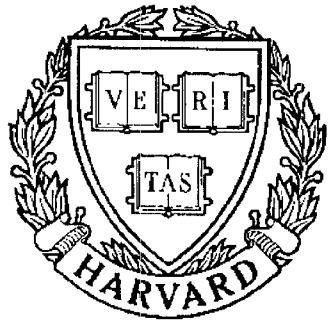
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), the University of Maryland, Harvard University, and Industry

A Simple Approach to Performing Set Operations on Polyhedra

by G. Vanecek, Jr., D.S. Nau and R.R. Karinthi

TR 91-34

A Simple Approach to Performing Set Operations on Polyhedra*

George Vaněček, Jr.
Department of Computer Science
Purdue University
West Lafayette, IN 47907
vanecek@cs.purdue.edu

Dana S. Nau
Computer Science Department
and Systems Research Center
University of Maryland
College Park, MD 20742
nau@cs.umd.edu

Raghu R. Karinthi
Department of Statistics and Computer Science
and Concurrent Engineering Research Center
West Virginia University
Morgantown, WV 26506
raghu@cs.wvu.wvnet.edu

Abstract

In performing regularized set operations on two solids, the most difficult step is boundary classification, in which the boundaries of each solid are split into portions that are inside, outside, or on the surface of the other solid. In this paper, we present a method for doing boundary classification on polyhedral solids. The approach is based on recursively decomposing space based on the boundaries of the solids being classified.

This approach has several appealing properties: it is simple to describe, efficient (tests indicate $O(n \log n)$ complexity in a variety of cases), and can handle both manifold and non-manifold 3-D solids. This approach serves as the basis for set operations in the Protosolid solid modeler.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms.*
General Terms: Algorithms, Design
Additional Key Words and Phrases: non-regular decomposition, boolean set operations, polyhedra, non-manifolds.

*This work has been supported in part by an NSF Presidential Young Investigator Award to Dana Nau with matching funds provided by General Motors Research Laboratories and Texas Instruments, NSF Grant NSFD CDR-85-00108 to the University of Maryland Systems Research Center, and NSF Grant NSFD CCR-86-19817 to Purdue University, and the DARPA Contract No. MDA972-88-C-0047 for the DARPA Initiative in Concurrent Engineering (DICE).

1 Introduction

The importance of regularized set operations in solid modeling is widely recognized [Req80]. The basic approach for performing regularized set operations on two boundary representations (B-Reps) can be separated into the following three steps:

1. Perform *boundary classification* on the two solids; that is, split the faces of each solid into sub-faces, each of which is inside, outside, or on the surface of the other solid.

2. Assemble the appropriate faces to produce the result of the desired set operation.

3. Topologically reduce the result, by combining coplanar adjacent faces and collinear adjacent edges.

Of these, the first step—boundary classification—is the most difficult. For B-Reps, boundary classification can be done in a brute-force manner by comparing every face of one solid with every face of the other, but this involves a quadratic number of comparisons, incurring a worst-case time complexity of at least $\Omega(n^2)$.[1] However, often many of the faces being compared do not actually intersect each other—and in such cases, significantly more efficient performance can be achieved if ways can be found to localize the face comparisons.

In this paper, we present a divide-and-conquer method for boundary classification that achieves locality of face comparisons by decomposing space in a non-regular manner called *input-directed decomposition*. The basic algorithm is relatively simple to describe, and it has been proved correct in [Van89]. Since it proceeds purely by dividing faces into subfaces, the algorithm does not require auxiliary data structures to represent explicitly the regions or the relationships among them. Instead, it is sufficient merely to keep track of what set of subfaces falls into each region.

Based on this algorithm, we have built a solid modeler called Protosolid. Protosolid is being used in several projects, at the University of Maryland [KNY91], Purdue University [NEW90, Van91a, Van91b], and Cornell University [NEW90]. In using Protosolid, we have found the algorithm to be quite efficient—and in this paper, we present the results of tests showing its time complexity to be $O(n \log n)$ on "typical case" problems.

In many previous approaches to boundary classification, limitations in the algorithm or the data structures have caused difficulty in handling non-manifold 3-D solids[2] [Bau72, Man88, GS85, PRS86]. Our input-directed decomposition algorithm is capable of handling non-manifold 3-D solids—and our implementation of this algorithm in the Protosolid solid modeler [Van89] incorporates data structures capable of representing such solids. Thus, Protosolid can easily handle non-manifold 3-D solids.

Section 2 contains the mathematical preliminaries needed to describe the algorithm. Section 3 describes the algorithm, as well as some specific methods for performing some of its basic operations. Section 4 briefly discusses our implementation of the algorithm in the Protosolid solid modeler, and Section 5 describes our measurements of the algorithm's

[1] We believe the worst-case complexity is actually $\Omega(n^2 \log n)$, because of the overhead involved in repeatedly searching for various faces, edges, and vertices—but we have not attempted to prove this.

[2] By "non-manifold 3-D solids", we mean regularized 3-dimensional sets that do not happen to be 2-manifolds. This set does not include objects that are not homogeneously three-dimensional, such as those discussed by Weiler [Wei86].

time complexity on a number of different solid modeling problems. Section 6 compares our approach to other related work, and Section 7 contains concluding remarks.

2 Mathematical Preliminaries

2.1 Standard Definitions

Below, we briefly review some of the basic definitions needed to characterize three-dimensional solids. Most of these definitions are quite well-known (for example, see Requicha and Tilove [RT78, Req77], Kuratowski [KM76] and Mendelson [Men75]).

Let X be any subset of Euclidean 3-space (E^3). Then $X^{-1} = E^3 - X$ is the complement of X. X is *compact* if and only if it is closed and bounded. kX and iX are the *closure* and *interior* of X, respectively; and $rX = k(i(X))$ is the *regularization* of X. X is *regular* if $X = rX$ (in intuitive terms, this means that X has no dangling faces, dangling edges or isolated points). X is *semi-analytic* if it is a finite combination, via the set operations union, intersection and complement, of sets X_i of the form $X_i = \{p \in E^3 : f_i(p) \geq 0\}$, where f_i is any analytic function on E^3. X is a *solid* if it is compact, regular and semi-analytic. $b(X) = kX - iX$ is the boundary of X. It can be shown [RT78] that the set of all solids is closed under the following *regularized set operations*:

$$X \cup^* Y = r(X \cup Y);$$

$$X \cap^* Y = r(X \cap Y);$$

$$c^* X = r(X^{-1});$$

$$X -^* Y = r(X - Y).$$

2.2 The Classification Scheme

We define a *fragment* to be any regular, semi-analytic subset of the boundary of a solid. Note that a fragment need not necessarily be connected. Since the boundary of a solid S is regular and semi-analytic, $b(S)$ itself is a fragment. It can also be shown that the set of all fragments of a solid is closed under regularized union, intersection, and difference [RT78]. Let S and T be any two solids, and f be any fragment of S. Then we say that

1. f is *homogeneously in* T if either $i(f) \subseteq i(T)$ or f is the union of finitely many smaller fragments $f_1, \ldots, f_k$, each of which is homogeneously in T;

2. f is *homogeneously outside* T if either $i(f) \subseteq T^{-1}$ or f is the union of finitely many smaller fragments $f_1, \ldots, f_k$, each of which is homogeneously outside T;

3. f is *homogeneously with* T if either (a) $f \subseteq b(T)$, and for every point $p \in i(f)$ and every neighborhood $N(p)$, $(N(p) \cap S) \cap^* (N(p) \cap T) \neq \emptyset$; or (b) f is the union of finitely many smaller fragments $f_1, \ldots, f_k$, each of which is homogeneously with T.

4. f is *homogeneously against* T if either (a) $f \subseteq b(T)$, and for every point $p \in i(f)$ there is a neighborhood $N(p)$ such that $(N(p) \cap S) \cap^* (N(p) \cap T) = \emptyset$; or (b) f is the union of finitely many smaller fragments $f_1, \ldots, f_k$, each of which is homogeneously against T.

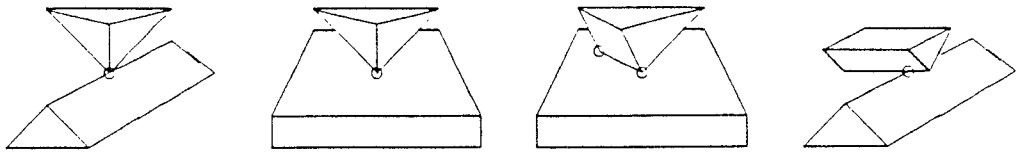


Figure 1: Examples of singularities that can occur in non-manifold 3-D solids.

