ASME, Intelligent and Integrated Manufacturing Analysis and Synthesis, PED-Vol. 25 Dec. 87.

NON-REGULAR DECOMPOSITION: AN EFFICIENT APPROACH FOR SOLVING THE POLYGON INTERSECTION PROBLEM

G. Vanecek Jr. and D. S. Nau
Department of Computer Science
University of Maryland
College Park, Maryland

ABSTRACT

Most approaches to the integration of solid modeling with automated process planning have essentially involved extracting features from the model and sending them the planning system, with no further interaction between the planner and the modeler. Further integration of solid modeling and process planning will require extensive interaction between the process planning system and the solid modeler during process planning. Most existing solid modeling systems are unable to do this efficiently, because of limitations in the algorithms they use.

As a first step toward overcoming this problem, this paper describes an efficient algorithm for producing the boundaries of objects resulting from regularized set operations (such as union, intersection, and subtraction) on two-dimensional polygons. The algorithm is significantly faster than the previously available algorithms for this task. For future work, we intend to extend our algorithm to handle three-dimensional objects containing both flat and curved surfaces, resulting in an efficient solid modeler for use in automated process planning.

INTRODUCTION

Most approaches to the integration of solid modeling with automated process planning have essentially involved using the modeler as a front end to the process planning system. For example, in feature-based planning systems such as SIPS[5], machinable features are produced from the model and sent to the planning system, which then reasons about these features without further interaction with the solid modeler.

In order to generate correct process plans for complex objects, this approach is not sufficient. What processes can be used for some machinable feature—or whether the feature can even be made at all—may depend on geometric information not available solely from the feature description. In order to produce correct process plans for complex objects, it will be necessary for the process planning system to interact extensively with the solid modeler while the process planning is going on. Thus, the solid modeler must be able to answer a large number of queries and perform a large number of incremental changes to the solid, all in an efficient manner.

Although many solid modeling systems exist, the primary focus guiding their development has been the fact that they will be used by humans rather than machines. Thus, much work has been done on efficient algorithms for operations such as rendering, but not so much work has been done on making it efficient to answer queries and make incremental changes. It is the goal of our research to develop techniques for representing and manipulating complex solid objects that can do these operations more efficiently than the modelers currently available.

As a first step in this direction, this paper describes an algorithm for producing the boundaries of objects resulting from regularized set operations (such as union, intersection, and subtraction) on two-dimensional polygons. The algorithm is significantly faster than the previously available algorithms for this task.

Most other algorithms for this task use a global approach of checking each edge of one object with respect to every edge of the other object. This results in a cost that is worse than $O(n^2)$. The algorithm described here uses a local approach based on the divide-and-conquer paradigm in the form of non-regular decomposition of space[2,3]. This results in an average-case performance which is empirically shown to be $O(n \log n)$, where n is the total number of edges of both the input and the output.

For future work, we intend to extend our algorithm to handle three-dimensional objects containing both flat and curved surfaces. This will result in an efficient solid modeler for use in automated process planning.

SET OPERATIONS

It is well-known that when set theoretic operations such as union, intersection, and subtraction are applied to two valid n-dimensional objects, the result is not necessarily a valid n-dimensional object. For example, if two squares touch on one side, their intersection is a single line segment, which is not a valid two dimensional object.

Requicha and Voelcker[7] have shown that this difficulty can be overcome by using regularized set operations instead of ordinary set operations. For example, regularized set intersection can be defined as

$$S \cap^* T = \mathrm{r}(S \cap T)$$
$$S -^* T = \mathrm{r}(S - T),$$

where the regularization $\mathrm{r}S$ of an object represented by S removes all lower dimensional topological entities that do not bound the interior of the object represented by S.

An object may be modeled by its boundary. Topologically, this boundary consists of a set of edges and a set of vertices. Geometrically, each edge is associated with a directed line, and each vertex is associated with a point. The direction associated with each edge is arbitrarily chosen so that the interior of the object lies to the right of the edge. In terms of these directed edges, a polygon is either clockwise, enclosing a finite interior, or counterclockwise, denoting a hole.

