JOURNAL OF ALGORITHMS 8, 348–361 (1987)

# Space Searching for Intersecting Objects*

DAVID P. DOBKIN[†]

*Department of Computer Science, Princeton University, Princeton, New Jersey 08544*

AND

HERBERT EDELSBRUNNER[‡]

*Institutes for Information Processing, Technical University of Graz, Schiessstattgasse 4a, A-8010 Graz, Austria*

Received May 30, 1985

Determining or counting geometric objects that intersect another geometric query object is at the core of algorithmic problems in a number of applied areas of computer science. This article presents a family of space-efficient data structures that realize sublinear query time for points, line segments, lines and polygons in the plane, and points, line segments, planes, and polyhedra in three dimensions. © 1987 Academic Press, Inc.

## 1. INTRODUCTION

A fundamental problem in geometric complexity concerns the detecting, counting, and/or reporting of intersections among members of a collection of objects. We consider here instances of this problem where a set of objects is to be preprocessed for multiple queries against objects from a different set. This problem is considered as a searching problem specified by two domains. The first is the search domain defining the sets of objects to be searched (e.g., line segments in the plane, halfspaces, . . . ). Next, the query domain defines those objects which are to be tested against this set for intersection.

*This work was performed while the first author was visiting at the Technical University of Graz during May and June 1983.

[†]Research of this author was supported in part by the National Science Foundation under Grant MCS83-03926 and DCR85-05517.

[‡]Research of this author was supported by the Austrian Fonds zur Foerderung der wissenschaftlichen Forschung. Present address: Department of Computer Science, University of Illinois, Urbana, Illinois 61801.

Given the specification of search and query domains, our goal is an algorithm for quickly preprocessing elements of the search domain to allow for efficient processing of queries involving elements of the query domain. Here, a query asks for a report of either the number (*the counting problem*) or names (*the reporting problem*) of objects intersected. Efficient processing is defined as a preprocessing algorithm requiring low-degree polynomial time which creates a search structure requiring linear space and answering queries in sublinear time.

In all cases, there are algorithms requiring no preprocessing which have linear search time and algorithms with logarithmic search time which require nonlinear storage for their search structure [EKM]. Only for the case handled by range-searching trees are better results known. Here the search domain consists of points of Euclidean space $E^d$, $d = 2, 3$; the query domain consists of individual halfplanes or halfspaces or simplices formed as intersections of $d + 1$ or fewer halfspaces; and a query asks how many (or which) points lie in the query simplex. This problem has been widely studied [EW, W, Y, YDE] in the past. We extend the space-cutting trees generated in range-searching algorithms to a class of recursive space-cutting trees. From these trees there evolve algorithms which apply to broader and more interesting search and query domains. Our domains allow line segments, polygons, tetrahedra, and other objects as we make precise below.

The best known results involving space-cutting trees are search algorithms using a polynomial amount of time in preprocessing and requiring linear space and $O(n^{a(d)})$ search time in $E^d$. Our results build upon the existence of these algorithms and we use $a(2)$ and $a(3)$ throughout to represent the current best values of these constants. At present, $a(2) = 0.695$ and $a(3) = 0.8999$ [EW, YDE].

The results of this paper are summarized by the following theorem:

THEOREM. *Let $S$ represent a search domain, $Q$ a query domain, and $q$ a query type. For a subset of $S$ of size $n$ and any real $\varepsilon > 0$, there is a linear space search structure constructible from polynomial-time preprocessing, which allows queries of type $q$ to be answered in time $T(n)$ for the following cases ($I$ always denotes the number of objects reported)*:

(a) *$S$ = triangles in $E^2$, $Q$ = points in $E^2$, $q$ = how many (resp. which) triangles of $S$ does a point intersect, $T(n) = O(n^{a(2)})$ (resp. $O(n^{a(2)+\varepsilon} + I)$).*

(b) *$S$ = segments and lines in $E^2$, $Q$ = segments in $E^2$, $q$ = how many (resp. which) segments/lines does a segment intersect, $T(n) = O(n^{a(2)+\varepsilon})$ (resp. $O(n^{a(2)+\varepsilon} + I)$).*

(c) *$S$ = polygons of $m$ or fewer sides in $E^2$, $Q$ = polygons of $m$ or fewer sides in $E^2$, $q$ = how many (resp. which) polygons intersect a query polygon, $T(n) = O(n^{a(2)+\varepsilon})$ (resp. $O(n^{a(2)+\varepsilon} + I)$).*

