NOTE: An Algorithm for Filling Regions on Graphics Display Devices

JEFFREY M. LANE
Calma Corporation
ROBERT MAGEDSON and MICHAEL RARICK
Boeing Corporation

The display of shaded polygons, either by lines, crosshatching, dots, or continuous color patterns is a task frequently used in many application areas of computer graphics. In applications such as the production of hidden surface images, cartography, and animation, algorithm efficiency is important. In this paper, a new fill algorithm is presented which will fill an arbitrary polygon in the plane. No sorting is required and the algorithm is suitable for microprocessor or hardware implementation.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*display algorithms, viewing algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representation*; I.3.6 [**Computer Graphics**]: Methodology and Techniques; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

General Terms: Algorithms

Additional Key Words and Phrases: Polygons, shading, fill algorithms

1. INTRODUCTION

The filling or shading of a polygon is a common task in many applications of computer graphics. On vector (stroke) devices, line, cross-hatch, and dot patterns are often used as filler. On raster devices, a constant color or continuous (linear) blend of color is often used as the fill medium. Applications exist in cartography, drafting, simulation, and synthetic imagery. This paper describes an algorithm which is oriented toward raster graphics devices, although the only required capability is to be able to "erase" previously filled areas. The algorithm handles non self-intersecting polygons including those with "holes." Explicitly, we define a polygon as a set of sequences of outline coordinates (x_1, y_1), $(x_2, y_2, \ldots, (x_n, y_n)$, the orientation of which determines whether the particular outline is considered a solid (counterclockwise) or a "hole" (clockwise).

There exist many polygon fill algorithms in the literature. These algorithms assume a polygon is convex, employ a parity check capability, [1, 5] or decompose

Authors' addresses: J. M. Lane, Vulcan Software Inc., 93 South Central Ave., Campbell, CA 95008; R. Magedson and M. Rarick, Boeing Corporation, Mail Stop 35-02, P.O. Box 3707, Seattle WA 98124.

the polygon into simpler shapes, usually trapezoids or triangles [2–4, 6]. The decomposition algorithms and the parity check algorithms each require the polygon to be multiply sorted before processing. This sorting can be time consuming and, in the case of many parity check algorithms, can require a good deal of memory [1, 5].

The fill "material" or pattern used depends on the application and type of graphic device being used. Typically, for vector (stroke) devices, the fill pattern consists of parallel sets of solid and dashed lines [2]. For raster devices solid color fill and half-tone techniques [8, 9] are additional possibilities. Drafting and mapping applications often make use of the crosshatching technique [2], while color fill and half-toning are used in animation and synthetic imagery [4].

The algorithm presented here is new, requires no sorting, handles nonconvex, non self-intersecting polygons properly, and is optimal (in the number of pixel updates) for convex polygons. Treating each edge (x_i, y_i), (x_{i+1}, y_{i+1}) in succession, the algorithm fills each triangle corresponding to that edge and the initial vertex (x_1, y_1) of the polygon. The fill pattern is written if the triangle is counterclockwise oriented, otherwise it is subtracted. Below we give a more detailed description and proof of the algorithm.

2. NEW ALGORITHM

We define a polygon as a set of sequences of points $(x_i, y_i)_1^n$ where the orientation of the individual outlines determines whether the polygon is considered solid (counterclockwise—material to the inside) or hole (clockwise—material to the outside) (see Figure 1). For each outline or boundary, and for each edge $((x_i, y_i), (x_{i+1}), (x_{i+1}), y_{i+1}))_2^{n-1}$, the triangle consisting of that edge and the initial vertex $(x(1), y(1))$ of the polygon is filled. The fill pattern is added to the image if the triangle is clockwise oriented, otherwise the fill pattern is subtracted from the image, and care is taken to fill each edge between triangles only once.

Triangle filling can be optimized to take advantage of the obvious facts that triangles are convex and have only three sides, and are, therefore, good candidates for hardware implementation. There is a considerable amount of literature on filling triangles [9–11], and we do not explicitly address that problem.

A slight restriction on the algorithm occurs when we need to subtract as well as add patterns to the image. All CRT raster devices have this capability, and the algorithm can be made to work on CRT stroke devices which have good repeatability. However, the algorithm is also applicable in any situation where the image is buffered before display, or in the case where all polygons are convex, since no pattern will be subtracted in that case.