The computation of a set operation on such objects can be separated into two stages. Stage one performs edge classification and stage two generates the edges needed for the desired operation. Edge classification labels each edge of both objects by a classification that tells where the edge is with respect to the other object. Edge classification was formalized by Tilove[10]. He showed that an edge in a given object can be classified with respect to the other objects by dividing the edge into segments, each of which falls into one of three sets: **IN**, **ON**, and **OUT**. Once this classification was done, differentiation among various types of **ON** edges was done using edge neighborhoods[7].

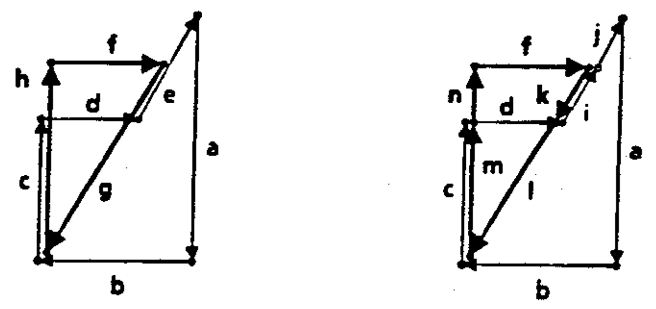

Fig. 1 Before and After Edge Classification of Edges.

We follow Tilove's approach, with one difference: for two-dimensional objects, the ON edges can directly be divided into two types, without the need of using edge neighborhoods. Thus, an edge bounding a two dimensional object will acquire one of four classifications:

OUT	the edge is outside the other object;
IN	the edge is inside the other object;
ON$^{\uparrow\uparrow}$	the edge is on the boundary of the other object, and both objects are on the same side of the edge;
ON$^{\uparrow\downarrow}$	the edge is on the boundary of the other object, and the objects are on opposite sides of the edge (i.e., the objects touch along that edge).

Since each edge must be uniquely classified with a single classification, the edges of one object that cross or partially touch the other object must be split by the introduction of new vertices. Such vertices are called *transition vertices*. The classification stage thus requires that the two objects split each other so that no edge of one object penetrate the other object.

Once the edges of an object represented by S have been partitioned into the four classification sets $S_{\mathrm{IN}}T$, $S_{\mathrm{OUT}}T$, $S_{\mathrm{ON}}^{\uparrow\uparrow}T$, $S_{\mathrm{ON}}^{\uparrow\downarrow}T$, and the boundary of object represented by T has been partitioned into the four classification sets $T_{\mathrm{IN}}S$, $T_{\mathrm{OUT}}S$, $T_{\mathrm{ON}}^{\uparrow\uparrow}S$, and $T_{\mathrm{ON}}^{\uparrow\downarrow}S$, the boundaries belonging to the desired set operation can be determined. Thus, the results of regularized set operations can be derived directly without the need for regularization. In particular, if S is the boundary of the object S,

$$S \cup^* T = S_{\mathrm{OUT}}T \cup T_{\mathrm{OUT}}S \cup S_{\mathrm{ON}}^{\uparrow\uparrow}T,$$
$$S \cap^* T = S_{\mathrm{IN}}T \cup T_{\mathrm{IN}}S \cup T_{\mathrm{ON}}^{\uparrow\uparrow}S,$$

and

$$S -^* T = S_{\mathrm{OUT}}T \cup (T_{\mathrm{IN}}S)^{-1} \cup S_{\mathrm{ON}}^{\uparrow\downarrow}T,$$
$$T -^* S = T_{\mathrm{OUT}}S \cup (S_{\mathrm{IN}}T)^{-1} \cup T_{\mathrm{ON}}^{\uparrow\downarrow}S.$$

It follows from the above equations that if the two classification sets $S_{\mathrm{ON}}^{\uparrow\downarrow}T$ and $T_{\mathrm{ON}}^{\uparrow\downarrow}S$ are ignored, the remaining six sets will construct both the union and the intersection. Analogously, if the two classification sets $S_{\mathrm{ON}}^{\uparrow\uparrow}T$ and $T_{\mathrm{ON}}^{\uparrow\uparrow}S$ are removed, the remaining six sets will construct both $S -^* T$ and $T -^* S$. In the case of subtraction, $(S_{\mathrm{IN}}T)^{-1}$ denotes the set of all directed edges in $S_{\mathrm{IN}}T$ with their directions reversed.