For each fragment f of S, at most one of the above properties can hold. If one of them does hold, then we say that f is *T-homogeneous* (or simply *homogeneous*, if the identity of T is clear).

In the above definitions, allowing f to be decomposed into $f_1, \ldots, f_k$ enables us to handle singularities that can arise in non-manifold situations such as the ones shown in Fig. 1.

Given any two solids S and T, the following four fragments are the *classification sets* for S with respect to T:

S IN T is the largest fragment x of $b(S)$ that is homogeneously in T:
S OUT T is the largest fragment x of $b(S)$ that is homogeneously outside T:
S WITH T is the largest fragment x of $b(S)$ that is homogeneously with T:
S ANTI T is the largest fragment x of $b(S)$ that is homogeneously against T.

$b(S)$ is the union of these four fragments. The IN and OUT classifications used by Tilove [Til80] correspond to our IN and OUT classifications, respectively, and his ON classification corresponds to the union of our WITH and ANTI classifications.

From the above definitions, it is straightforward to show that

$$b(S \cup^* T) = (S \text{ OUT } T) \cup (T \text{ OUT } S) \cup (S \text{ WITH } T); \quad (1)$$
$$b(S \cap^* T) = (S \text{ IN } T) \cup (T \text{ IN } S) \cup (S \text{ WITH } T); \quad (2)$$
$$b(S -^* T) = (S \text{ OUT } T) \cup (T \text{ IN } S) \cup (S \text{ ANTI } T); \quad (3)$$
$$b(T -^* S) = (T \text{ OUT } S) \cup (S \text{ IN } T) \cup (S \text{ ANTI } T). \quad (4)$$

3 The Algorithm

Eq.'s 1–4 state that from the classification sets, we can compute any regularized set operation we desire. In a B-Rep of a polyhedron, solids and fragments are represented as collections of planar faces, each of which has a finite number of bounding edges and a normal vector pointing outwards from the solid. Thus for B-Reps of polyhedra, our approach is as follows: given collections of faces $\mathcal{S}$ representing $b(S)$ and $\mathcal{T}$ representing $b(T)$, find collections of faces $\mathcal{S}_{\text{IN}}$ representing the classification set S IN T, $\mathcal{T}_{\text{IN}}$ representing the classification set T IN S, and so forth. Once this has been done, the results of the regularized set operations are represented by the following collections of faces:

$$b(S \cup^* T) \text{ is represented by } \mathcal{S}_{\text{OUT}} \cup \mathcal{T}_{\text{OUT}} \cup \mathcal{S}_{\text{WITH}}; \quad (5)$$

procedure CLASSIFY($\mathcal{S}, \mathcal{T}$)

1. Initially, set $\mathcal{S}_{\text{IN}} = \mathcal{S}_{\text{OUT}} = \mathcal{S}_{\text{WITH}} = \mathcal{S}_{\text{ANTI}} = \mathcal{T}_{\text{IN}} = \mathcal{T}_{\text{OUT}} = \mathcal{T}_{\text{WITH}} = \mathcal{T}_{\text{ANTI}} = \emptyset$, and $X = \{(\mathcal{S}, \mathcal{T})\}$. X is the set of all fragments that need to be examined further.

2. Repeat Steps 3–5 until $X = \emptyset$. Then return $\mathcal{S}_{\text{IN}}$, $\mathcal{S}_{\text{OUT}}$, $\mathcal{S}_{\text{WITH}}$, $\mathcal{S}_{\text{ANTI}}$, $\mathcal{T}_{\text{IN}}$, $\mathcal{T}_{\text{OUT}}$, $\mathcal{T}_{\text{WITH}}$, and $\mathcal{T}_{\text{ANTI}}$.

3. Select a pair of fragments $(s, t) \in X$, and remove it from X.

4. If s is either empty or T-homogeneous and t is either empty or S-homogeneous, then do the following:

   - If s is nonempty, then put it into one of $\mathcal{S}_{\text{IN}}$, $\mathcal{S}_{\text{OUT}}$, $\mathcal{S}_{\text{WITH}}$, or $\mathcal{S}_{\text{ANTI}}$, depending on its classification. If t is nonempty, then put it into one of $\mathcal{T}_{\text{IN}}$, $\mathcal{T}_{\text{OUT}}$, $\mathcal{T}_{\text{WITH}}$, or $\mathcal{T}_{\text{ANTI}}$, depending on its classification.

5. Otherwise, do the following:

   (a) Select a closed half-space H that intersects s and t. (We call the plane P that bounds H the *splitting plane*.)

   (b) Split s and t into subfragments s_1, s_2, t_1, and t_2 representing $s \cap^* H$, $s -^* H$, $t \cap^* H$, and $t -^* H$, respectively. Put (s_1, t_1) and (s_2, t_2) into X.

end CLASSIFY

Figure 2: Divide-and-conquer algorithm for boundary classification.

$$b(S \cap^* T) \text{ is represented by } \mathcal{S}_{\text{IN}} \cup \mathcal{T}_{\text{IN}} \cup \mathcal{S}_{\text{WITH}}; \tag{6}$$

$$b(S -^* T) \text{ is represented by } \mathcal{S}_{\text{OUT}} \cup (\mathcal{T}_{\text{IN}})^- \cup \mathcal{S}_{\text{ANTI}}; \tag{7}$$

$$b(S -^* T) \text{ is represented by } \mathcal{T}_{\text{OUT}} \cup (\mathcal{S}_{\text{IN}})^- \cup \mathcal{S}_{\text{ANTI}}; \tag{8}$$

where $(\mathcal{S}_{\text{IN}})^-$ and $(\mathcal{T}_{\text{IN}})^-$ are the same as $\mathcal{S}_{\text{IN}}$ and $\mathcal{T}_{\text{IN}}$, respectively, except that the faces have their normal vectors pointing in the opposite direction. As mentioned in Section 1, these representations can then be simplified by combining coplanar adjacent faces and collinear adjacent edges.

Fig. 2 shows our divide-and-conquer algorithm CLASSIFY for computing collections of faces representing the classification sets. This algorithm maintains a set X that contains pairs of fragments. Initially, X contains only the pair $(\mathcal{S}, \mathcal{T})$. On each iteration of its main loop, the algorithm removes a pair of fragments (s, t) from X, and checks them for homogeneity. If the algorithm determines that they are homogeneous, then it classifies them. Otherwise, it uses a splitting plane P to split s and t into pairs of subfragments (s_1, t_1) and (s_2, t_2), and puts these pairs into X.

The CLASSIFY algorithm is based on certain abstract operations: selecting fragments from X, testing them for homogeneity, selecting a splitting plane P and a half-space H bounded by P, and splitting the fragments. These operations can be performed in many different ways, so long as they produce the desired result. In the following subsections, we describe the algorithms we use for these operations in the Protosolid solid modeler—and in

[Van89], we prove that if these algorithms are used, then CLASSIFY is correct.

To simplify the presentation, we will use the term "fragment" sometimes to refer to the fragment itself, and sometimes to refer to the collection of faces representing the fragment. Which meaning is intended should be clear from context.

3.1 Selecting Fragments

Step 3 of CLASSIFY selects a pair of fragments in X so that they may be classified or split. Each pair of fragments in X must eventually be selected—so which pair of fragments is selected first has no effect on the correctness and time complexity of CLASSIFY. However, to achieve the best space complexity, we always select fragments from X in a depth-first manner.

3.2 Testing for Homogeneity

For correctness of the algorithm, we want the "if" test in Step 4 of CLASSIFY to succeed only if both s and t are either empty or homogeneous. Ideally, we would also like the converse to be true. However, it is computationally expensive to detect *all* the cases where s and t are homogeneous, and it is not necessary to detect all of these cases in order for the algorithm to be correct. Thus, we instead test a simple set of conditions that are sufficient (but not necessary) to guarantee homogeneity. In cases where s and t are homogeneous but our test is not satisfied, CLASSIFY will simply subdivide s and t further, into subfragments for which the the test will succeed.

