New upper bounds in Klee's measure problem

Mark H. Overmars and Chee-Keng Yap

RUU-CS-89-28
November 1989

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

New upper bounds in Klee's measure problem

Mark H. Overmars and Chee-Keng Yap

Technical Report RUU-CS-89-28
November 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

New upper bounds in Klee's measure problem*

Mark H. Overmars† Chee-Keng Yap‡

Abstract

We give new upper bounds for the measure problem of Klee which significantly improve the previous bounds for dimensions greater than 2. We obtain an $O(n^{d/2} \log n, n)$ time-space upper bound to compute the measure of a set of n boxes in Euclidean d-space. The solution is based on a new data structure that we call an orthogonal partition tree that has other applications as well. The method involves several new ideas including application of the inclusion/exclusion principle, the concept of trellises and streaming.

1 Introduction

Around 1977, Klee [5] posed the *measure problem*: given a set of n intervals (of the real line), find the length of their union. He gave an $O(n \log n)$ time solution and asked if this was optimal. This generated considerable interest in the problem, and shortly after, Fredman and Weide [4] proved that $\Omega(n \log n)$ is a lower bound under the usual model of computation. Bentley [2] considered the natural extension to d-dimensional space where we ask for the d-dimensional measure of a set of d-rectangles. He showed that the $O(n \log n)$ bound holds for $d = 2$ as well, and, for $d > 2$, the result generalizes to an upper bound of $O(n^{d-1} \log n)$. Thus the results are optimal for $d = 1, 2$. We refer to the book [7] for an account. Concerning these results for $d \geq 3$, Preparata and Shamos remarked in their book ([7] pp.328-9):

> What is grossly unsatisfactory about the outlined method for $d \geq 3$ is the fact that there is a "coherence" between two consecutive sections in

*Research of the first author was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM). Research of the second author has been supported by ONR grant N00014-85-K-0046, and NSF grants DCR-84-01898 and CCR-87-03458.

†Dept. of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

‡Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA.

> the sweep that we are unable to exploit. ... Although it seems rather difficult to improve on this result, no conjecture about its optimality has been formulated.

The only progress made since was a small improvement by van Leeuwen and Wood [8] who removed the $\log n$ factor from Bentley's upper bound for $d \geq 3$. The test case seems to be $d = 3$: is $O(n^2)$ really necessary for computing the volume of a set of n boxes in 3-space? In this paper we show that $O(n^{1.5} \log n)$ suffices. This immediately implies that in d-dimensions ($d \geq 3$), the bound becomes $O(n^{d-1.5} \log n)$.

The idea is to use a plane-sweep approach and dynamically maintain the measure of a set of 2-dimensional rectangles in time $O(\sqrt{n} \log n)$ per update.

Such a result means that we can maintain the area of a set of rectangles *implicitly* without having to represent the full boundary structure. This is because any explicit representation of the boundary of n rectangles requires $\Omega(n^2)$ time in the worst case because of the simple 'trellis' example (see figure 1): it consists of n long vertical rectangles which are pairwise disjoint, superposed on n long horizontal rectangles also disjoint among themselves.

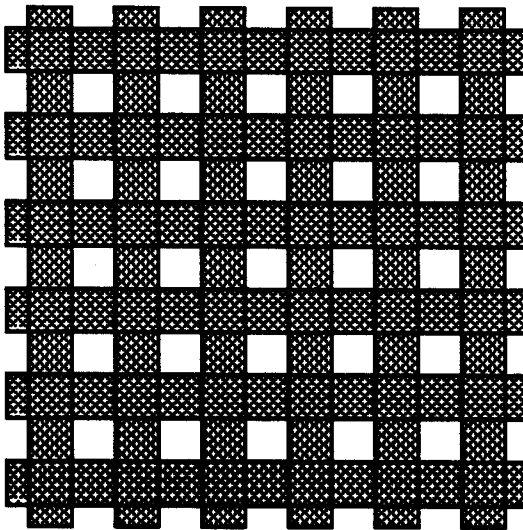

Figure 1: Trellis

The first idea is to exploit the regularity of such trellis structures by maintaining only $O(n)$ amount of information (at the boundary of the box containing the trellis) to keep track of the area of the trellis rectangles. Of course, a union of rectangles is too irregular to be consistently exploited in this way, so the next idea is to partition the plane into a collection of trellises. Using a generalization of the k-d tree of Bentley [1], we are able to form such a partition with only $O(n)$ trellises each of size

$O(\sqrt{n})$. As we shall see, extending this to higher dimensions requires a partition with interesting properties that might be useful for other applications as well.