As an example of edge classification, consider the two objects represented by S and T in figure 1. The edge set S and T are $\{a, b, c, d, e\}$ and $\{f, g, h\}$.

After edge splitting and classification, the eight classification sets are

$$\begin{aligned} S_{\mathrm{OUT}}T &= \{a,b,j\} \qquad & T_{\mathrm{OUT}}S &= \{f,n\} \\ S_{\mathrm{IN}}T &= \{d\} & T_{\mathrm{IN}}S &= \{l\} \\ S_{\mathrm{ON}}^{\uparrow\uparrow}T &= \{c\} & T_{\mathrm{ON}}^{\uparrow\uparrow}S &= \{m\} \\ S_{\mathrm{ON}}^{\uparrow\downarrow}T &= \{i\} & T_{\mathrm{ON}}^{\uparrow\downarrow}S &= \{k\} \end{aligned}$$

Using equations (8) and (9),

$$\begin{aligned} S \cup^* T &= \{a,b,j,f,n,c\} \\ S \cap^* T &= \{d,l,m\} \end{aligned}$$

THE ALGORITHM

This section presents the algorithm for performing binary set operations on two dimensional objects. The algorithm is as shown below:

```
Algorithm PerformSetOp(Objects $S$, $T$; Op)
1. $(S', T')$ = MakeObjects(Op, ClassifyEdges($S, T$))
2. CleanUp($S'$)
3. CleanUp($T'$)
4. return($S', T'$)
```

The argument *Op* to *PerformSetOp* indicates whether the union/intersection operations (8, 9) or the subtraction operations (10, 11) are desired. Accordingly, the resulting objects S' and T' are either $S \cap^* T$ and $S \cup^* T$, or $S -^* T$ and $T -^* S$. The algorithm first classifies the edges of both objects by the *ClassifyEdges* algorithm which produces the eight edge classification sets–see (12). These classification sets and the desired operation pairs are then passed to the *MakeObjects* algorithm which selects the desired edges and reconstructs the boundaries. At this point, the sets are regularized; however, since there can exist adjacent pair of collinear edges–that is, redundant vertices–these boundaries are cleaned up by the *CleanUp* algorithm. Each of these steps are discussed below.

Edge Classification Using Decomposition

The edge classification algorithm described below uses object oriented decomposition of space to recursively fragment the edge sets into easily classifiable sets. The algorithm falls somewhere in the middle of a spectrum of similar two dimensional approaches. At one end of the spectrum are the spatially based approaches such as quad-trees, where the division criteria is related to the spatial coordinate system[1]. At the other end of the spectrum are object-based approaches, in which the object's characteristics are the sole means of division of the space, and division boundaries, called splitting lines, are determined from the set of edges. The approaches at this end of the spectrum include the Weiler polygon clipping algorithm[7] and the Sutherland and Hodgman clipper[12] for general polygon clipping.

It is not unusual to implement edge classification by a simple double nested loop in which each edge of one object is checked against each edge of the other object. Improvement upon this simple strategy can be achieved by utilizing a divide and conquor paradigm on the edge sets. Instead of considering each edge independently, the edge sets are recursively split until trivial cases are found for which the same classification label can be applied to all its edges. The edge sets are thus fragmented into several disjoint sets of edges, called *fragments*. The boundary classification algorithm accepts two such fragments, A and B, where fragment A is a subset of the edges of S and fragment B is a subset of the edges of T. In the first call to *ClassifyEdges*, A and B contain all the edges of S and T respectively. Thereafter, each call works on progressively smaller fragments. At each call, the algorithm returns the eight classification sets in the following order:

$$\left(A_{\mathrm{IN}}T, A_{\mathrm{OUT}}T, A_{\mathrm{ON}}^{\uparrow\uparrow}T, A_{\mathrm{ON}}^{\uparrow\downarrow}T, B_{\mathrm{IN}}S, B_{\mathrm{OUT}}S, B_{\mathrm{ON}}^{\uparrow\uparrow}S, B_{\mathrm{ON}}^{\uparrow\downarrow}S\right)$$