There are three cases in which we detect homogeneity:

1. $s = \emptyset$. Then t is homogeneously either in or outside S. Distinguishing whether t is in S or outside S can be done straightforwardly. Our technique for doing this is similar but not identical to that used in Thibault and Naylor [TN87]. A discussion of the details of our technique is outside the scope of this paper, but the reader can find these details in [Van89].

2. $t = \emptyset$. Then s is homogeneously either in or outside T; and as above, it is straightforward to tell which.

3. s and t are the same point set, and each is represented by a single face. Then let the faces be f_s and f_t, respectively. If the normal vectors for f_s and f_t point in the same direction, then s is homogeneously with T and t is homogeneously with S. Otherwise, s is homogeneously against T and t is homogeneously against S.

3.3 Selecting a Half-Space

Step 5a of CLASSIFY selects a half-space H, in order to try to separate non-homogeneous portions of s and/or t. Since non-homogeneities occur only along the boundaries of the solids, this suggests that we choose H based on faces of s and t. We call this strategy *input-directed* decomposition.

This selection strategy works by choosing three things: a face f, a splitting plane P based on f, and a half-space H bounded by P. These choices depend on various relationships

among s, t, and the convex region R that contains both s and t. In the implementation, we never need to compute R or represent it explicitly, because all of these properties can easily be computed from the fragments alone. However, the easiest way to explain how the strategy makes its choices is to refer to R as if it were an explicit entity. There are two cases, depending on whether or not R is planar:

1. R is nonplanar. Then we want the plane P that bounds H to contain one of the faces of s or t. If f is any face of s or t, then either $i(f) \subseteq i(R)$ or $f \subseteq b(R)$; and if $f \subseteq b(R)$, then f's normal vector either points outward from R (indicating that the part of S or T bounded by f is inside R) or into R (indicating that the part of S or T bounded by f is outside R). Thus, there are three subcases:

   (a) There is a face f of s or t such that $i(f) \subseteq i(R)$. Then let f be any such face, P be the plane containing f, and H be either of the two closed half-spaces bounded by P.

   (b) For every face f of s or t, $f \subseteq b(R)$, but there is a face f of s or t such that f's normal vector points outward from R. Then let P be the plane containing f, and let H be the closed half-space bounded by P that does not contain R.

   (c) For every face f of s or t, $f \subseteq b(R)$ and f's normal vector points into R. Then let f be any face of s or t, P be the plane that contains f, and let H be the closed half-space bounded by P that contains R.

2. R is planar. Then we want P to be perpendicular to the faces in R and to pass through one of their edges (this is analogous to the task of cutting polygons in 2D). There must be at least one edge e of s or t such that $i(e) \subseteq i(R)$, for otherwise s and t would represent the same point set and each would consist of a single face, and thus our homogeneity test (see Section 3.2) would have succeeded in Step 4 of CLASSIFY. Let e be any such edge, P be the plane perpendicular to R and containing e, and H be either of the two closed half-spaces bounded by P.

In each of the cases above, f, P and/or H are chosen arbitrarily as any face, plane or half-space that satisfies some property. In these cases, the algorithm is correct no matter which choice is made: which plane or half-space to choose is purely an efficiency consideration. We have investigated various heuristic criteria for this choice (see [Van89] for details). However, in testing the performance of our implementation (see Section 5), we did not use such heuristics, but instead had the algorithm choose P and H at random from among the available candidates. This way, we could average its performance over a number of runs, for purposes of curve-fitting.

3.4 Splitting Fragments

Step 5b of CLASSIFY splits s into subfragments representing $s \cap^* H$ and $s -^* H$, and t into subfragments representing $t \cap^* H$ and $t -^* H$. In order to describe how this is done, we will need the definitions presented in the following paragraph.

Any closed half-space H can be uniquely written in the form

$$H = \{(x, y, z) | Ax + By + Cz + D \geq 0\} \tag{9}$$

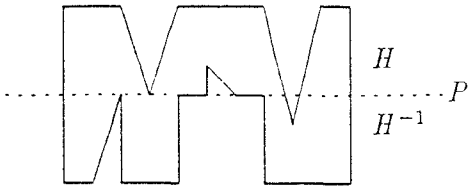


Figure 3: Splitting a face.

where (A, B, C) is a unit-vector normal to H and pointing outward from it, and $|D|$ is the distance from the origin to the plane P that bounds H.[3] For each point $p = (x, y, z)$, we define

$$d(p, H) = Ax + By + Cz + D. \tag{10}$$

Then $p \in H$ if and only if $d(p, H) \geq 0$, and $p \in P$ if and only if $d(p, H) = 0$.

Let F be either s or t. We want to split F into two subfragments $F_1 = F \cap^* H$ and $F_2 = F -^* H$. As described below, we consider each face f of F, and put it into F_1 or F_2, or split it:

1. Initially, set $F_1 = F_2 = \emptyset$.

2. For each face f in the fragment F,

   (a) If $d(v, H) \geq 0$ for every vertex v of f, then $f \subseteq H$, so put f into F_1.

   (b) Otherwise, if $d(v, h) \leq 0$ for every vertex v of f, then $i(f) \subseteq H^{-1}$, so put f into F_2.

   (c) Otherwise, split f into two or more subfaces as described below, such that for each subface g, either $g \subseteq H$ or $i(g) \subseteq H^{-1}$. Put each face g into either F_1 or F_2, as appropriate.

3. At this point, $F_1 = F \cap^* H$ and $F_2 = F -^* H$, so return F_1 and F_2.

Step 2c above requires splitting a face f into two or more subfaces. This is done as follows (a similar method is described in [Hof89]):

1. For each edge e of f, if one vertex u of e is in H and the other vertex v is not, then split e by introducing into f a new vertex w at the point where e intersects P. This point can be computed as the simultaneous solution of the equation $d(w, H) = 0$ and the equation for the line segment $\overline{uv}$.

2. $P \cap f$ consists of one or more line segments. As illustrated in Fig. 3, some of them may already correspond to edges of f, and some may not. To tell these two cases apart, let V be a list of all vertices v of f such that $d(v, H) = 0$, sorted in order along the line of intersection between P and the plane of f. Tracing along this line of intersection,

[3] In our implementation of Protosolid, H is represented by the 4-tuple (A, B, C, D).

we are originally outside f. Every time we encounter a vertex v_i in V, we can tell whether we have entered or exited f by checking to see whether the vertices close to v_i along the boundary of f are on opposite sides of P.[4] If we are inside f and there no edge (v_i, v_{i+1}), then we create one.

4 Implementation

We have implemented the algorithm described in this paper as the basis for performing set operations on B-Reps in a solid modeler called Prototolid. Currently, Protosolid is being used in research projects at several different locations:

1. Protosolid is being used at Purdue University and Cornell University as part of the Newton [NEW90] project. Project Newton is a dynamic simulation system that uses rigid body dynamics to simulate the motion of objects. Hoffmann states in [NEW90] that Protosolid is particularly suitable for the Newton project because

   > it has the additional capability of constructing an object representation especially well-suited to answering efficiently whether two objects interfere, and to delivering the needed geometric data for estimating the moment of collision.

2. At the University of Maryland, Protosolid is used as part of an automated manufacturing project that includes design and process planning. Parts are designed using an user-interface built on top of Protosolid. The design is manipulated by a system for algebraic feature translation [Kar90, KNY91], in order to produce input to a process planning system [Tho89]. Throughout this process, Protosolid is used both to display the part and the features, and to answer several kinds of queries about them.

To describe the details of Protosolid is beyond the scope of this paper, but below we describe a few of the implementation issues.

Protosolid is a B-Rep modeler that uses the "fedge-based" data structure described in [Van89]. Since this data structure was designed to handle non-manifold 3-D polyhedral solids, Protosolid can handle such solids without difficulty. Protosolid's user interface allows the user to build complex solids as combinations of parameterized primitives, using set operations. Currently, Protosolid is written in Common Lisp, and runs on the TI/Explorer and Symbolics Lisp machines—but to improve its portability, we are currently rewriting Protosolid in C++.

Since the input-directed decomposition algorithm was developed for use on polyhedral solids, one issue that arose in implementing Protosolid was how to handle curved surfaces. One standard technique in polyhedral modelers is to represent curved surfaces using faceted approximations—but by itself such an approach is not sufficient for applications such as the automated manufacturing project. The current aproach used in Protosolid is to augment the faceted approximation so that it also includes exact representations of several curved surfaces, including cylindrical, spherical, conical, and toroidal surfaces. Protosolid uses the faceted approximations both for graphics display and for set operations—but each facet has

[4] Interested readers are invited to try this out using Figure 3 as an example.

a pointer to the curved surface it approximates, so that the properties of this surface can be retrieved when needed. For future work, recent results by developed by Shapiro and Vossler [SV90a, SV90b] lead us to believe that our input-directed decomposition algorithm can straightforwardly be extended to handle curved surfaces directly.