(d) *$S$ = tetrahedra in $E^3$, $Q$ = points of $E^3$, $q$ = how many (resp. which) tetrahedra contain a query point, $T(n) = O(n^{a(3)})$ (resp. $O(n^{a(3)+\varepsilon} + I)$).*

(e) *$S$ = planes in $E^3$, $Q$ = line segments in $E^3$, $q$ = how many (resp. which) planes intersect a query segment, $T(n) = O(n^{a(3)})$ (resp. $O(n^{a(3)} + I)$).*

(f) *$S$ = segments in $E^3$, $Q$ = planes in $E^3$, $q$ = how many (resp. which) segments intersect a query plane, $T(n) = O(n^{a(3)})$ (resp. $O(n^{a(3)+\varepsilon} + I)$).*

These results are achieved by introducing search trees which are composed of the trees used for range searching.

We begin by summarizing the range searching trees (and known results) we will be using. Next, we describe our methods for combining range searching trees to extend these methods. The former is done in the next section and the latter in Section 3. Finally, we show how these results can be applied to the intersection problems described in our main theorem above.

## 2. SPACE-CUTTING TREES

Let $P$ be a finite multiset of points in $E^d$; $d = 1, 2, 3$; and $h$ be a directed hyperplane (i.e., the normal to $h$ gives a positive and negative sense). If $h^+$ and $h^-$ represent the open halfspaces on the positive and negative sides of $h$, then $P(h^+)$, $P(h^-)$, and $P(h)$ are the subsets of $P$ in $h^+$, $h^-$, and $h$. We define a $d$-dimensional space-cutting tree as follows:

> The root of $T$ contains a hyperplane $h$ and its three children are space-cutting trees for the sets $P(h^+)$, $P(h^-)$, and $P(h)$, if these sets are nonempty. The first two of these are $d$-dimensional and the third is $(d-1)$-dimensional. $h$ is chosen so that $|P(h^-)| \leq a|P|$ and $|P(h^+)| \leq (1-a)|P|$, for fixed $a$ with $0 < a < 1$ (we will use $a = 1/2$ here).

A node, $v$, of $T$ is termed $k$-dimensional if the path from the root to $v$ contains $d$-$k$ branches to middle sons (i.e., to $P(h)$ sets). We define the domain of $v$, denoted by dom($v$), as the intersection of regions on the path from the root to $v$; that is, the domain of the root is the whole space. If $v$ is a son of $w$ and $h$ is the hyperplane stored at $w$, then dom($v$) is defined as dom($w$) $\cap\, h^-$, dom($w$) $\cap\, h$, or dom($w$) $\cap\, h^+$, depending on whether $v$ is the left, middle, or right son of $w$.

Using these trees, the standard query is a simplex query where we wish to determine how many of a preprocessed set of points lie in a simplex $s$.

Here, the following search strategy arises:
Let $v$ be the current node. (Initially, $v$ is the root of $T$.)

Case 1. If $s$ contains $\mathrm{dom}(v)$ report all points in the subtree.

Case 2. If $s$ and $\mathrm{dom}(v)$ do not intersect return without any further action.

Case 3. If $s$ intersects $\mathrm{dom}(v)$ but does not contain it, visit the sons of $v$ recursively.

We will be concerned here with the space-cutting trees which arise from known range searching algorithms in 1, 2, and 3 dimensions. These trees admit linear storage, (low-degree) polynomial preprocessing, and sublinear query time. For 1-dimensional range-searching algorithms, we use binary search algorithms with running times of $O(\log n)$. These are called *b-trees* in what follows (not to be confused with the B-trees of [BM]). For 2-dimensional range searching, space-cutting trees arising from conjugation trees [EW] of query time $O(n^{a(2)})$ are used (currently, $a(2) = 0.695$). These are called *c-trees*. Three-dimensional range searching relies on the space-cutting trees of [Co, DE] which we call *d-trees*. These trees have query time $O(n^{a(3)})$ (currently, $a(3) = 0.8999$). In the appendix we argue that $d$-trees indeed support simplex queries within the same complexity as halfspace queries.