The algorithm for boundary classification follows.

```
Algorithm ClassifyEdges(Fragments $A$, $B$)
1.  if $A = \{E\}$ and $B = \{F\}$ and
      Edges $E$ and $F$ are collinear with common endpoints then
2.      if $E$ has the same direction as $F$
3.      then return($\emptyset, \emptyset, A, \emptyset, \emptyset, \emptyset, B, \emptyset$)
4.      else return($\emptyset, \emptyset, \emptyset, A, \emptyset, \emptyset, \emptyset, B$)
5.  else if $A$ is empty then
6.      if region containing $B$ is inside $S$
7.      then return($\emptyset, \emptyset, \emptyset, \emptyset, B, \emptyset, \emptyset, \emptyset$)
8.      else return($\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, B, \emptyset, \emptyset$)
9.  else if $B$ is empty then
10.     if region containing $A$ is inside $T$
11.     then return($A, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$)
12.     else return($\emptyset, A, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$)
    else begin(* Recursively Decompose and try again *)
13.     line $L$ = FindSplitLine($A$, $B$)
14.     ($A_{\mathrm{Left}}$, $A_{\mathrm{Right}}$) = SplitFragment($A$, $L$)
15.     ($B_{\mathrm{Left}}$, $B_{\mathrm{Right}}$) = SplitFragment($B$, $L$)
16.     ($C_L^i$, for $i = 1..8$) = ClassifyEdges($A_{\mathrm{Left}}$, $B_{\mathrm{Left}}$)
17.     ($C_R^i$, for $i = 1..8$) = ClassifyEdges($A_{\mathrm{Right}}$, $B_{\mathrm{Right}}$)
18.     return($C_L^i \cup C_R^i$, for $i = 1..8$)
    end
```

Steps 1 through 12 are the trivial cases in which the edges of fragments A and B can be classified without further decomposition. Of these, only conditions 6 and 10 require careful geometric evaluation which is not elaborated on here. The usual way of checking these conditions is to cast a ray. A careful consideration shows that a ray needs only be cast through the parent fragment of the now empty fragment. This itself can be reduced to considering only one edge of that parent fragment. The details of can be found in a technical report written by the authors[13].

When both fragments contain more than one edge, steps 13 though 18 perform the fragments' decomposition. Step 13 first chooses a splitting line L. This line demarcates the region containing the fragments into a left open halfplane, and a right closed halfplane. Using this splitting line, steps 14 and 15 partition the fragments A and B into their corresponding left fragment A_{Left} and B_{Left}, and the right fragments A_{Right} and B_{Right}. Thus, the splitting line cuts the region containing both fragments into two subregions and decomposes the contained fragment into a left and a right subfragment. This splitting line is chosen deterministically from among the lines associated with the edges contained in the fragments. Since a given line should only be chosen once, each edge is tagged when its line is chosen for splitting. Using such tags allows for efficient line selection. Given the splitting line L, and an edge set A, the left and right fragments can be defined by:

$$A_{\text{Right}} = A_1 \cup A_2 \cup A_3$$
$$A_{\text{Left}} = A_4 \cup A_5 \cup A_6 \cup A_7$$

where the fragments

$$A_1 = \{(\overline{P,Q}) \in A \mid P \succeq L \wedge Q \succeq L\}$$

is the sets of edges that are on or to the right of the splitting line, and

$$A_2 = \left\{\left(\overline{\text{Inter}(\text{Line}(P,Q),L),Q}\right) \mid (\overline{P,Q}) \in A \wedge P \prec L \wedge Q \succ L\right\}$$

and

$$A_3 = \left\{\left(\overline{P,\text{Inter}(\text{Line}(P,Q),L)}\right) \mid (\overline{P,Q}) \in A \wedge Q \prec L \wedge P \succ L\right\}$$

are the set of new edges that resulted from splitting edges that cross the splitting line. Similarily, the left fragment consists of the edge sets