3. ANALYSIS

The proof that the filling algorithm produces the proper filled region is quite straightforward. First we note the obvious similarity between filling a polygon and computing the area of that polygon, unit by unit. Now the Jordan-Brower Separation Theorem (or "ray casting") can be used to show the well-known fact that the area of a polygon can be written as the sum of the signed areas of the triangles constructed above. The sign for any particular triangle will, as before,

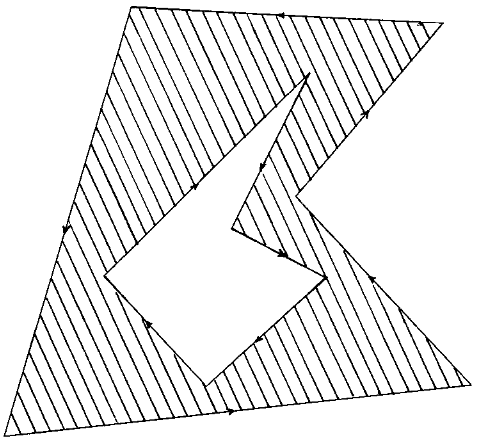
Fig. 1. Outside polygon is counterclockwise oriented, thus material is to the inside. The inside polygon is clockwise oriented, thus material is to the outside of this polygon.

be positive if that triangle is oriented counterclockwise, and negative otherwise. Adding or subtracting the triangle area corresponds to adding or subtracting the filled triangles in our algorithm.

Note that for convex polygons, oriented counterclockwise, the algorithm remains optimal, since each image unit within the polygon is filled only once because no negative area triangles occur. The area of the polygon is given by

$$\text{area} = \frac{1}{2} \sum_{i=2}^{n-1} ((x_i - x_1)(y_{i+1} - y_1) - (x_{i+1} - x_1)(y_i - y_1)).$$

Further, each of the terms (triangle areas) of the sum is positive, and the amount of computation involved in our algorithm is proportional to the area as computed using a similar triangle area summation technique. However, for nonconvex, counterclockwise oriented polygons, some of the internal terms may be negative; therefore an upper bound to computation costs is given by the sum of the unsigned triangle areas:

$$\frac{1}{2} \sum_{i=2}^{n-1} |(x_i - x_1)(y_{i+1} - y_1) - (x_{i+1} - x_1)(y_i - y_1)|.$$

4. CONCLUSIONS

A polygon fill algorithm has been presented which is optimal in the number of pixel updates required for convex polygons, and eliminates the sorting steps required for the parity check and decomposition techniques that are normally employed when nonconvex polygons are part of the data base. The algorithm requires little storage above that for the polygon, although an "erasable" image buffer is required to fill nonconvex polygons properly.

The primary advantage of the algorithm is its simplicity, which makes it easy to put into hardware, and to take advantage of triangle filling hardware already extant. The simplicity of the algorithm derives from its correspondence to polygon area calculation. In this light we outline the following variant of this algorithm (also observed by one of the reviewers): rather than tile triangles we can instead tile trapezoids, each of which is formed from an edge of the polygon and a fixed vertical or horizontal line, traversing the polygon edges in order. Such trapezoids have all vertical and horizontal edges except one, and can take advantage of the high speed horizontal and vertical line generation of many hardware devices. Although this approach is not optimal in terms of the number of pixel updates required for convex polygons, it does have the advantage that no internal edges need to be created and followed. This approach is indeed faster in many cases where a polygon has a large number of short edges, a large area, and an extremely rough boundary, as with a continent outline. We recommend this variant when such data are present, or when trapezoid filling hardware is available.

The algorithm as originally presented should prove useful in applications where the majority of the polygons processed are convex or "nearly" convex, since the algorithm is optimal in terms of the number of pixel updates for these cases and can handle nonconvex polygons as well. Boundary representations for solid models typically have faces that do not deviate severely from convexity, and the algorithm presented in this paper was developed in conjunction with a hidden surface algorithm to display images of solid models. Further, the boundary polygons are determined on the fly, and thus no prior knowledge of convexity is available before the filling process begins. It should be noted that for particularly oscillating polygons many "adds" of fill at a pixel may occur before the corresponding "subtracts" catch up, and an overflow of an integer-valued intensity can occur. If this situation is common, a separate polygon mask buffer must be kept, as is the case in many parity check algorithms [1, 5]. However, this buffer need not be large, since if we fill each triangle in scanline order, the only buffer needed is an array representing the pixel locations on a scanline. This scanline approach to the algorithm was combined with a scanline hidden surface algorithm to make the images of the solids in Figure 2.