## 3. Recursive Space-Cutting Trees

We are now ready to introduce recursive space-cutting trees. To form these trees, we combine space-cutting trees by decomposing the underlying Euclidean search space as we show below. Before proceeding, we define terms necessary to our derivation.

Let $P$ be a finite set of points in $E^d$ and let $d = d_1 + d_2$, $d_1 \geq 1$, and $d_2 \geq 0$. If we let $F_1$ (resp. $F_2$) denote the flat spanned by the first $d_1$ (resp. last $d_2$) coordinate axes, then, $P(F_i)$ will denote the orthogonal projection of $P$ onto $F_i$, for $i = 1, 2$. We consider $P(F_1)$ and $P(F_2)$ as multisets. That is, even if two points have the same projection in $F_i$, they are considered to be different in $P(F_i)$, for $i = 1, 2$. Finally, an internal node of a tree is said to be on level $m$, if the path from the root to the node has $m - 1$ branches (i.e., involves $m - 1$ decisions).

Let $T_1$ be a $d_1$-dimensional space-cutting tree of $P(F_1)$ and let $T_2$ be a $d_2$-dimensional recursive space-cutting tree. Then, a *$d$-dimensional $k$-sparse recursive space-cutting tree $T$ of $P$* ($k > 0$) is defined as follows (if $d_2 > 0$, otherwise $T = T_1$):

> For each $i > 0$ and each node $v$ on level $ik$, let $P_v$ be the subset of $P$ such that the subtree with root $v$ stores $P_v(F_1)$. Then $T$ consists of $T_1$ with each node $v$ on level $ik$ ($i = 1, 2, \dots, \text{depth}(T_1)/k$), augmented with an instance of $T_2$ storing $P_v(F_2)$.

Let $T_1$ (resp. $T_2$) accommodate the range query for query simplex $s_1$ (resp. query range $s_2$). $s_2$ is either a $d_2$-dimensional simplex or the Cartesian product of lower dimensional simplices. Then $T$ can be used to answer the range query for the Cartesian product $s$ of $s_1$ and $s_2$. The search strategy for $s$ searches $T_1$ with $s_1$ using the standard search strategy outlined in Section 2. Let $v$ be a node visited during this search. If $\text{dom}(v)$ is contained in $s_1$ (Case 1 in the standard search strategy) then two cases are distinguished: If $v$ is on level $ik$, for some integer $i$, then the associated instance of $T_2$ is searched recursively with $s_2$. Otherwise, then at most three sons of $v$ are visited recursively until a level $ik$ is reached.

The time and space requirements for $T$, a *d-dimensional k-sparse recursive space-cutting tree T of P* are given by:

LEMMA. *Let the storage and query time required by $T_1$, $T_2$, $T$ be $S_1(n)$, $S_2(n)$, $S(n)$ and $Q_1(n)$, $Q_2(n)$, $Q(n)$. Then $S(n) = (c_1 \log n/k)S_2(n) + S_1(n)$, for some positive constant $c_1$. And, $Q(n) = Q_1(n) + (3^{-1})Q_2(n/(3^{k-1}))\log n$.*

*Proof.* The space result follows because the depth of $T_1$ is $O(\log n)$ and an instance of $T_2$ occurs only on every $k$th of those levels. Also, the space required by $T_1$ is at most linear. For the query time, we can assume that all three subtrees of a node are of equal size. This assumption is valid since $Q_2(n)$ is at most linear.

In what follows, we choose $k$ to make $S(n)$ linear while not significantly increasing $Q(n)$. For example, the choice $k = \varepsilon \log_3 n$ for some $\varepsilon > 0$ always works.

Recursive space-cutting trees are denoted by words over the alphabet $b, c, d$ with choices of sparsity made to optimize query time while guaranteeing linear space. We demonstrate this by the following examples:

EXAMPLE 1. A $b^2$-tree $T$ can be used to store a set of $P$ of $n$ points in $E^2$. By definition, $T$ is a $b$-tree $T_1$ storing the points with respect to their $x_1$-coordinates. The nodes $v$ on distinguished levels of $T_1$ are augmented with $b$-trees storing $P_v$ with respect to $x_2$-coordinates. The kind of queries supported by $T$ involve axis-parallel rectangles (that is, Cartesian products of two one-dimensional simplices) as query range. Let $k$ be the sparsity of $T$, that is, only the nodes on levels $ik$, for positive integers $i$, are augmented.