$$A_4 = \{(\overline{P,Q}) \in A \mid P \preceq L \wedge Q \prec L\},$$

$$A_5 = \{(\overline{P,Q}) \in A \mid Q \preceq L \wedge P \prec L\},$$

$$A_6 = \left\{\left(\overline{\text{Inter}(\text{Line}(P,Q),L),Q}\right) \mid (\overline{P,Q}) \in A \wedge P \succ L \wedge Q \prec L\right\},$$

and

$$A_7 = \left\{\left(\overline{P,\text{Inter}(\text{Line}(P,Q),L)}\right) \mid (\overline{P,Q}) \in A \wedge P \prec L \wedge Q \succ L\right\}.$$

In the above, $(\overline{P,Q})$ is the line segment between the points P and Q, Inter(L_1, L_2) is the intersection point of the two lines L_1 and L_2, and $\prec$, $\preceq$, $\succeq$ and $\succ$ are predicates for point/line classification (e.g., $P \succeq L$ is true if point P is on or right of line L).

The usual approaches to decomposition of space (such as quadtree approaches) always create rectilinear regions.[1,8] However, the object-oriented decomposition used by *Splitfragment* instead creates arbitrarily shaped convex regions[6]. Conceptually, the edges contained in the fragment to be split lie entirely within some convex region. The splitting of the fragment partitions the edges into two subfragments that fall entirely into two convex subregions, one on each side of the splitting line.

As an example of the classification process the edges of two objects S and T of figure 1, are shown in the tree in figure 2. In the figure, each node of the tree denotes the fragment resulting from splitting the parent fragment.

The classification algorithm resembles Hoare's Quicksort algorithm[4], which divides a set of keys to be sorted into two subsets around some partition element. The average-case complexity of Quicksort results from the fact that although the partition element is chosen randomly, it will usually not be too far from the middle of the set, and thus the two subsets will be of comparable size.

The *SplitFragment* algorithm partitions the edges about some line chosen from among the edges. In the Quicksort splitting algorithm, any individual element is a valid candidate, but in *SplitFragment*, not every edge can be used to provide a candidate splitting line. Instead, a splitting line can be used only if its application maintains properties P1 and P2. This requires the examination of at least two edges.

Creati

Aft
all the
vertices.
be relinl
creating
vertex 1
the sam

ANAL

The
algorith
m to be

$C(n, r$

The first
conditio
line, the
for class
of subre
used and
analytic

ontained in
agged when
. Given the
l by:

tting line.

1, L_2) is
cates for

ches) al-
used by
he edges
he split-
into two

figure 1,
ragment

ivides a
ge-case
chosen
subsets

among
didate,
g line.
nd P2.

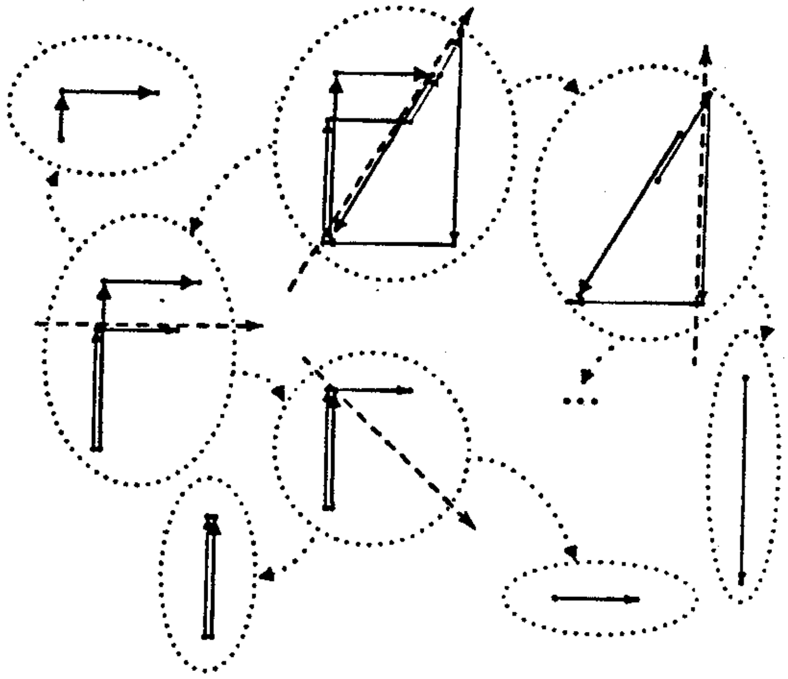

Fig. 2 The trace resulting from the decomposition of objects S and T.

