Set theoretic operations on polygons using the scan-grid approach

L M Patnaik*, R S Shenoy† and D Krishnan*

This paper describes an algorithm to compute the union, intersection and difference of two polygons using a scan-grid approach. Basically, in this method, the screen is divided into cells and the algorithm is applied to each cell in turn. The output from all the cells is integrated to yield a representation of the output polygon. In most cells, no computation is required and thus the algorithm is a fast one. The algorithm has been implemented for polygons but can be extended to polyhedra as well. The algorithm is shown to take $O(N)$ time in the average case where $N$ is the total number of edges of the two input polygons.

The problem of computing the union, intersection and difference of polygons and polyhedra finds an important application in CAD work. Examples of algorithms to carry out these operations can be found in the literature. For example see Sutherland and Barton and Buchanan. Bentley and Wood describe an algorithm to report intersections of rectangles. Nievergelt and Preparata discuss a plane sweep algorithm which computes $s$ intersections of two convex polygons in $O(n \log n + s)$ time, where $n$ is the total number of edges of the two polygons. A grid cell approach has been employed by Tilove to compute the wireframe representations of solids represented using constructive solid geometry (CSG). In this paper, we use this approach to compute the result of performing a set theoretic operation on two polygons. As the algorithm is implemented on a line drawing display, it is assumed in the following that:

- the input is a set of edges and vertices of the input polygons
- the output from the algorithm is a set of edges and vertices of the output polygon

An edge is defined as a line joining its starting and ending points where the points are referred in a clockwise manner. Thus, viewing the edge from the starting point, the polygon's inside is to the right of the edge.

BASIC IDEAS BEHIND THE ALGORITHM

The algorithm operates in the image space. Initially, the screen is divided into a number of cells by overlaying the scene with an $n \times n$ grid. Computations are performed on each cell to decide the output edge segments from the cell and finally all these outputs are integrated to produce the complete output polygon. The algorithm scans the grid on a cell-by-cell basis to determine the output and hence the name scan-grid. The input polygons to the algorithm can be convex, concave or self-intersecting in nature.

Consider Figure 1 which shows two intersecting polygons $A$ and $B$ overlaid by a $4 \times 4$ grid. When computing $A \cap B$, we need consider only the cells where both the polygons are present namely, 6, 7, 10 and 11. Thus, the other cells need not be considered at all. Similarly for $A-B$ we need consider only the cells where $A$ is present (ie 2, 3, 6, 7, 10 and 11) whereas for $A \cup B$, all cells excepting 1, 4, 5, 8, 13 and 16 (ie the nonempty cells) are considered. Note that empty cells are not considered regardless of what the operation is.

There are mainly two situations encountered in each cell. These are:

- only one polygon exists in the cell
- both polygons exist in the cell

**Situation 1**

Whenever only one polygon exists in the cell, the segments of the edges of the polygon, which are in the cell, are all either a part of the output or are not so. For example, in cell 2 only polygon $A$ is present. The part of $A$ that is in the cell is part of the output when computing $A \cup B$ or $A-B$. When computing $A \cap B$ or $B-A$, nothing within the cell is part of the output.

![Figure 1. Scene overlaid by a 4 x 4 grid](image)
Situation 2

Whenever both polygons exist in the cell there are three possible cases. They are the following:

- the cell lies wholly within the boundaries of one of the polygons
- the cell lies wholly within the boundaries of both the polygons
- the cell contains a part of the edges of both the polygons

Consider Figure 2(a) which shows a single cell when the first case is true (i.e., it lies wholly within A). Here A is called the containing polygon. In this case when computing $A \cap B$, the segments $e_1$ and $e_2$ are obviously a part of the output. For $A - B$ also, $e_1$ and $e_2$ are part of the output. Note that if we are shading the polygons, then $A \cap B$ corresponds to the crosshatched region of $A - B$ to the shaded portion. In either case, however, $e_1$ and $e_2$ are boundaries of the output region. In a line drawing display (or when no shading is done), it is therefore sufficient to merely include $e_1$ and $e_2$ in the output. As can be seen, when computing $A \cup B$ or $B - A$, there is no output from the cell.

For a cell in which the second case is true, there is no output from the cell for any set theoretic operation.