5 Efficiency

Since one of the major motivations for our work was to improve the efficiency of boundary classification, we were interested in determining both the worst-case and average-case time complexity of our input-directed decomposition algorithm. However, since there is no clear notion of what constitutes a "random polyhedron", it is not entirely clear what the term "average-case time complexity" means—so doing a direct mathematical analysis of the algorithm's complexity did not appear feasible.

Our solution was to examine the algorithm's time complexity experimentally on several different problems, using the Protosolid solid modeler. In each experiment, the problem was to compute some sequence of regularized union or intersection operations $S_i = T_i \cup^* U_i$ or $S_i = T_i \cap^* U_i$, for $i = 1, 2, \ldots$.

In each experiment we wanted to measure, for each i, the total number of faces n_i in the two input solids and the time t_i taken for boundary classification on these solids. However, the value of t_i is not determined solely by n_i. As discussed in Section 5a, when splitting a region there is often more than one possible choice for what splitting plane to use. A lucky choice can result in fewer subsequent decompositions than an unlucky choice, so t_i can vary significantly depending on which plane is chosen. Thus, to get a good idea how t_i depends on n_i, our approach was as follows: each time the algorithm needed to choose a splitting plane, we had it choose the plane at random from among the available candidates; and we let t_i be the average time required for boundary classification over several computations of S_i. After measuring t_i and n_i in this way for each i, we used least-squares curve-fitting techniques to find a function fitting the data points (n_i, t_i) and having the form

$$t(n) = \sum_k c_{k1} n^k + c_{k2} n^k \log n.$$

We took the big-O complexity of $t(n)$ to be an estimate of the time complexity of our algorithm on this problem.

Below, we summarize the results of four such experiments. The experiments are discussed in more detail in Sections 5.1—5.4.

The first two problems are the "star" and "ring" problems discussed in Sections 5.1 and 5.2. We chose them to represent cases that are "typical" in terms of their demands on the boundary classification algorithm. In both of these cases, curve-fitting on the timing data produced $t(n) = O(n \log n)$.

The third problem is the "fans" problem discussed in Section 5.3. We chose it as an example of a worst-case time-complexity problem for boundary classification. On this problem, curve-fitting on the timing data produced $t(n) = O(n^2 \log n)$. Because of the worst-case nature of this problem, we expect other boundary classification algorithms to have the same time complexity or worse on this kind of problem.

The fourth problem is the "spheres" problem discussed in Section 5.4. We chose this problem for the following reasons. Since input-directed decomposition is a divide-and-conquer algorithm, it performs most efficiently where the problem decomposition produced by splitting along the faces of the solids results in balanced subproblems. But in problems such as the spheres problem, the input-directed decomposition strategy will produce badly unbalanced subproblems, so the time complexity will not be very good. Curve-fitting on the timing data for the spheres problem produced $t(n) = O(n^2)$.

In problems such as the spheres problem, it is possible that other appropriately-chosen decomposition strategies might perform more efficiently than the input-directed decomposition strategy. To get better time complexity on such problems, we are experimenting with a hybrid approach that combines the input-directed decomposition strategy with a regular decomposition strategy. Section 5.5 discusses our experience with one such hybrid algorithm, and its implications for development of more effective hybrid algorithms.

5.1 The "Star" Problem

For the "star" problem, we took an 8-sided faceted cylinder, and subtracted from it a 6-sided faceted cylinder. This produced a hollow tube S_0 centered at the origin and having 16 faces (including the two end faces). Then, for $i = 1, \ldots, 12$, we computed $S_i := S_{i-1} \cup^* S_0^i$, where S_0^i is S_0 rotated $3i$ degrees about the origin. This produced a many-pointed asterisk-shaped solid, as shown in Figure 4. For each i, we let n_i be the total number of faces in the two solids S_{i-1} and S_0^i to be classified, and t_i be the average time required for boundary classification over six computations of S_i.

The data points (n_i, t_i) for this problem closely approximated an $O(n \log n)$ function:[5]

$$t(n) = 0.0029479233 n \log n + 4.2609991.$$

The sum-of-squares value for the fit was $\sum_{i=1}^{12}(t_i - t(n_i))^2 = 1.11872$. Figure 5 graphs the data points (t_i, n_i) and the function $t(n)$.

5.2 The "Ring" Problem

For the "ring" problem, S_0 was a hollow tube similar to the one in the "star" problem, but translated away from the origin. For $i = 1, \ldots, 12$, we computed $S_i := S_{i-1} \cup^* S_0^i$, where S_0^i is S_0 rotated $12i$ degrees about the origin. This produced a solid shaped like a portion of a many-sided ring, as shown in Figure 6.

Just as before, for each i we let n_i be the total number of faces in the two solids S_{i-1} and S_0^i to be classified. But this time, to get good data for curve-fitting, we needed to average t_i over a larger number of runs than in the star problem. In both the ring problem and the star problem, the number of candidate splitting planes is similar—but in the ring problem, if the right planes are chosen, fewer of them are actually needed in order to produce homogeneous fragments. Thus, a lucky choice of splitting planes can compute the boundary classification very quickly, but an unlucky choice of splitting planes can cause the boundary classification to take about as long as it did in the star problem. This means that the time required for boundary classification has a larger variance in the ring problem than it did in the star

[5] As is standard in discussions of time complexity, we use log to mean $\log_2$.