Creating the Desired Boundaries

After edge classification, the desired boundaries can be constructed by first collecting all the proper edges (as indicated by (8) through (11)) and by eliminating redundant vertices. If the objects are modeled by connected edge loops, then transition vertices must be relinked. As an example from figure 1, vertices 5 and 11 need their edges relinked when creating the union and the intersection. Once this has been done, pseudo vertices, such as vertex 10, should be removed. A pseudo vertex is a vertex whose two incident edges lie on the same line.

ANALYSIS

The classification algorithm presented manipulates edges. The cost $C(n,m)$ of the algorithm can be expressed in a recurrence relation on the size of the two fragments n and m to be classified:

$$C(n,m) = \begin{cases} n+m & \text{if } n=0 \text{ or } m=0 \text{ or } n=m=1 \\ 1+n+m+C(n_L,m_L)+C(n_R,m_R) & \text{o/w, } 0 \le n_L, n_R \le n, \\ & \quad 0 \le m_L, m_R \le m \end{cases}$$

The first condition covers steps 1 through 12 of the *ClassifyEdges* algorithm. The second condition handles steps 13 through 18. This consists of a cost of 1 for selecting the splitting line, the cost of $n+m$ for splitting the fragments and the costs $C(n_L,m_L)$, and $C(n_R,m_R)$ for classifying the left and the right subregions respectively. The bounds on the sizes of subregions are very loose. The actual values depend greatly on the splitting strategy used and more so on the probability. As such the closed form might never be determined analytically. Although the worst case should be easier to determine than the average case, it has been observed that in geometric modeling systems, the worst case is generally very pessimistic and sheds little light on the average case running time[14].

The average case running time can be observed statistically. The algorithm's complexity has been measured under numerous test runs. Conclusions are derived from statistical analysis of the collected data. This is done by measuring several key operators utilized by the algorithm. One such key operation is any geometrical consideration of an edge. Using an algorithm for generating random polygons of arbitrary size, several batches of 50 pairs of unique polygons have been intersected and their averages plotted. Eight such batches with increasing number of edges from 4 to 512 each has been performed. Figure 3a shows the average number of edge comparisons.

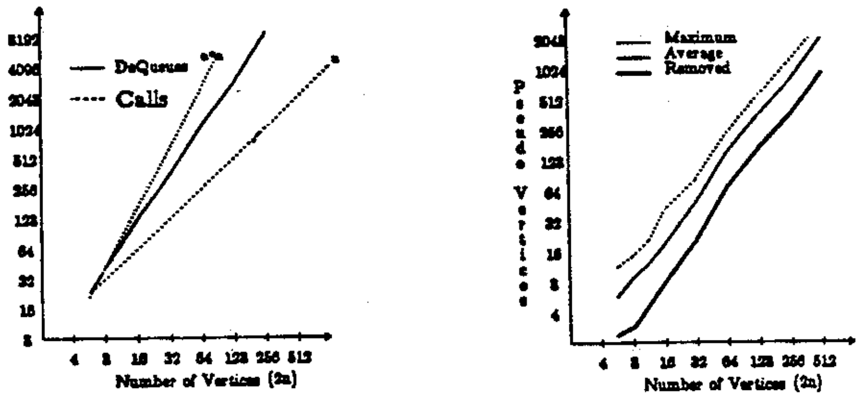

Fig. 3 Edge comparisons and Pseudo Vertices vs Size.

An edge comparison is taken to be the action of removing an edge from a fragment for some geometric consideration. The curve for the number of edge comparisons fits between the curves for n and n^2 shown as dotted lines. The same figure also indicates the number of recursive calls to *ClassifyEdges*. This simple statistical analysis shows that at least in practice, the algorithm has an average-case complexity that is better than quadratic for intersecting polygons of equal size. Figure 3b shows the relation between the polygon size and the number of pseudo vertices generated and removed.