If the third case is true and there are no intersecting edges within the cell (i.e., the polygons are disjoint within the cell), then for $A \cup B$ all the edge segments of both the polygons form a part of the output. For $A \cap B$, there is no output from the cell and for $A - B$ and $B - A$, the edge segments of the appropriate polygon are output. Figure 2(b) illustrates the more complicated case when intersecting edges exist. The edge segments to be output are decided depending on the operation to be performed. Thus the intersection point(s) X should be known. Then for $A \cap B$, the output will be $s_1$ and $s_2$. In case of $A \cup B$, the output will be $s_1$, $s_2$, and $s_4$ and for $A - B$ it is $s_{11}$, $s_2$, and $s_4$.

Performing the procedure outlined above to each cell in turn, a set of edges and segments are obtained. These describe the output polygon.

**DETAILED DESCRIPTION**

The algorithm as implemented in this work operates slightly differently from the procedure described above. While performing the previous section's procedure to each cell, three types of edges are encountered within a cell:

- **visible edges**
  - if an edge or a segment of the edge within the cell forms a part of the output, then the edge is classified as visible
- **invisible edges**
  - if an edge or a segment of the edge within the cell does not form a part of the output, then the edge is classified as invisible
- **intersecting edges**
  - in our implementation of the algorithm, the intersecting edges are classified as invisible if the intersection point lies within the cell else they are classified as visible

Thus three lists called Visible, Invisible and Intersections are maintained. The following procedure is adopted to calculate the intersections of each edge of one polygon with every edge of the other polygon present in the cell (the Intersections list is looked up first, to see if it is already calculated). A cell normally contains only one or two edges. The average number of edges within a cell is obviously a function of the cell size. Define the cell size, $C$, as $C = aL$ where $L$ is the average length of an edge and $a$ is an arbitrary constant, $0 < a < 1$. Now, we can intuitively expect to find no more than one or two edges per cell, because the polygon has been scaled to fill the entire screen. The constant $a$ can be varied to obtain best results. Thus calculating intersections is not very difficult. For each edge, the intersection points are stored in Intersections. Note that if a vertex of one polygon lies on one of the edges of the other, two intersections are recorded, if both the edges belonging to that vertex lie on the same side of the edge of the polygon. Only one intersection is recorded if the edges of the vertex lie on different sides of the edge of the polygon as shown in Figure 3.

In the case of overlapping edges, a different procedure is followed, and this is discussed later. If an edge of one polygon does not intersect with any edge of the other polygon within the cell, then it is totally inside or totally outside the other polygon. If it is outside, apply the criteria used when only one polygon exists in the cell. If it is inside, then apply the criteria used when a containing polygon exists within the cell. (A faster way when there are few edges is to look up the Visible and Invisible lists first. If the edge is present in either, go to the next edge, otherwise perform the above mentioned steps.)

After executing the above procedure for each cell, the following steps are performed to arrive at the complete output:
Figure 3. Case where a vertex of one polygon lies on the edge of the other polygon (a) intersection point X is recorded once, and (b) intersection point X is recorded twice

- Any edge belonging to Visible that is not in Invisible is added to the output.
- Any edge in Invisible that is not in Visible is discarded.
- For edges that are in both the lists i.e. the intersecting edges, the intersection points are sorted in increasing order of parameter T (the edges are stored parametrically). If there is only one intersection point, the edge is divided into two segments — one within the other polygon and the other outside. Depending on the operation, the following criteria are applied: for union, the segment outside is added to the output; for intersection, the segment inside is added to the output; and for B-A, the segment inside B is added to the output. When there is more than one intersection point for an edge, the alternate segments are inside and outside the other polygon. Procedures similar to Tilove's classification and combine procedures are used to obtain the segments comprising the output.

IMPLEMENTATION AND ANALYSIS OF THE ALGORITHM

Implementation

The algorithm has been implemented on a DEC 10/90 system using the Pascal programming language. A Tektronix storage tube display was used for obtaining the output. The algorithm is presented in a high level pseudocode in Appendix 1.