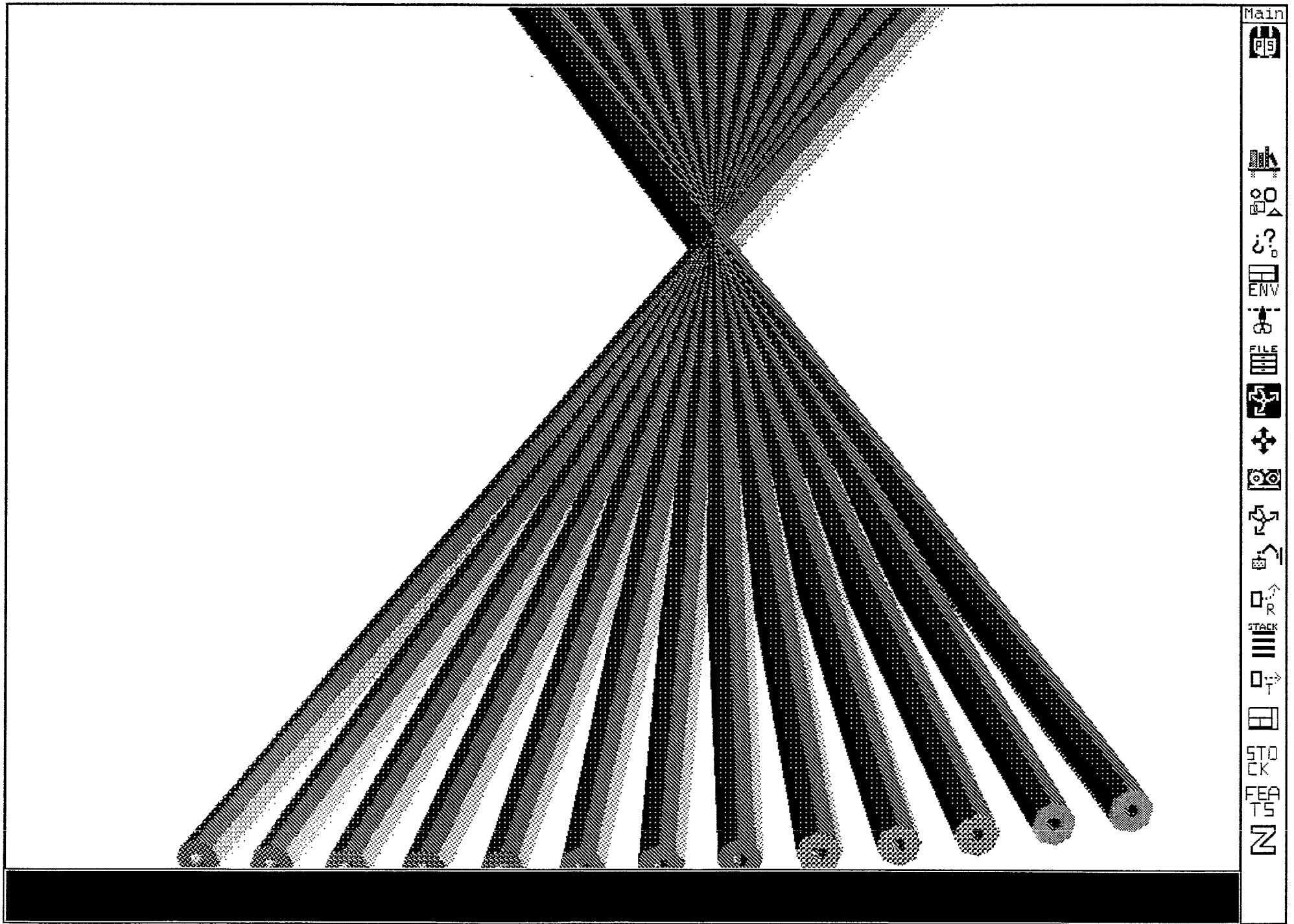
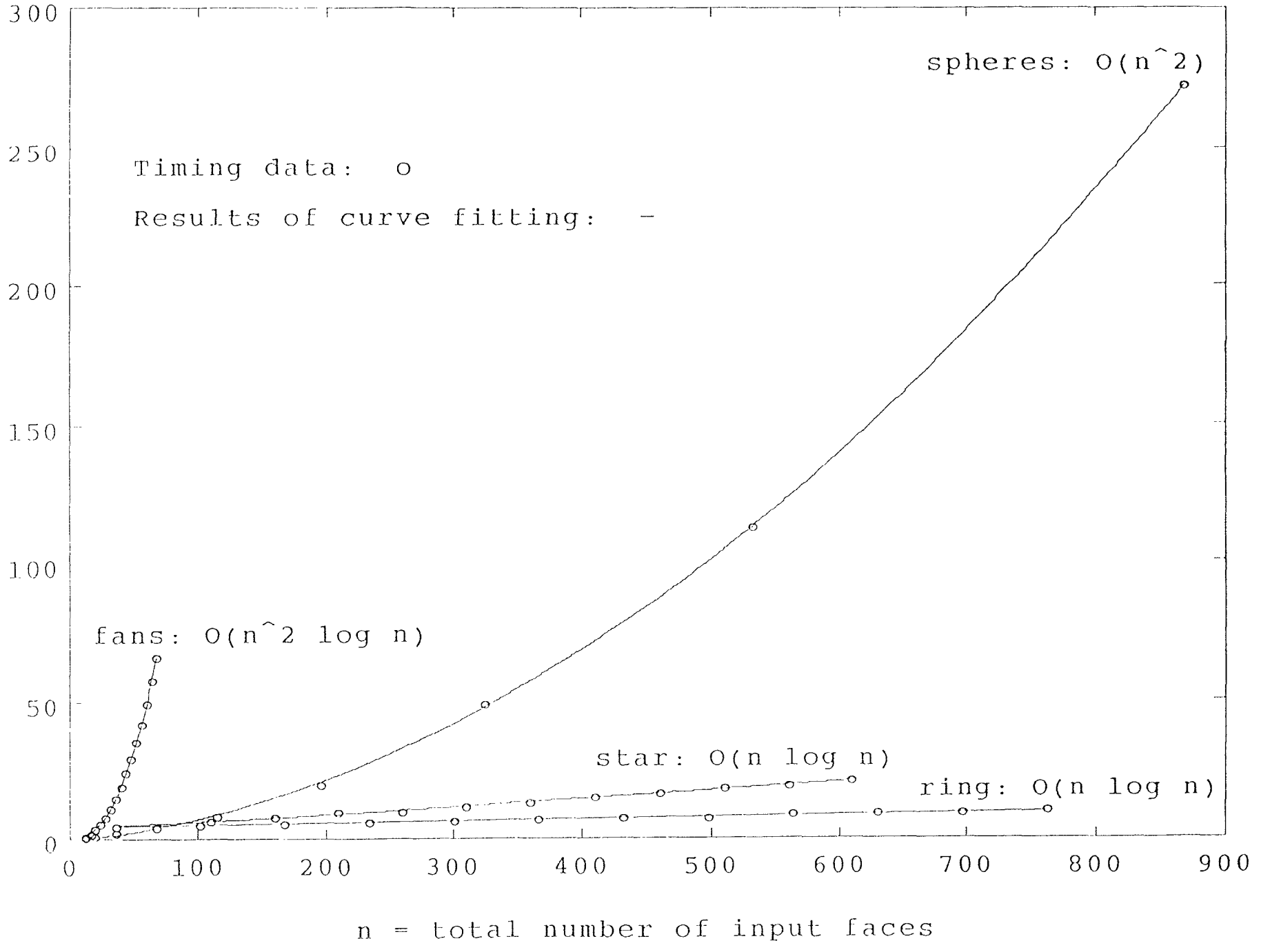


Figure 4: A portion of the solid S_{12} produced by Protosolid for the "star" problem.

Fig. 5: Timing data and curve-fits for input-directed decomposition.



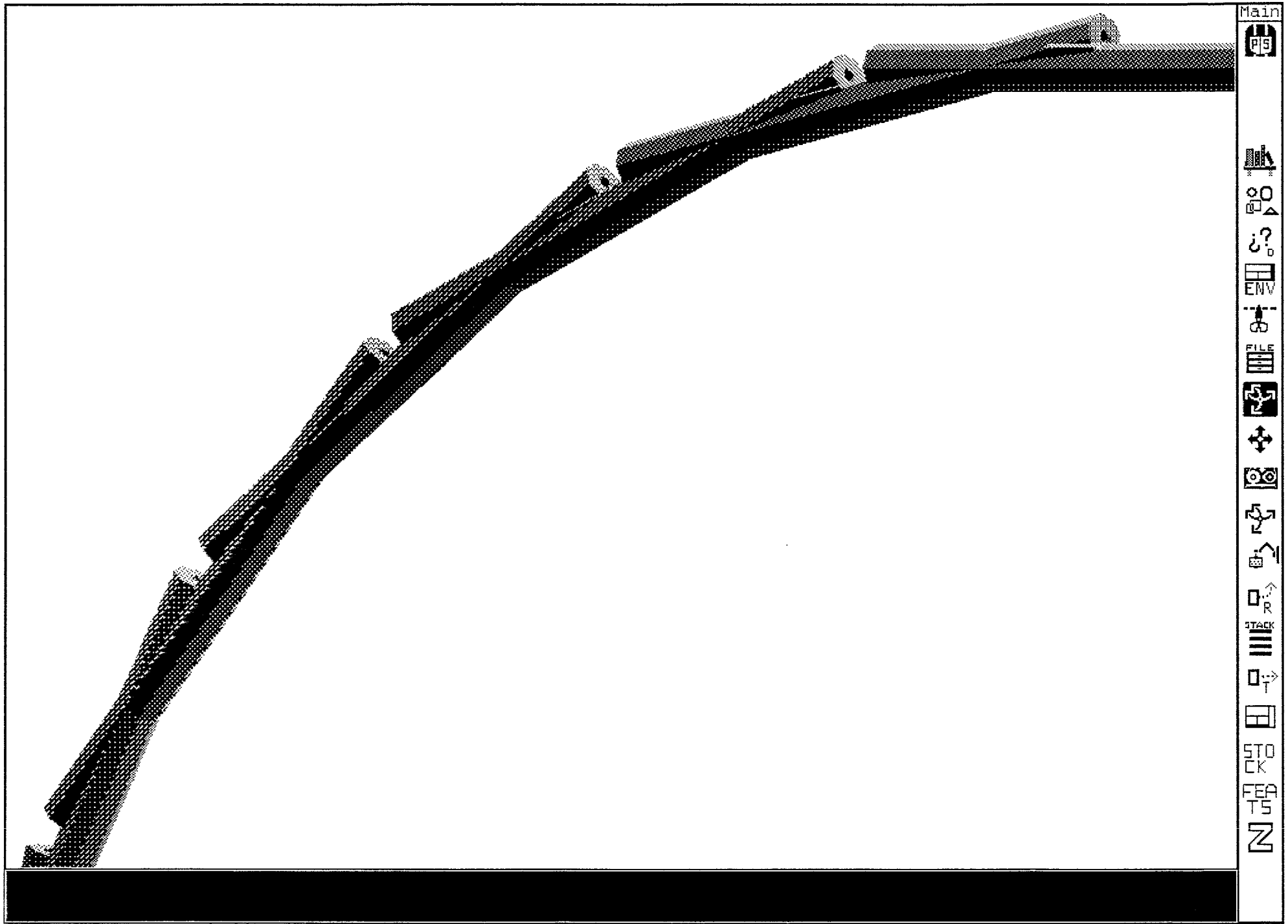


Figure 6: A portion of the solid S_{12} produced by Protosolid for the "ring" problem.

