date structure presented by Muller and Preparata in [13]. Associated with each row of the table we hold a counter which is initialized to the cardinality of the row in the beginning of the navigation.

```
Algorithm Navigate
; There are m rows (and thus m pivots) in the navigation table where $m = O(n)$.
; Initially all elements of the rows are marked as "unused."
; We use integers instead of the vertex names (e.g. 1 means $v_1$).
;Count field of each row has also been initialized.
CurrentPivot ← 1
; Come here after outputting a face.
; Find new pivot to start with.
L1:
IF Count(CurrentPivot) = 0 THEN
  CurrentPivot ← CurrentPivot + 1
  IF CurrentPivot > m THEN STOP ELSE GO TO L1 FI
FI
CurrentRow ← CurrentPivot
CurrentVertex ← First "unused" element of CurrentRow
Face ← {CurrentPivot} ; Initialize the face to be output.
DO FOREVER ; This loop corresponds to the navigation.
  Append CurrentVertex to Face
  Mark CurrentVertex as "used"
  Count(CurrentRow) ← Count(CurrentRow) − 1
  If CurrentVertex = CurrentPivot THEN
    OUTPUT Face ; This face finished.
    GO TO L1
```

30-7

```
    FI
    PreviousRow ← CurrentRow
    CurrentRow ← CurrentVertex
    CurrentVertex ← The element of CurrentRow following PreviousRow
      (with wrap-around)
  OD
  End
```

Correctness: The steps prior to the execution of the algorithm, i.e. building the graph, are obviously correct. The correctness of the algorithm is easy to establish under our assumption that the graph is a legal subdivision. In particular, we start with the first pivot and the correct handling of Count guarantees that the right pivots are always chosen after that. As soon as Face is output the algorithm starts anew with a partially marked navigation table. Each element in each row is marked "used" once and only once. The navigation stops as soon as there are no pivots with a nonzero Count field.

Complexity: The initial sorting of E'' to obtain the pivot vertices takes $O(n \log n)$ time. Renaming the vertices and obtaining the graph also takes this much time since we simply take each coordinate pair and via binary search find its vertex name (binary search on first coordinate followed by binary search on second coordinate). Sorting in angular order around a pivot takes $O(d_i \log d_i)$ for pivot i with degree d_i. (The degree of a pivot is the cardinality of its associated row.) An upper bound on the total time spent in this process is again $O(n \log n)$ since the sum of the degrees is $2n$. Now, consider the navigation process. Assume that we are constructing one of the faces and suppose that it will eventually have v vertices. The cost of constructing this face is then only $O(v \log n)$ since the only non-constant time operation in the logarithm is locating the first "unused" element of CurrentRow and this clearly takes $O(\log n)$ time by binary search since a row is in sorted angular order. Each vertex i of the graph will appear in exactly d_i faces, giving again a total time of $O(n \log n)$ for all the faces to be output. Thus, after the navigation table is built the algorithm uses $O(n \log n)$ time. The total space used is clearly $O(n)$.

Optimality: We shall show that the solution is worst-case optimal by demonstrating that a polygon reconstruction algorithm can be used to sort real numbers $r_1, r_2, \ldots, r_n$. This will give the lower bound $\Omega(n \log n)$ on the complexity of the problem. Assuming that the numbers are lying on the x-axis in the xy-plane the following construction is made (Figure 5): For each r_i include in the set of edges to be submitted to the algorithm the edges of the triangle A_iB_iC where $C = (0, 1)$, $A_i = (r_i - \delta, 0)$ and $B_i = (r_i + \delta, 0)$ where δ is a small positive constant. When terminated the algorithm would have output the boundary

$$A'_1 B'_1 C A'_2 B'_2 C \cdots A'_n B'_n C$$

30-8

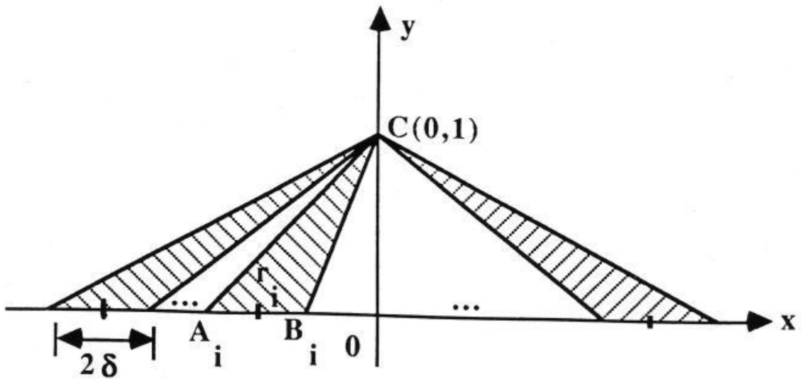

Fig. 5. Sorting via polygon reconstruction.

although not necessarily in this order where A_i' and B_i' are associated with the ith smallest r_j. From this boundary polygon one can infer in linear time the ascending sorted order of the given numbers. Note that in this proof, we have assumed that the polygon reconstruction algorithm always outputs the boundary of the infinite face of the subdivision. Clearly, this boundary is not necessitated and one can argue that there may exist an algorithm which *only* outputs the proper faces. However, even an algorithm which does not output the infinite face would require $O(n \log n)$ time since one can easily build a single polygon ("thicker" at C) from the triangles in Figure 5.