The rest of this paper is organized as follows: Section 2 describes the basic space sweep algorithm we use and introduces the generalized k-d tree, that we will call an orthogonal partition tree, for storing the boxes. Section 3 contains the solution to the 3-dimensional measure problem. Section 4 generalizes this solution to a d-dimensional method, using an interesting partition scheme of the d-dimensional space. This results in a $O(n^{d/2} \log n, n^{d/2})$ time-space upper bound. In section 5 we exploit a streaming technique of Edelsbrunner and Overmars [3] to reduce the amount of storage required to $O(n)$ only, for any dimension d. In section 6 we briefly mention some other applications of the method like e.g. computing the measure of the boundary of the union of a set of boxes. Finally, in section 7, some conclusions, extensions and open problems are given.

Throughout the paper we will use the following terminology. A *d-box* is the cartesian product of d intervals in d-dimensional space. The *i-boundaries* of a d-box are the parts of the boundary that are perpendicular to the i-th coordinate axis. Each d-box has two i-boundaries for $1 \leq i \leq d$. We refer to them as the left and right i-boundary. The *i-interval* is the projection of the d-box on the x_i-axis. For a d-box R we denote with $Int(R)$ the interior of R.

Definition 1.1 *A d-box R_1 is said to* partially cover *R_2 if the boundary of R_1 intersects $Int(R_2)$. R_1 is said to* (completely) cover *R_2* if *$R_2 \subseteq R_1$.*

Definition 1.2 *For two d-boxes R_1 and R_2 we say that R_1 is an i-*pile *w.r.t. R_2 if R_1 partially covers R_2 and for all $1 \leq j \leq d$ with $j \neq i$ the j-interval of R_2 is fully contained in the j-interval of R_1.*

In other words, in each direction, except for direction i, R_1 completely covers R_2. i-piles will play an important role in this paper.

An extended abstract of this paper appeared in [6].

2 General framework

The basic method for solving the d-dimensional measure problem is as follows. Let V be the set of n d-boxes for which we want to compute the measure. Let $V' = \{a_1, ..., a_{n'}\}$ be the set of all different x_d-coordinates of vertices of the boxes, i.e., all different endpoints of d-intervals. We sort the boxes both by left and right d-boundary. We solve the measure problem using a space sweep approach turning the static d-dimensional problem into a dynamic $(d-1)$-dimensional problem. We sweep a hyperplane along the d-th coordinate axis stopping at each value in V'. During the sweep we maintain the $(d-1)$-dimensional measure of the boxes intersected by the sweep plane. At each step of the sweep we multiply this $(d-1)$-dimensional measure by the distance traveled with the sweep hyperplane and add this to the measure found so far. The algorithm looks as follows.

S:=$\emptyset$;
$MEAS$:=0;
for i:=1 **to** $n' - 1$ **do**
    Insert all d-boundaries of boxes that start at a_i in S;
    M:=$(d-1)$-dimensional measure of boxes in S;
    $MEAS$:=$MEAS + (a_{i+1} - a_i) \times M$;
    Delete all d-boundaries of boxes that end at a_{i+1} from S
end;

At termination $MEAS$ will contain the measure of the set of boxes. S will be a dynamic data structure for maintaining the $(d-1)$-dimensional measure. If insertions and deletions in S can be performed in time $F_{d-1}(n)$ the method will take time $O(n \log n + nF_{d-1}(n))$. This approach is due to Bentley.

To maintain the measure of the set of boxes intersected by the sweep-plane we introduce a generalization of the k-d tree.

Definition 2.1 *A d-dimensional orthogonal partition tree is a balanced binary tree. With each internal node δ is associated a region C_δ of the d-dimensional space, with the following properties:*

- *C_{root} is the whole d-dimensional space.*
- *For each node δ C_δ is a (possibly unbounded) d-box.*
- *For each node δ with sons δ_1 and δ_2, $Int(C_{\delta_1}) \cap Int(C_{\delta_2}) = \emptyset$ and $C_{\delta_1} \cup C_{\delta_2} = C_\delta$.*

C_δ will be called the *region associated with δ*. When δ is a leaf we refer to C_δ as a *cell.* It immediately follows that for each full level of the orthogonal partition tree all regions are essentially disjoint and their union is the d-dimensional space. From now on we drop the qualifying word "orthogonal" and speak only of "partition trees", which is not to be confused with the "non-orthogonal" partition trees of e.g. Willard [11] and Welzl [10].