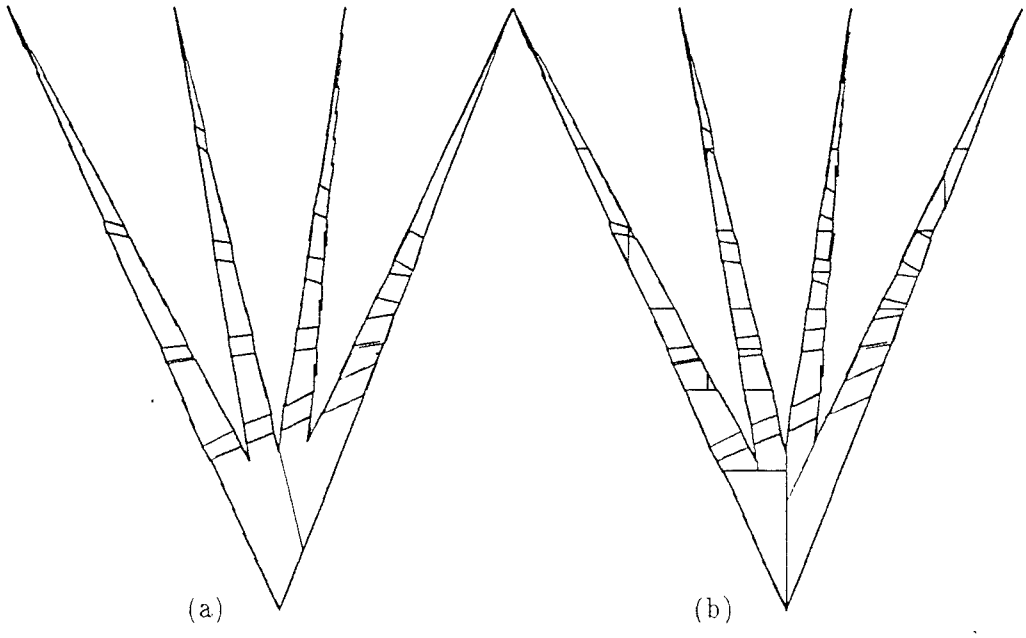


Figure 7: Fragmentation of the top face of a four-fingered fan, produced by boundary classification in the fans example. The subfaces shown in Figure (a) were produced by input-directed decomposition, and those shown in Figure (b) were produced by hybrid decomposition (see Section 5.5).

problem. For this reason, we let t_i be the average time required for boundary classification over 60 computations of S_i, rather than only six computations.

The data points (n_i, t_i) for this problem approximated an $O(n \log n)$ function:

$$t(n) = 0.00072388609 n \log n + 4.6797639.$$

The sum-of-squares value for the fit was $\sum_{i=1}^{12}(t_i - t(n_i))^2 = 0.338124$. Figure 5 graphs the data points (n_i, t_i) and the function $t(n)$.

5.3 The "Fans" Problem

We considered the "fans" problem specifically because it is a worst-case time-complexity problem for boundary classification. In this case, for $i = 2, 3, \ldots, 16$, the problem is to compute the union S_i of two i-fingered fans T_i and U_i, positioned in the $z = 0$ plane in such a way that every finger of T_i intersected every finger of U_i. To do boundary classification in this problem, one must divide the faces of the fans into at least a quadratic number of subfaces, as illustrated in Figure 7(a)). Clearly, this problem requires time at least $\Omega(n^2)$ for all boundary classification algorithms. We believe it actually requires time $\Omega(n^2 \log n)$, because of the overhead involved repeatedly searching for various faces, edges, and vertices—but we have not attempted to prove this.

For each i, we let n_i be the total number of faces in the two solids T_i and U_i to be classified, and t_i be the average time required for boundary classification over four computations

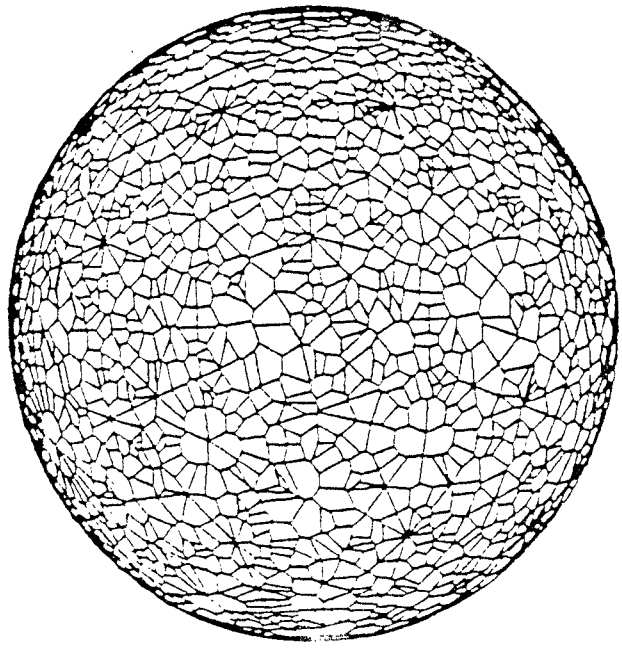


Figure 8: The solid S_{12} in the spheres problem, generated by Protosolid using hybrid decomposition. This solid has 3034 faces, 8236 edges and 5204 vertices.

of S_i. However, we could have obtained good enough data for curve-fitting by averaging each t_i over fewer computations of S_i. The reason is that in this problem, the algorithm does nearly the same number and kind of decompositions (and thus takes nearly the same amount of time) regardless of the order in which the splitting planes are chosen.

For this problem, the data points (n_i, t_i) matched an $O(n^2 \log n)$ function almost perfectly:

$$t(n) = 0.0031549874n^2 \log n - 0.0048458659n^2 = 0.0031549874n^2 \log(0.34485499n).$$

The sum-of-squares value for the fit was $\sum_{i=2}^{16}(t_i - t(n_i))^2 = 0.25016$. Figure 5 graphs the data points (n_i, t_i) and the function $t(n)$.

5.4 The "Spheres" Problem

In some cases, the problem decomposition produced by splitting along the faces of the solids will always result in badly unbalanced subproblems. When this occurs, the input-directed decomposition algorithm will exhibit poor time complexity, even though the time complexity might be better if the decomposition were done in some other manner.[6] One example of this is a problem proposed in [HHK87]: let S_0 be a unit cube centered at the origin, and for $i = 1, 2, \ldots, 12$, compute $S_i = S_{i-1} \cap^* S'_{i-1}$, where S'_{i-1} is a copy of S_{i-1} that has been rotated 45° around the x, y, or z axis, depending on whether $i/3$ has a remainder of 1, 2, or 0, respectively. Figure 8 shows the B-Rep of the solid S_{12}.

For each i, we let n_i be the total number of faces in the two solids S_{i-1} and S'_{i-1} to be classified. Just as in the fans problem, the time required for boundary classification

[6] Analogous behavior occurs in a number of divide-and-conquer algorithms, such as the well-known quicksort algorithm.

in computing S_i was nearly independent of the order in which the splitting planes were chosen—so we let t_i be the average time required for boundary classification over two computations of S_i.

For this problem, the data points (n_i, t_i) closely matched an $O(n^2)$ function:

$$t(n) = 0.00028978021n^2 + 0.0064192472n \log n.$$

The sum-of-squares value for the fit was $\sum_{i=1}^{9}(t_i - t(n_i))^2 = 4.08272$. Figure 5 graphs the data points (n_i, t_i) and the function $t(n)$.[7]

5.5 Hybrid Decomposition

Above, we have examined the performance of input-directed decomposition in several kinds of boundary classification problems. The results can be summarized as follows:

1. In the ring and star problems, input-directed decomposition can localize the face comparisons quite effectively. We believe these problems represent typical cases (for example, in [Man88], Mantyla states that a "typical feature of 'practical cases' is that the effect of a set-operation is *localized* in the three-dimensional space E^{3}").