Analysis

To analyse the algorithm we take the three steps it contains separately and analyse each one in turn. The three steps involved are:

- Step 1: calculation of intersections of edges with the grid cells
- Step 2: calculation of intersection of edges within the same cell and
- Step 3: sorting of the intersections of those edges that are in both Visible and Invisible lists

The operations performed by this algorithm are regularized. Regularization refers to the closure of set theoretic operations in the 2D Euclidean space i.e. the union, intersection or difference of two polygons should yield a polygon with no isolated vertices or dangling edges. (It is assumed that the input polygons are also free from such anomalies.) For example, in the case of overlapping edges (see Figure 4(a)) there is the problem of regularization. The intersection of polygons ABCD and XYZ yields a dangling edge XY, when in fact, the intersection should yield a null set. To overcome this problem, the overlapping edges are defined to be not intersecting each other and each edge lies within the polygon when the union operation is computed. Hence, when A ∪ B is to be computed, two intersection points, one at X and one at Y, are generated (see Figure 4(b)). The output is shown in Figure 4(c). For intersection and difference operations, the edges are defined to be intersecting only at the end points and the edges are defined to be lying outside the other polygon (see Figure 4(d)). This produces the null set for intersections and either polygon in the case of difference operations.

A similar procedure is applied when the polygons share a vertex or vertices. Consider the case when two polygons A and B have a common vertex X. If the union of these polygons is to be computed, then the vertex X for polygon A is defined to be inside polygon B while for B the vertex is as originally defined. For intersection and difference, the vertex X of polygon A is defined to be outside polygon B. This procedure sees to the fact that no isolated vertices are generated as a result of these operations.
Assume that there are a total of $N$ edges ($N/2$ belonging to both polygon 1 and polygon 2). Assume that the side of each grid cell $C$ is given by $C = aL$ where $L$ is the average length of an edge and $a$ is an arbitrary constant, $0 < a < 1$. The constant $a$ is related to the cell size and hence to the total number of cells $M$, where $M = \frac{screen~size}{aL}$. $Nc$, the expected number of cells an edge will lie in, is given by $Nc = \frac{L}{C} + 1$ assuming the edge is parallel to the grid. Thus $Nc = \frac{L}{aL} + 1 = \frac{1}{a} + 1$. Hence the total number of grid cell boundaries which are intersected will be $Nc - 1 = \frac{1}{a}$. Therefore, the number of computations required to compute Step 1 will be of the order of $N \times \frac{1}{a} = O(N)$.

Since the polygons have been scaled to fill the entire screen, the average number of edges per cell will be one or two. Thus the intersections to be calculated per cell will be very small and can be assumed to be constant. Hence the number of computations for Step 2 will depend on the number of cells containing both the polygons. In the worst case, this will be $M$ and since $M$ is assumed to be constant, then the computation time will be $O(N)$.

In the worst case, the line of intersection is linear in the number of input edges. The algorithm covers a domain which includes convex, concave and self-intersecting polygons. Another useful feature of the algorithm is that the output can contain shading and texture information. This feature would be of use when dealing with polyhedra.

CONCLUSIONS

An algorithm has been presented to perform set theoretic operations on polygons. The extension of this algorithm to polyhedra has also been discussed. In an average case, the complexity of the algorithm is linear in the number of input edges. The algorithm covers a domain which includes convex, concave and self-intersecting polygons. Another useful feature of the algorithm is that the output can contain shading and texture information. This feature would be of use when dealing with polyhedra.

REFERENCES


APPENDIX 1: ALGORITHM IN PSEUDOCODE

Step 0

Read in the two polygon edges. Scale the picture so that it fills the entire screen. Overlay the grid.

Initialize all the data structures.

Step 1

for each polygon do

for each edge do

...
Calculate intersections with grid cells.
Associate the edge with the grid cell
if it intersects the cell.
end
end

Step 2
for each grid cell do
if the grid cell is not empty then
if the grid cell contains edges belonging to only
polygon then
if the cell lies wholly within the other polygon
then
apply the criteria corresponding to case(1) of
situation 2 and add edges to Visible or
Invisible.
else
apply criteria for situation 1.
end if
else
for each edge in the cell do
find intersections.
end
end
end

Step 3
for each edge in Visible do
if edge not in Invisible then include edge in output.
end
for each edge in Invisible do
if edge in Visible then
Sort intersections.
Output segments depending on the operation.
end
end

Step 4
for each edge or edge segment which is a part of the
output do
Display edge or edge segment.
end