Before closing this section we shall briefly treat the problem of shading the regions obtained by the algorithm above. Consider any region f found by the algorithm. Let p be an interior point of f. Notice that since there may be separate components enclosing each other, p must be taken very close to the boundary of f to guarantee a correct picture. Compare p against the projections of all faces. The algorithm in [2] does this much more efficiently via adaptive grid but this is not the issue here. Let F be the closest face whose projection contains p. Then f corresponds to a visible part of F and should be given an appropriate shading value, e.g. proportional to the surface normal of F. If p is not on any face then it corresponds to the background and is given the default shading value.

4. An example.

We now give an example to clarify how the algorithm works. Consider the graph shown in Figure 6. The input is

$$E = \{((2,4),(3,4)), ((3,2),(3,4)), ((4,2),(3,4)), ((3,2),(4,2)), ((3,2),(2,2)), ((2,2),(2,4)), ((1,3),(2,4)), ((2,1),(3,1)), ((2,1),(2,2)), ((3,2),(3,1)), ((1,3),(2,2))\}.$$

30-9

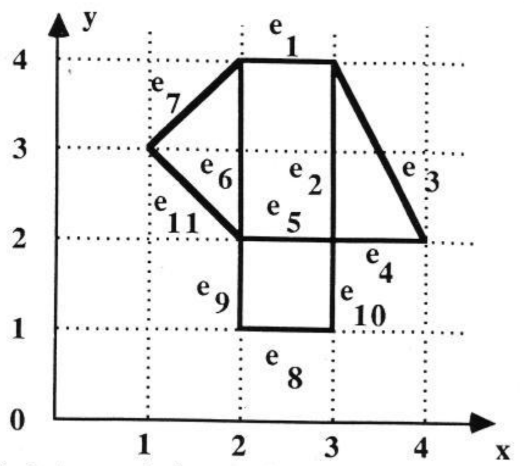

Fig. 6. An example planar graph to illustrate the algorithm.

After "reversing" E to get E' and uniting these two we get E''. After lexicographic sorting we obtain the new E'' as

$((1, 3), (2, 4)), ((1, 3), (2, 2)),$
$((2, 1), (3, 1)), ((2, 1), (2, 2)),$
$((2, 2), (2, 4)), ((2, 2), (3, 2)), ((2, 2), (2, 1)), ((2, 2), (1, 3)),$
$((2, 4), (3, 4)), ((2, 4), (2, 2)), ((2, 4), (1, 3)),$
$((3, 1), (2, 1)), ((3, 1), (3, 2)),$
$((3, 2), (3, 4)), ((3, 2), (4, 2)), ((3, 2), (2, 2)), ((3, 2), (3, 1)),$
$((3, 4), (2, 4)), ((3, 4), (3, 2)), ((3, 4), (4, 2)),$
$((4, 2), (3, 4)), ((4, 2), (3, 2)).$

Note that we have here omitted the set signs and listed E'' in such a way that the elements in each line have the same pivot. Now we can rename the graph vertices, i.e. deal with names instead of coordinates. The renaming is as follows:

$$v_1 = (1, 3);\ v_2 = (2, 1);\ v_3 = (2, 2);\ v_4 = (2, 4);$$
$$v_5 = (3, 1);\ v_6 = (3, 2);\ v_7 = (3, 4);\ v_8 = (4, 2).$$

After angular sorting of the rows about the pivots we get the navigation table as follows:

v_1:	v_4, v_3
v_2:	v_5, v_3
v_3:	v_6, v_4, v_1, v_2
v_4:	v_7, v_1, v_3
v_5:	v_6, v_2
v_6:	v_8, v_7, v_3, v_5
v_7:	v_4, v_6, v_8
v_8:	v_7, v_6.

30-10

In the table above the pivots are the first elements in each row before the column sign. It is emphasized that there is a wrap-around for each row, e.g. in row 7 it is implicit that v_4 follows v_8. The regions created by the navigation algorithm are then

$$\begin{aligned}
f_1 &= (v_1, v_4, v_3)\\
f_2 &= (v_1, v_3, v_2, v_5, v_6, v_8, v_7, v_4)\\
f_3 &= (v_2, v_3, v_6, v_5)\\
f_4 &= (v_3, v_4, v_7, v_6)\\
f_5 &= (v_6, v_7, v_8).
\end{aligned}$$

Note that f_2 is the boundary of the infinite face. Also note that all proper faces are in cw order whereas f_2 is ccw.