2. In the fans problems, input-directed decomposition cannot localize the face comparisons very well. But in this kind of problem, no other algorithm can localize the face comparisons either.

3. In the spheres problem, input-directed decomposition cannot localize the face comparisons very well—but this is not a worst-case boundary classification problem. For this kind of problem, it is possible to devise other algorithms that run significantly faster.

One alternative to input-directed decomposition is regular decomposition, in which every region is a parallelepiped, each of whose sides is perpendicular to one of the x, y, and z axes. The initial region is taken to be the smallest such parallelepiped that encloses both solids (this is easily determined from their extents). Whenever a region is split, it is split in half using a splitting plane that is perpendicular to one of the axes.

During boundary classification of two solids S and T, regular decomposition cannot always decompose S and T into homogeneous fragments. However, homogeneous fragments can be produced by combining some regular decomposition steps with some input-directed decomposition steps. We call this approach "hybrid decomposition."

As a preliminary investigation of hybrid decomposition, we have experimented with the following hybrid decomposition algorithm:

> Let c_1 and c_2 be positive integer constants. During the boundary classification of S and T, suppose R is a region that needs to be decomposed, and s and t are the fragments of S and T in R. If R is the result of c_1 decompositions or fewer, and at least one of s and t has more than c_2 faces, then use regular

[7] Since the number of topological entities in S_i increases exponentially with i, for $i \geq 10$ the algorithm required enough memory that paging occurred, invalidating our measurements of t_i for $i \geq 10$. Thus, we have only included data points for $i = 1, 2, \ldots, 9$ rather than $i = 1, 2, \ldots, 12$.

decomposition to decompose R. Otherwise, use input-directed decomposition to decompose R.

As an example, Figure 7(b), shows the kind of fragmentation produced by running this algorithm on two 4-fingered fans with $c_1 = 6$ and $c_2 = 15$. However, for our tests of the hybrid algorithm, we instead used $c_1 = 15$ and $c_2 = 20$.

In the star and fans problems, the hybrid algorithm performed very inefficiently, taking many times the amount of time required by input-directed decomposition. In the ring problem, the hybrid algorithm took more time than input-directed decomposition for small i, but took less time than input-directed decomposition for large i. In the spheres problem, the hybrid algorithm took much less time than input-directed decomposition, but not in a big-O sense.

In the star problem and the fans problem, it is not surprising that the hybrid algorithm performed inefficiently: regular decomposition is unable to decompose space in a way that can localize the face comparisons effectively, so the 15 levels of regular decomposition simply introduced a large amount of overhead into the the boundary classification procedure. In such cases, we would prefer a much smaller number of regular decomposition steps. But in the spheres problem, regular decomposition improves the efficiency—and if we had increased the number of regular decomposition steps as a function of i, we might have been able to improve the big-O complexity. One way to address both of these issues simultaneously would be to alternate regular decomposition steps with input-directed decomposition steps. For future work, we intend to experiment with this approach.

6 Related Work

6.1 Regular Decomposition of Space

Quadtrees and octrees use regular decomposition of space to represent rectilinear solids, but do not provide an exact representation for non-rectilinear solids. Extended octrees provide an efficient and exact representation of polyhedra [Nav87, NFB87]. Isabel Navazo in her Ph.D. thesis [Nav86], and Ayala [ABN85] showed how to perform set operations on extended octrees. Carlbom [Car87] has developed an approach to perform set operations using the polytree data structure, which is an extension of the *exact octree* [BN85, Nav87] data structure. Her techniques use the regular decomposition of space provided by the polytrees to perform set operations.

The main similarities and differences between these approaches and ours are as follows:

1. Each of these approaches proceeds by building data structures to represent localized regions of space, and then using these data structures as the underlying representations for the objects being manipulated, and it is well known that these data structures can become very large for complex objects [Kar88]. Our approach avoids this problem by dealing purely with boundary representations, without converting to and from an alternative representation.

2. The size of a polytree or extended octree corresponds to the number of spatial decomposition steps used to create the tree. In some cases input-directed decomposition

will require a larger number of spatial decompositions, and in some cases it will require a smaller number. We believe that the best way to get good performance over a wide variety of problems is to combine regular decomposition with input-directed decomposition (as we have described in Section 5.4).

3. Set operations on extended octrees or polytrees involve a large number of cases. For example, in a polytree there are five different types of cells—so in order to perform set operations, there are at least twenty-five different combinations to handle. By using input-directed decomposition and by manipulating the B-Reps instead of polytree cells, we avoid this proliferation of cases.

Woodwark [Woo86] has used partitioning of space in order to determine the wireframe of a solid object from a set-theoretic (half-space) model. Woodwark's approach is like ours in the sense that it decomposes space without building a tree structure to represent the decomposition—but the main distinctions of our approach from that of Woodwark are as follows:

1. Woodwark uses regular decomposition of space, whereas we use a combination of regular and input-directed decomposition. Thus, our previous comments about the number of spatial decomposition steps are applicable here as well.

2. The problem Woodwark addressed is that deriving the wire-frame representation of an object from its set-theoretic description. The problem of computing set operations would require considerable extensions to Woodwark's partioning technique.

The EXCELL scheme used by Tamminen [Tam81] uses additional data structures to represent a solid object. To begin with, the space occupied by an object is divided into extendible cells that contain several data pages, one for each kind of topological entity. The idea is not just to partition the space into cells of equal size, but to have the size of a cell vary with the sparsity of the data. The primary advantage of such data structures is the following: in order to perform set operations, one first localizes the region of space where the operation (say intersection) occurs. Suppose we have to intersect a face of an object X with the faces of an object Y. Now, instead of doing the intersection with all the faces of Y, one can do it just with the cells of Y and only if the face-cell intersection is positive, we have to proceed further. This can result in considerable savings in time for two reasons:

1. A face-cell intersection is faster than a face-face intersection,

2. The underlying assumption here is that the set operations are localized to a small region and therefore, a face-cell intersection will be positive only in a small percentage of the cases. A worst-case scenario for the localization assumption is the *fans* example.

This is similar to the regular decomposition used in our approach. One drawback of the EXCELL approach is the overhead involved in continuously updating the directory structures, cells, and the data pages, etc. used by this scheme.

6.2 Nonregular Decomposition of Space

Nonregular decomposition of space was first used by Fuchs in his presentation of Binary Space Partitioning (BSP) trees to represent polyhedra for the use in a hidden surface algorithm [FKN80]; the complexity of this approach was later analyzed by Paterson and Yao [PY89]. The use of BSP trees has been extended by Thibault and Naylor to allow for the conversion from CSG to BSP tree to B-Rep, and to allow for the computation of set operations between a BSP tree and a CSG primitive [TN87]. Considerable research is underway to use BSP trees as an alternative [Nay90] representation scheme for solids.

In building a BSP tree, one decomposes space by partitioning it along the faces of the solids being represented—and we decompose space in a very similar manner. However, with BSP trees one builds data structures to represent localized regions of space, and then uses these data structures as the underlying representations for the objects being manipulated. In contrast, our algorithm deals purely with boundary representations, without converting to and from an alternative representation. However, since our scheme repeatedly decomposes space in the same manner that BSP trees do, it can easily be modified to produce an explicit BSP tree after the set operations have been performed—and we have recently extended Protosolid to do this.

6.3 Other Related Work

B-Rep data structures have been around for nearly two decades, since Baumgardt's [Bau72] winged-edge data structure. Since then, several researchers have developed various types of B-Rep data structures, including the bridge-edge data structure of Yamaguchi and Tokieda [YT85], the half-edge data structure of Mantyla [Man88], and the non-manifold data structures of Weiler [Wei86] and Karasick [Kar88].

For our implementation of input-directed decomposition in the Protosolid solid modeler [Van89], we developed a "fedge-based data structure" that is very similar to the star-edge representation used by Karasick. The primary differences are that the fedge-based data structure uses bridge edges and separates geometry from topology, and the star-edge representation doesn't. Although our data structures are similar to Karasick's, the input-directed decomposition algorithm we use to manipulate our data structures is very different from Karasick's algorithm for computing the intersection of two solids. Karasick's algorithm does not do any decomposition of space, but instead classifies every face of a solid with respect to the other solid, thus incurring a typical-case time complexity of $O(n^2 \log n)$. Also, for all points that lie on the same plane of one solid, Karasick's algorithm tries to classify them with respect to another solid—which leads to a proliferation of special cases, especially because of isolated vertices. This problem does not arise in our algorithm.