ACKNOWLEDGMENT

The authors would like to thank the *TOG* referees for their valuable insights and suggestions.

REFERENCES

1. ACKLAND, B. D., AND WESTE, N. H. The edge flag algorithm—A fill method for raster scan displays. *IEEE Trans. Comput. C-30*, 1 (Jan. 1981), 762–791.
2. BRASSEL, K. E., AND FEGEAS, R. An algorithm for shading regions on vector display devices. In *Proc. Siggraph '79* (Chicago, Ill., August 6–10), ACM, New York, pp. 126–133.
3. CHAZELLE, B., AND DOBKIN, D. Decomposing a polygon into its convex parts. In *Proc. Eleventh ACM Symposium on the Theory of Computing* (Atlanta, Ga., April 30–May 2, 1979), ACM, New York, pp. 38–48.
4. LEWIS, B. A., AND ROBINSON, J. S. Triangulation of planar regions with applications. *Comput. J. 21*, 4 (Jan. 1978), 324–332.
5. PAVLIDIS, T. Filling algorithms for raster graphics. *Comput. Gr. Image Process. 10* (1979), 126–141.
6. SCHACHTER, B. Decomposition of polygons into convex sets. *IEEE Trans. Comput. C-27*, 11 (Nov. 1978), 1078–1082.
7. GOURAUD H. Computer display of curved surfaces. *IEEE Trans. Comput. C-20*, 6 (June 1971), 623–628.

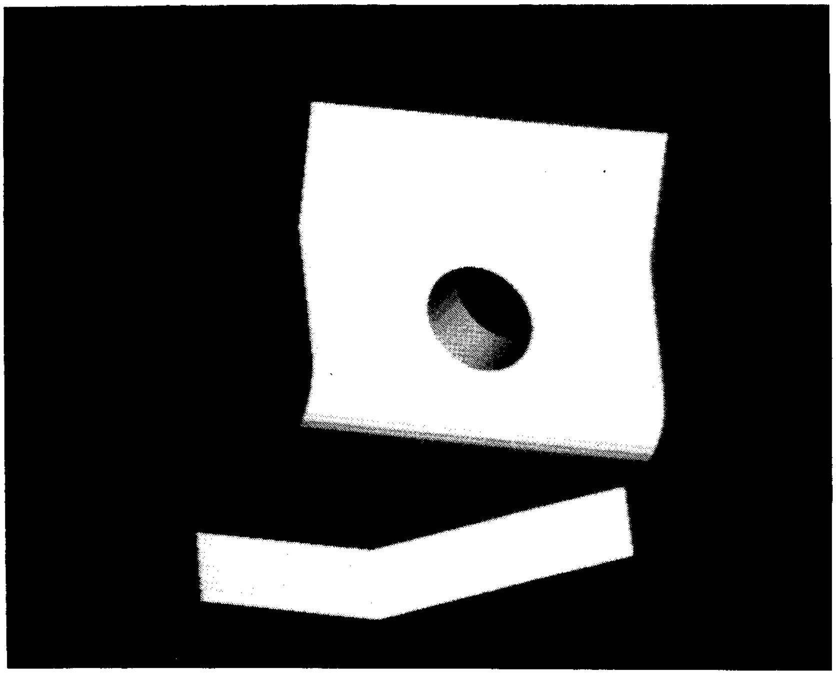

Fig. 2. Solid bracket.

8. JARVIS, J. F., JUDICE, C. N., AND NINKE, W. H. A survey of techniques for the display of continuous tone pictures on bilevel displays. *Comput. Gr. Image Process. 5* (1976), 13–40.
9. MYERS, A. An efficient algorithm for computer generated pictures. Ohio State University Computer Graphics Research Group document, 1975.
10. BOURKNIGHT, W. J. A procedure for generation of three-dimensional half-toned computer graphics representations. *Commun ACM 13*, 9 (Sept. 1970), 527–536.
11. SHAMOS, M. I. *Computational Geometry*. Ph.D. dissertation, Yale University, Dept. of Computer Science, New Haven, Conn., 1978.

Received August 1982; revised July 1983; accepted August 1983