Generating the desired boundaries from the eight classification sets can be performed in $O(n \log n)$ steps, where n is the sum of both the input number of edges and the total number of edges produced by the *ClassifyEdges* algorithm. Note that the actual number of resulting edges can be of $O(m^2)$ given m input edges. This would occur when every edge of one object crosses every edge of the other object. Finally, removing redundant pseudo vertices can then be performed in $O(n)$ steps.

CONCLUSION

This algorithm has been implemented in C, in less than 1000 lines of C code, and runs on a Sun-3 under Unix. It has been tried on a large number of problems, including a number of examples specially designed to be hard, and has been found to run very fast. For example, intersecting two objects with 32 edges each resulting in roughly 90 edges each

takes
edges
I
of eve
stand,
most
decon
few ar
C
set op
intenc
dimer
imple

REF

1. A
N
1
2. E
2
3. E
t
4. E
5. N
f
A
6. E
c
7. E
a
8. S
S
9. S
S
10. T
t
12. S
(
13. V
I
T
14. V
1

takes less than 3 seconds; intersection two object with 256 each that result in around 840 edges each takes about 45 seconds.

In addition to providing improved time complexity, reducing implementation detail is of even more importance. Solid modeling systems are typically large and hard to understand, at least partly because of the need for handling numerous special cases. One of the most appealing features of our algorithm is its simplicity. It has the inherent ability to decompose until only the most primitive topological components exist–and handling the few arising cases is thus much simplified.

Currently work is in progress in producing a solid modeler which can do efficient set operations on three-dimensional solids modelled by a boundary representation. We intend to do this by extending the non-regular spatial decomposition algorithm to three dimensions. Theoretically, this problem is well understood, but it is still a challenge to implement efficiently.

REFERENCES

1. Ayala D., Brunet P., Juan R., and Navazo I., Object Representation by Means of Nonominimal Division Quadtrees and Octrees, *ACM Trans. on Graphics* 4, 1 (Jan 1985), 41-59.
2. Bentley J. L., Multidimensional divide and conquor, *Comm. ACM* 23, 4, (April 1980) 214-229.
3. Fuchs, H., Kedem, M. Z., and Naylor, F. B., On visible surface generation by a priori tree structres, *Conf. Proc. of Siggraph '80* 14, 3, (1980), 124-133.
4. Hoare, C. A. R., Quicksort, *The Computer Journal* 5, (1962), 10-15.
5. Nau, D. S., and Gray, M., Hierarchical Knowledge Clustering: A New Representation for Problem-Solving Knowledge, *in* J. Hendler, *Expert Systems: The User Interface*, Ablex, 1987, to appear.
6. Preparata, Franco P., and Shamos, Michael Ian, Computational Geometry, An Introduction, Springer-Verlag, 1985.
7. Requicha, A. A. G., and Voelcker, H. B., Boolean Operations in Solid Modeling Boundary Evaluation and Merging Algorithms, *Proc. IEEE* 73, 1 (Jan 1985), 30-44.
8. Samet, H., The quadtree and related hierarchical data structures, *ACM Computing Surveys* 16, 2 (June 1984), 187-260.
9. Shamos, I. M., and Hoey Dan, Geometric Intersection Problems *7th Annual IEEE Symp. on Foundations of Comp. Sci.* (Oct. 1976), 208-215.
10. Tilove, R. B., Set Membership Classification: A Unified Approach to Geometric Intersection Problems, *IEEE Trans. Comp.* C-29, 10 (Oct 1980), 874-883.
12. Sutherland I. E., and Hodgman G. W., Reentrant polygon clipping, *CACM*, 17, 1, (Jan 1974), 32-42.
13. Vanecek G., Nau D. S., Computing Geometric Boolean Operations by Input Directed Decomposition, Systems Research Center, University of Maryland, Technical Report TR-87-8, Jan 1987.
14. Weiler K., Polygon comparison using a graph representation, *SIGGRAPH '80*, (1980), 10-18.