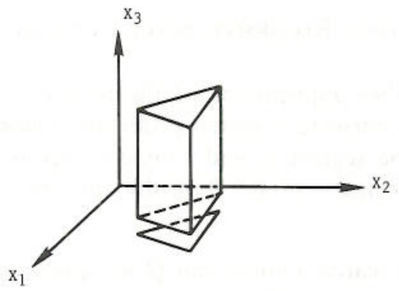

FIG. 1. Query range for a *cb*-tree. The *c*-tree discriminates w.r.t. the $x_1x_2$-plane, the *b*-tree discriminates w.r.t. the $x_3$ coordinates.

Then $T$ requires $O(n \log n/k)$ space. The query time satisfies $Q(n) = Q(n/2) + 3^{k-1}O(\log(n/3^{k-1}))$.

It is worthwhile to note that, for $k = 1$, $T$ is essentially a two-dimensional range tree as developed in [B]. Then $T$ requires $O(n \log n)$ space and realizes $O(\log^2 n)$ query time. In [B] a technique is described that constructs $T$ in $O(n \log n)$ time. Here we use linear space and $O(n^\varepsilon)$ query time for any positive $\varepsilon$.

EXAMPLE 2. A *cb*-tree $T$ stores a set $P$ of $n$ points in $E^3$. Let $k$ be the sparsity of $T$. Then $T$ is a *c*-tree $T_1$ with each node on level $ik$, for positive integers $i$, augmented with a *b*-tree. $T$ requires $O(n \log n/k)$ space. The range queries supported by $T$ involve cylindric query ranges that are the Cartesian product of two-dimensional and one-dimensional simplices in the $x_1x_2$-plane and $x_3$-axis, respectively (see Fig. 1).

The time required to answer such a range query satisfies $Q(n) = O(n^{a(2)}) + 3^{k-1}O(\log(n/3^{k-1}))$. Now, $Q(n) = O(n^{a(2)})$ provided $3^{k-1}O(\log(n/3^{k-1})) = O(n^{a(2)}/\log n)$. Choosing $k$ as above yields linear space and $O(n^{a(2)})$ query time.

More generally, we state

THEOREM. *Let $T$ be a u-tree of $P$, with u in $\{b, c, d,\}^+$. Let $u = aw$, with a in $\{b, c, d\}$ and w in $\{b, c, d,\}^*$. For every positive real number $\varepsilon$ there exists a choice of sparsity such that $T$ needs $O(n)$ space and $Q(n)$ query time with*

(i) $Q(n) = O(n^\varepsilon)$ if $a = b$ and $w$ in $\{b\}^+$,
(ii) $Q(n) = O(n^{a(2)})$ if $a = c$ and $w$ in $\{b\}^*$,
(iii) $Q(n) = O(n^{a(2)+\varepsilon})$ if $a = c$ and $w$ in $\{b\}^*\{c\}\{b, c\}^*$,
(iv) $Q(n) = O(n^{a(3)})$ if $a = d$ and $w$ in $\{b, c\}^*$, and
(v) $Q(n) = O(n^{a(3)+\varepsilon})$ if $a = d$ and $w$ in $\{b, c\}^*\{d\}\{b, c, d\}^*$.

## 4. Applying Recursive Space-Cutting Trees

This section describes applications of the recursive space-cutting trees developed above to intersection search problems. These problems involve points, lines, rays, line segments, and simple polygons in $E^2$ or $E^3$, and planes and convex polyhedra in $E^3$. Basically, our approach consists of the three steps:

(1) Let $S$ be the search domain and $Q$ the query domain. Each object in $S$ and each query object in $Q$ is decomposed into simpler parts. Thus $S$ is written as $S_1 S_2 \cdots S_t$ and $Q$ as $Q_1 Q_2 \cdots Q_t$. The query on $S$ involving query object $q$ is done by querying $S_i$ by the projection of $q$ onto $Q_i$ and combining the partial answers obtained.

(2) A transform $M$ is established that maps $S_i$ into $E^d$, for appropriate $d$, and $Q_i$ into $R^d$, where $R^d$ is the space of ranges in $E^d$. $M$ is chosen so that a query object $q$ in $Q_i$ intersects $a$ in $S_i$ if and only if the point $M(a)$ belongs to the range $M(a)$.

(3) $M(S_i)$ is stored in one or several space-cutting trees that permit range queries with query ranges out of $M(Q_i)$.