5. Summary and extensions.

We presented an optimal algorithm for reconstructing visible regions from a set of visible segments output by a hidden line program. This is an important operation in object space hidden surface algorithms operating on polyhedral scenes and in cartography.

As pointed out to us by one of the referees, the problem can also be solved using a plane-sweep algorithm such as the region-finding algorithm of Nievergelt and Preparata [14]. Our algorithm is simpler and it is conceivable to generalize it to take into account dangling edges. All one needs is to recognize the "dangling vertices," i.e. vertices with Count equal to 1 in Algorithm Navigate. Similarly, it can be made to work by a suitable extension in the context of holes and enclosing components. This would require filtering the ccw faces. We leave these rather straightforward enhancements to the interested reader.

Acknowledgement.

We thank the referees for their perceptive remarks and suggestions which improved the content and the presentation considerably.

REFERENCES

1. W. R. Franklin, *Combinatorics of hidden surface algorithms*, Ph.D. dissertation and report TR-12-78, Harvard University, Center for Research in Computing Technology, Cambridge, Mass. (1978).
2. W. R. Franklin, *A linear time exact hidden surface algorithm, ACM Computer Graphics* 14(3), pp. 117–123 (1980).
3. W. R. Franklin, *A simplified map overlap algorithm, Proc. Harvard Computer Graphics Conf.*, Cambridge, Mass., (1983).
4. W. R. Franklin, *Adaptive grids for geometric operations, Proc. Sixth International Symp. on Automated Cartography* 2, pp. 230–239, Ottawa, Canada, (1983).

30-11

5. W. R. Franklin, *Cartographic errors symptomatic of underlying algebra problems, Proc. International Symp. on Spatial Data Handling* 1, pp. 190–208, Zurich, Switzerland, (1984).
6. W. R. Franklin, *Computational geometry and Prolog*, pp. 737–749, in *Fundamental Algorithms for Computer Graphics*,' ed. R. A. Earnshaw, Springer-Verlag, Heidelberg (1985).
7. W. R. Franklin and V. Akman, *Reconstructing visible regions from visible segments*, University of Utrecht, Dept. of Computer Science, report RUU-CS-86-5, Utrecht, the Netherlands (March 1986).
8. W. R. Franklin and V. Akman, *Adaptive grid for polyhedral visibility in object space: An implementation*, University of Utrecht, Dept. of Computer Science, report RUU-CS-86-4, Utrecht, the Netherlands (March 1986).
9. H. Fuchs, Z. M. Kedem and B. F. Naylor, *On visible surface generation by a priori tree structures, ACM Computer Graphics* 14, pp. 124–133 (1980).
10. J. C. Gonzales, M. H. Williams and I. E. Aitchison, *Evaluation of the effectiveness of Prolog for a CAD application. IEEE Computer Graphics and Applications* 4(3), pp. 67–75 (1984).
11. J. G. Griffiths, *Bibliography of hidden line and hidden surface algorithms, Computer Aided Design* 10(3), pp. 203–206 (1979).
12. P. P. Loutrel, *A solution to the hidden line problem for computer drawn polyhedra, IEEE Transactions on Computers* 19(3), pp. 205–213 (1970).
13. D. E. Muller and F. P. Preparata, *Finding the intersection of two convex polyhedra, Theoretical Computer Science* 7, pp. 217–236 (1978).
14. J. Nievergelt and F. P. Preparata, *Plane-sweep algorithms for intersecting geometric figures, Communications of the ACM* 25(10), pp. 739–747 (October 1982).
15. O. Nurmi, *A fast line-sweep algorithm for hidden line elimination, BIT* 25, pp. 466–472 (1985).
16. T. Ottmann and P. Widmayer, *Solving visibility problems by using skeleton structures, Proc. 11th Symp. on the Mathematical Foundations of Computer Science*, pp. 459–470, Springer-Verlag, (1984).
17. A. Schmitt, *Time and space bounds for hidden line and hidden surface algorithms, Proc. Eurographics-81*, North-Holland, Amsterdam, (1981).
18. S. Sechrest and D. P. Greenberg, *A visible polygon reconstruction algorithm, ACM Transactions on Graphics* 1(1), pp. 25–42 (1982).
19. I. E. Sutherland, R. F. Sproull and R. Schumacker, *A characterization of ten hidden surface algorithms, ACM Computing Surveys* 6(1), pp. 1–55 (1974).
20. P. S. G. Swinson, *Logic programming: A computing tool for the architect of the future, Computer Aided Design* 14(2), pp. 97–104 (1982).
21. K. Weiler and P. Atherton, *Hidden surface removal using polygon area sorting, ACM Computer Graphics* 11(2), pp. 214–222 (1977).
22. F. F. Yao, *On the priority approach to hidden surface algorithms, Proc. 21st Annual IEEE Symp. on the Foundations of Computer Science*, pp. 301–307, Potsdam, NY, (1980).

30-12