7 Conclusion

In this paper, we have presented a divide-and-conquer method for boundary classification. The basic algorithm is based on recursively decomposing space based on the boundaries of the solids being classified—an approach which we call input-directed decomposition. BSP trees [Nay90] decompose space in a very similar manner—but in contrast to BSP trees,

our algorithm operates directly on B-Reps without converting to and from an alternative representation. Our algorithm has been proved correct in [Van89], and it can easily handle both manifold and non-manifold 3-D solids.

Input-directed decomposition provides the basis for set operations in our Protosolid solid modeler. Protosolid is being used in several projects, at the University of Maryland [KNY91], Purdue University [NEW90, Van91a, Van91b], and Cornell University [NEW90]. Using Protosolid, we have performed experiments to evaluate the time complexity of input-directed decomposition. The experimental results indicate that its worst-case time complexity is $O(n^2 \log n)$, and its typical-case time complexity is $O(n \log n)$.

For future work on input-directed decomposition, we have several topics in mind:

1. As described in this paper, input-directed decomposition does not work on curved surfaces. However, it appears that some of the methods developed by Shapiro and Vossler [SV90a, SV90b] to convert B-Reps into CSG can be adapted for use with input-directed decomposition, to extend it to handle curved surfaces. This provides a promising direction for future research.

2. We have done preliminary investigations of a hybrid decomposition strategy (see Section 5.5) that combines input-directed decomposition with regular decomposition of space. Our current version of hybrid decomposition takes less time than input-directed decomposition on some problems and more time on others. We believe that a more sophisticated version of the hybrid approach may turn out to give the best overall performance over a wide variety of problems, and we intend to investigate this further.

3. The good performance of Protosolid on problems such as Karasick's spheres problem [HHK87] was due to careful implementation and the inherent properties of the decomposition process, rather than through any particular methods for achieving robustness. However, we are currently involved in further research on the issue of robustness, and intend to incorporate this work into future versions of Protosolid.

References

[ABN85] D. Ayala, P. Brunet, and I. Navazo. Object Representation by Means of Nonminimal Division Quadtrees and Octrees. *ACM Transactions on Graphics*, 4(1):41–59, January 1985.

[Bau72] B. Baumgardt. Winged-Edge Polyhedron Representation. Technical Report CS–320, Computer Science Department, Stanford University, 1972.

[BN85] P. Brunet and I. Navazo. Geometric Modelling Using Exact Octree Representation for Polyhedral Objects. In *Proceedings of Eurographics 1985, North Holland, Amsterdam 1985*, pages 159–169, 1985.

[Car87] Ingrid Carlbom. An Algorithm for Geometric Set Operations Using Cellular Subdivision Techniques. *IEEE Computer Graphics and Applications*, pages 44–55, May 1987.

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On Visible Surface Generation by a priori Tree Structures. In *Proceedings of SIGGRAPH-87*, pages 124–133, July 1980.

[GS85] L. Guibas and J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computations of Voronoi Diagrams. *ACM Transactions on Graphics*, pages 74–123, April 1985.

[HHK87] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Robust Set Operations on Polyhedral Solids. Technical Report TR-87-875, Department of Computer Science, Purdue University, 1987.

[Hof89] Christoph M. Hoffmann. *Geometric and Solid Modeling. An Introduction.* Morgan Kaufmann Publishers, Inc., 1989.

[Kar88] Michael Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, Department of Computer Science, McGill University, Aug 1988.

[Kar90] Raghu Karinthi. *An Algebraic Approach to Feature Interactions.* PhD thesis, Department of Computer Science, University of Maryland, College Park, December 1990.

[KM76] K. Kuratowski and A. Mostowski. *Set Theory.* North Holland Publishing Company, 1976.

[KNY91] Raghu Karinthi, Dana Nau, and Qiang Yang. Handling feature interactions in process planning, 1991. To be Published in the Journal of Applied Artificial Intelligence.

[Man88] M. Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, College Park MD, 1988.

[Men75] B. Mendelson. *Introduction to Topology.* Allyn and Bacon, Inc., 1975.

[Nav86] I. Navazo. *Geometric Modelling of Octree Encoded Polyhedral Objects.* PhD thesis, Departament de Metodes Informatics, Universitat Politecnica de Catalunya, January 1986.

[Nav87] I. Navazo. Extended Octree Representation of General Solids with Plane Faces: Model Structure and Algorithms. *Computers and Graphics*, July 1987.

[Nay90] Bruce Naylor. Binary Space Partitioning Trees as an Alternative Representation of Polytopes. *Computer-Aided Design*, 22(4), May 1990.

[NEW90] Computing about Physical Objects. Purdue University, Department of Computer Science, Spring 1990.

[NFB87] I. Navazo, J. Fontdecaba, and P. Brunet. Extended Octrees, between CSG Trees and Boundary Representations. In *Eurographics-87*, 1987.

[PRS86] A. Paoluzzi, M. Ramella, and A. Santarelli. UN Modellatore Geometrico su Rappresentazioni Triango-Alate. Technical Report 13.86, Disp. Informatica e Sistemistica, Univ. of Rome, Rome, Italy, 1986.

[PY89] M. S. Paterson and F. F. Yao. Binary Partitions with Applications to Hidden-Surface Removal and Solid Modeling. In *Proceedings of ACM on Computational Geometry*, pages 23–32. ACM, 1989.

[Req77] A. G. Requicha. Mathematical Models of Rigid Solid Objects. Technical Report TM–28, Production Automation Project, University of Rochester, Nov 1977.

[Req80] A. A. Requicha. Representation of Solid Objects–Theory, Methods and Systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[RT78] A. G. Requicha and R. Tilove. Mathematical Foundations of Constructive Solid Geometry : General Topology of Closed Regular Sets. Technical Report TM–27a, Production Automation Project, University of Rochester, June 1978.

[SV90a] V. Shapiro and Donald L. Vossler. B-Rep -¿ CSG Conversion I: Construction and Optimization of CSG Representations. Technical Report CTA89-3A, Sibley School of Mechanical and Aerospace Engr., Cornell University, September 1990.

[SV90b] V. Shapiro and Donald L. Vossler. B-Rep -¿ CSG Conversion II: Efficient CSG Representation of Planar Solids. Technical Report CTA89-4A, Sibley School of Mechanical and Aerospace Engr., Cornell University, May 1990.

[Tam81] M. Tamminen. The EXCELL Method for Efficient Geometric Access to Data. *Acta Polytechnica Scandinavica, Mathematics and Computer Science Series*, 34, 1981.

[Tho89] Scott Thompson. Environment for Hierarchical Abstraction: A User Guide, May 1989. Master's Scholarly paper, University of Maryland, Department of Computer Science.

[Til80] Bruce Tilove. Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE Transactions on Computers*, C-29(10), October 1980.

[TN87] W. C. Thibault and B. F. Naylor. Set Operations on Polyhedra Using Binary Space Partitioning Trees. In *Proceedings of SIGGRAPH-87*, pages 53–162, July 1987.

[Van89] G. Vanecek Jr. *Set Operations on Volumes Using Decomposition Methods*. PhD thesis, University of Maryland, College Park, 1989.

[Van91a] G. Vanecek. BRep-Index: A Multi-Dimensional Spatial Partioning Tree. In *First ACM/SIGGRAPH Symposium on Solid Foundations and CAD/CAM Applications*, page to appear, June 1991.

[Van91b] G. Vanecek. A Data Structure for Analyzing Collisions of Moving Objects. In *Proc. IEEE 24th Hawaii Internat. Conf. on System Sciences*, volume 1, pages 671–680, January 1991.

[Wei86] Kevin Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1986.

[Woo86] J R Woodwark. Generating Wireframes from Set-Theoretic Solid Models by Spatial Division. *Computer-Aided Design Journal*, 18(6):307–315. Jul/Aug 1986.

[YT85] F. Yamaguchi and T. Tokieda. Bridge Edge and Triangulation Approach in Solid Modeling. In Tosiyasu L. Kunii, editor, *Frontiers in Computer Graphics '84*, pages 44–65. Springer-Verlag. 1985.