If the intersecting objects are to be reported, Step (1) of our approach is to represent a (query) object as the union of simpler (query) objects. If the intersections are to be counted, we take advantage of the ability to subtract numbers by using more involved representations involving (set) differences of (query) objects. Here, we must apply the restriction that the subtracted (query) object must be contained in the (query) object from which it is subtracted.

The most common transformation we use in step (2) is the dual transform. Let $h$ be the nonvertical hyperplane satisfying $x_d = h_1 x_1 + \cdots + h_{d-1} x_{d-1} + h_d$, and let $p$ be the point $(p_1, \ldots, p_d)$ of $E^d$. Then, the dual of $h$ is the point $D(h) = (h_1, \ldots, h_d)$ and the dual of $p$ is the hyperplane $D(p)$ satisfying $x_d = -p_1 x_1 - \cdots - p_{d-1} x_{d-1} + p_d$. The usefulness of the dual transform $D$ for algorithmic aims ([Br, Ch, EKM]) stems from the observation:

Fact. A point $p$ is above, on, or below a hyperplane $h$ if and only if the point $D(h)$ is below, on, or above the hyperplane $D(p)$, respectively.

We are now ready to prove our main theorem.

*Case* a. "The point in triangles problem." We begin by observing that any triangle can be decomposed into two triangles each having a vertical edge. The common edge is assigned to the left subtriangle to make the original triangle the disjoint union of the two smaller triangles. Note that in the degenerate case where the triangle has a vertical edge, the left triangle may be the degenerate triangle of one edge or the right triangle may be empty.

Without loss of generality, we can now consider the subproblem where each triangle in the set, $L$, of left triangles is to the left of its vertical edge. Such a triangle $u$ is the intersection of three halfplanes:

$$x_2 \geq a_1 x_1 + a_2,$$
$$x_2 \leq a_3 x_1 + a_4,$$
$$x_1 \leq a_5.$$

A point $p = (p_1, p_2)$ in the plane is contained in this triangle if and only if

$$a_2 \leq -p_1 a_1 + p_2,$$
$$a_4 \geq -p_1 a_3 + p_2,$$
$$a_5 \geq p_1.$$

This suggests defining the function $M$ (of Step (2)) as: $M(u) = (a_1, a_2, a_3, a_4, a_5)$, with $M(p)$ the Cartesian product of $x_2 \leq -p_1 x_1 + p_2$ in the $x_1, x_2$-plane, $x_4 \geq -p_1 x_3 + p_2$ in the $x_3 x_4$-plane, and $x_5 \geq p_1$ on the $x_5$-axis. Then, $p$ is in $u$ if and only if $M(u)$ is in $M(p)$. The "point in triangles" problem is then solved by storing $M(L)$ in a $c^2b$-tree.

To solve the corresponding counting problem, each triangle is written as the sum and difference of 4 (possibly degenerate) triangles such that each triangle has one vertical and one horizontal edge. Disjointedness is again guaranteed by assigning common edges appropriately. We then perform 4 problems, each on a subset of triangles of common orientation. If a positive (resp. negative) triangle contains $p$, the count is incremented (resp. decremented).

Each of the four subproblems in this case corresponds to a triangle given by a horizontal line, a vertical line, and a diagonal line. Each triangle can be mapped into a point of $E^4$ consisting actually of the Cartesian product of $E^2$ with $E^1$ and $E^1$. The final result is a $cb^2$-tree.

*Case* b. "The segment intersecting segments problem." Let $s$ and $t$ be two segments and let $h(s)$ and $h(t)$ be their supporting lines. Then, $s$ and $t$ intersect if and only if $h(s)$ and $t$ intersect *and* $h(t)$ and $s$ intersect. So, solutions to the "segment intersecting lines" and "line intersecting segments" problems, would solve the "segment intersecting segments" problem. We consider here a solution to the first of these problems.

Let $h: x_2 = h_1 x_1 + h_2$ be a line and $s$ be a query segment of endpoints $l(s) = (s_1, s_2)$ and $r(s) = (s_3, s_4)$ (assume w.l.o.g. that $s_1 \neq s_3$). Then, $s$ intersects $h$ if and only if $l(s)$ and $r(s)$ lie on different sides of $h$. That is,

$$s_2 \leq h_1 s_1 + h_2 \qquad \text{and} \qquad s_4 \geq h_1 s_3 + h_2$$

or

$$s_2 \geq h_1 s_1 + h_2 \quad \text{and} \quad s_4 \leq h_1 s_3 + h_2.$$

We let $M$ be the dual mapping that transforms the line $h$ into the point $M(h) = (h_1, h_2)$, and the segment $s$ into the double wedge $M(s)$ whose points $(x_1, x_2)$ satisfy

$$x_2 \geq -s_1 x_1 + s_2 \quad \text{and} \quad x_2 \leq -s_3 x_1 + s_4$$

or

$$x_2 \leq -s_1 x_1 + s_2 \quad \text{and} \quad x_2 \geq -s_3 x_1 + s_4.$$

By definition of $M$, $s$ intersects $h$ if and only if $M(h)$ is contained in $M(s)$. The double wedge $M(s)$ can easily be partitioned into two wedges (a wedge is a simplex in $E^2$) which suggests storing the set of $M(s)$ in a $c$-tree. There results a $c$-tree for solving the "segment intersecting lines problem."

The "line intersecting segments problem" is solved in a similar manner resulting in a $c^2$-tree. These solutions can then be combined to yield a $c^3$-tree for solving the reporting problem and a $c^2b$-tree for solving the counting problem for "segment intersecting segments."

*Case* c. "The polygon intersecting polygons problem." We assume that all polygons are $m$ sided, closed, bounded, and simple. We assume without loss of generality that no polygonal edges or chords are vertical or horizontal.

Intersection includes both area and boundary intersections of polygons. This leads to the following straightforward observation: If $g$ is a polygon of $S$ and $q$ a query polygon of $Q$ and $p(g)$ and $p(q)$ are arbitrary points of $g$ and $q$, respectively, then $g$ and $q$ intersect if and only if (i) $g$ contains $p(q)$, (ii) $p(g)$ is contained in $q$, or (iii) there are two edges, one of $g$ and one of $q$, that intersect.

We begin with the reporting case, treating the three cases in sequence. We will come back to the issue of multiple reporting later:

(i) To determine the polygons of $S$ that contain $p(q)$, we first decompose each polygon of $S$ into $m$-2 disjoint triangles and form the set $S'$ of all such triangles. Next we answer a "point in triangles" query for $p(q)$, as in Case a.

(ii) All polygons $g$ of $S$ contained in $q$ are found by storing the points $p(g)$ in a $c$-tree, decomposing $q$ into triangles, (i.e., simplices), and answering simplex range queries for each of these simplices.

(iii) Edge intersections are determined by "the segments intersecting segments" method (Case b).

Since cases (i), (ii), and (iii) are not exclusive, the sketched solution leads to multiple reporting of intersecting polygons. However, each polygon is reported at most a constant, that is, at most $m^2 + 2$, number of times. Using a bitvector that has a component $c(g)$ for each polygon $g$ in $S$, multiple reporting can be eliminated as follows:

Prior to a query, $c(g) = 0$, for each polygon $g$ in $S$. When a polygon $g$ is found to intersect $q$ then two cases are distinguished:

Case 1. $c(g) = 0$, then $c(g)$ is set to 1 and $g$ is put onto a stack.

Case 2. $c(g) = 1$, then no action is taken at all.

After searching the space-cutting trees, we remove all polygons $g$ from the stack, report $g$, and set $c(g)$ back to 0.

Since these actions can be implemented in a time proportional to the number of polygons to be reported, this solves the reporting problem.

Since cases (i), (ii), and (iii) are not mutually exclusive, the described solution does not permit efficient counting of intersecting polygons. For this problem it is crucial to perform the various queries dependently. One possibility is to store $S$ in a $cb^2c(c^2b)^m$-tree. (The initial $cb^2$-tree is for case (i), the $c$-tree layer following solves case (ii), and the ($m$ $c^2b$-tree layers are responsible for detecting edge intersection (case (iii)). It would be advantageous to reorder the attributes so that $S$ can be stored in a $c^{2m+2}b^{m+2}$-tree. This would increase the practicality of the theoretical result ($O(n)$ space and $O(n^{a(2)+\varepsilon})$ query time).

*Case* d. "The point in tetrahedra problem." A solution similar to that given for Case a handles this case. Here we decompose a tetrahedron into (at most) 4 disjoint tetrahedra each having two faces normal to the $x_1x_2$-plane. Figure 2 illustrates the two cases which may occur. These cases

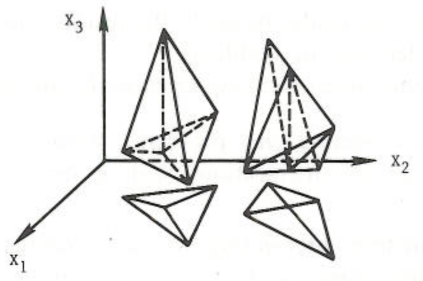

FIGURE 2

depend upon whether the orthogonal projection of $t$ onto the $x_1x_2$-plane is a triangle or a quadrilateral.

From here, it is easy to write a tetrahedron as the union and difference of at most eight tetrahedra having 2 faces normal to the $x_1x_2$-plane and one face normal to the $x_1x_3$-plane.

Repeating the techniques of Case a, we map the tetrahedron into $E^9$, where $dc^3$-trees can be applied to yield the desired result. If tetrahedra are to be reported, then the final decomposition cannot be performed which means that a $d^2c^2$-tree is required.

*Case* e. "The segment intersecting planes problem." The algorithm here is completely analogous to the "segment intersecting lines" problem. A plane intersects a segment if and only if the two endpoints of the segment are on opposite sides of the plane.

We use the dual transform to map the plane into a point and the segments into wedges. The result is a problem solvable by a $d$-tree.

*Case* f. "The plane intersecting segments problem." Let $S$ be a set of $n$ segments of $E^3$. Each segment of $S$ can be mapped into $E^6$ and then stored in a $d^2$-tree to report those segments which intersect a query plane.

If intersections are to be counted, each segment can be written as the difference of two rays. The rays can then be mapped into $E^5$ and stored in $dc$-trees.

## 5. Extensions and Conclusions

The contributions of this paper are two-fold: a family of space-efficient data structures (so-called recursive space-cutting trees) is developed, and various intersection search problems in $E^2$ and $E^3$ are solved making use of members of this family. The recursive space-cutting trees are presented along with the applications to clarify the application of recursive space-cutting trees. We invite the reader to apply the results and methods presented here to an even wider class of problems.

Particular problems which we leave open in $E^3$ include the following:

1. The "line intersecting lines problem," where a set of lines is to be preprocessed so that all intersections with a new line can be quickly reported/counted.

2. The "polyhedron intersecting polyhedra problem," where a set of polyhedra are preprocessed so that all intersections with a new query polyhedron can be quickly reported/counted.

These problems along with those solved in this paper are likely to lead to more efficient solutions to some practical problems which arise in computer graphics. For example, the problems of windowing, haloing, hidden line, and hidden surface elimination have at their core geometric problems such as those considered here.

It is possible to dynamize the data structures which we present here. That is, to modify them so that queries, insertions, and deletions of objects can be performed efficiently. Let $T$ be some recursive space-cutting tree requiring $O(n)$ space, $O(f(n))$ query time, with $f(n) = \Omega(n^a)$ for some $a > 0$, and $O(g(n))$ time for construction, with $g(n) = \Omega(n^{1+b})$ for some $b > 0$. We can derive a data structure $T'$ with

$O(n)$ space,

$O(f(n))$ query time, and

$O(g(n)/n)$ time for performing an insertion or deletion.

$T'$ is basically a system of "disjoint" instances of $T$ a la Bentley [$B$].

If the recursive space-cutting tree $T$ to be dynamized is a $w$-tree, for $w$ in $\{b, c\}^+$ then $f(n) = n^{a(2)}$ or $n^{a(2)+\varepsilon}$, for any $\varepsilon \geq 0$, and $g(n) = n^{1+\varepsilon}$[$M$]. Thus, $T'$ realizes $O(n^{a(2)})$ (resp. $O(n^{a(2)+\varepsilon})$) query time and $O(n^{\varepsilon})$ insertion and deletion time. The sketched method does not yield reasonable time-bounds if $w$ is in $\{b, c\}^*\{d\}\{b, c, d\}^*$ as no algorithm is known that constructs a $d$-tree for $n$ points efficiently.

Finally, there is the challenging problem of finding space-cutting trees that are more efficient than the current best $c$- and the $d$-trees. A range searching tree in $E^4$ was recently discovered by Cole [Co]; obvious extensions of this work follow.

## APPENDIX—Analysis of Query Time in $c$- and $d$- Trees

In this Appendix we justify the assumption that a $d$-tree allows for answering a three-dimensional simplex query in $O(n^{a(3)} + I)$ time. While this result is not difficult, it has been used here and we are unaware of any previous proof. Our proof uses the following

*Fact.* If $f(v)$ be any function growing no faster than $c\, n^{a(3)}/\log n$ for some positive $c$ (i.e., $f(v) = O(n^{a(3)}/\log n)$. Then, in searching $d$-trees when our search object is a simplex,

$$\sum_{v \text{ a node visited}} f(v) = O(n^{a(3)}).$$

To prove this, we note that each node, $v$, in our tree is represented by a hyperplane $h(v)$, defining 2 halfspaces $h^+(v)$ and $h^-(v)$, a point $p(v)$ which lies in dom($v$), and a number $\#(v)$ representing the number of points in dom($v$).

A query is represented by a polyhedron which is initially a simplex, $q$, and shrinks in volume as it is intersected with halfspaces encountered in our search. For our algorithms, this polyhedron can be represented by 4 polygons which represent its intersections with the 4 (cross-sectional) faces of $q$. These polygons are labeled $P_i$ $(i = 1, 2, 3, 4)$. For a typical polyhedron, $P$, the search at $v$ proceeds as follows:

```
if all $P_i$ are empty then
        if $p(v)$ lies in the interior of $q$
        then add $\#(v)$ to the count
        else ($p(v)$ is not in $q$) do nothing.
else for each non-empty $P_i$, we form the 3 sons of $P_i$ by intersecting $P_i$
with $h^-(v)$, $h^+(v)$ and $h(v)$.
```

In the first two cases, the results are 2 polygons having between them 4 vertices more than $P_i$ had. In the third case, the result is an interval (to be searched by a $c$-tree). These polygons (and interval) can be constructed in $|P_i|$ time where $|P_i|$ is the number of vertices of $P_i$.

Having created these 2 polygons, our algorithm now proceeds as

```
        S1. Visit leftson($v$) with $P_i \cap h^-(v)$
        S2. Visit rightson($v$) with $P_i \cap h^+(v)$
        S3. Visit middleson($v$) with $P_i \cap h(v)$
```

Note that a node creates only 6 new vertices (4 on polygons, 2 on an interval) in time $O(|P|)$ and each new vertex is only used in 1 place. So, $f(v) \leq \log n$ (since each vertex follows only 1 path) and the lemma holds.

*Note added in proof*. Since this paper was submitted, range searching trees in $E^d$ have been found for all $d$ (Yao and Yao, "Proceedings 17th Ann. ACM Symp. Theor. Comput." and Haussler-Welzl, *Discrete and Computational Geometry*, to appear).

## References

[BM] R. Bayer and E. M. McCreight, Organization and maintenance of large ordered indices, *Acta Inform.* **1** (1972), 173–189.

[B] J. L. Bentley, Decomposable searching problems. *Inform. Process. Lett.* **8** (1979), 244–251.

[Br] K. Q. Brown, "Geometric Transforms for Fast Geometric Algorithms," Rep. CMU-CS-80-101, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[Ch] B. M. CHAZELLE, "Reporting and Counting Arbitrary Planar Intersections," Rep. CS-83-16, Dept. of Computer Science, Brown University, Providence, RI, 1983.

[Co] R. COLE, private communication.

[EKM] H. EDELSBRUNNER, D. G. KIRKPATRICK, AND H. A. MAURER, Polygonal intersection searching, *Inform. Process. Lett.* **14** (1982), 74–79.

[EW] H. EDELSBRUNNER, AND E. WELZL, "Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query Time," Rep. F111, Inst. Inform. Proc., Techn. Univ. Graz, Austria, 1983.

[M] N. MEGGIDO, Partitioning with lines in the plane, *J. Algorithms*, in press.

[W] D. E. WILLARD, Polygon retrieval, *SIAM J. Comput.* **11** (1982), 149–165.

[Y] F. F. YAO, A 3-space partition and its applications, in "Proceedings, 15th Ann. ACM Symp. Theor. Comput.," (1983), pp. 258–263.

[YDE] F. F. YAO, D. P. DOBKIN, AND H. EDELSBRUNNER, to appear.