To use partition trees for maintaining the measure of a set of d-boxes we store the following extra information in the partition tree: With each leaf δ we store all boxes that intersect $Int(C_\delta)$ but do not cover the region associated with the father of δ. For each internal node δ we store a counter TOT_δ that contains the number of d-boxes that completely cover C_δ but only partially cover $C_{father(\delta)}$. Finally, with each node δ we associate a field M that is defined as follows: If δ is a leaf M contains the measure of the boxes stored at δ restricted to C_δ. Otherwise, if $TOT_\delta > 0$ then M is the measure of C_δ, otherwise $M = M_{lson(\delta)} + M_{rson(\delta)}$. It is easy to verify that M_{root} is the measure of the set of d-boxes.

To maintain the measure in a dynamically changing set we have to be able to insert and delete d-boxes in the partition tree. The basic insertion algorithm is the following:

```
procedure Insert(box,δ);
    if δ is a leaf then
        Store box at δ;
        Recompute M_δ
    elsif box covers C_δ then
        TOT_δ:=TOT_δ + 1;
        M_δ:=measure of C_δ
    elsif box partially covers C_δ then
        Insert(box,lson(δ));
        Insert(box,rson(δ));
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    end;
```

The routine is invoked via Insert(*box*,*root*). The deletion routine is similar:

```
procedure Delete(box,δ);
    if δ is a leaf then
        Remove box at δ;
        Recompute M_δ
    elsif box covers C_δ then
        TOT_δ:=TOT_δ − 1;
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    elsif box partially covers C_δ then
        Delete(box,lson(δ));
        Delete(box,rson(δ));
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    end;
```

The routine is invoked via Delete(*box*,*root*). Note the similarity with the methods of Bentley [2] and van Leeuwen and Wood [8] for the 1- and 2-dimensional case. The main difference is that we no longer insist that the leaves are fully covered by the boxes that intersect them. It is immediately clear that the amount of time required depends on the number of nodes visited and the amount of time required for computing the measure at the leaves. In the sequel of this paper we will show that partition trees exist in which both are small.

3 Dynamic measure problem in two dimensions

To illustrate the general solution we will develop in the next section, we first solve the 3-dimensional measure problem. Solving the 3-dimensional problem means that we have to design a 2-dimensional partition tree with good performance. To obtain such a partition tree we first define a subdivision of the plane into rectangular cells with some interesting properties.

Let V be the set of the rectangles that will be inserted and deleted in the partition tree. First we split the x_1-axis into $2\sqrt{n}$ intervals such that each interval contains $\leq \sqrt{n}$ 1-boundaries of rectangles. This defines $2\sqrt{n}$ slabs in the plane. Each slab s will be split by horizontal line segments into a number of cells. Let V_s^1 be the set of rectangles that have a 1-boundary inside s. Let V_s^2 be the set of rectangles that only have a 2-boundary intersecting s. (Note that the size of V_s^1 is bounded by $\sqrt{n}$ but the size of V_s^2 can be almost n.) We draw a line segment along each 2-boundary of a rectangle in V_s^1. Moreover, we draw a line segment along each $\sqrt{n}$-th 2-boundary of a rectangle in V_s^2. In this way s is partitioned into $\leq 4\sqrt{n}$ rectangular cells.

Lemma 3.1 *The partition has the following properties:*

1. *There are $O(n)$ cells.*

2. *Each rectangle of V partially covers at most $O(\sqrt{n})$ cells.*

3. *No cell contains vertices in its interior.*

4. *Each cell has at most $O(\sqrt{n})$ rectangles partially covering it.*

Proof. Property 1 follows from the fact that there are $2\sqrt{n}$ slabs, each with $4\sqrt{n}$ cells.

To show property 2 note that each vertical line cuts through one slab, i.e., through at most $4\sqrt{n}$ cells. Each horizontal line cuts in each slab through one cell. Hence, through at most $2\sqrt{n}$ cells in total. V partially covers a cell only if its boundary cuts through the cell. This boundary consists of two vertical and two horizontal line segments. Hence, it cuts through at most $12\sqrt{n}$ cells.

Property 3 follows from the fact that, if a vertex lies in the interior of a slab s, it belongs to a rectangle in V_s^1 and, hence, it will lie on the horizontal segment drawn through its 2-boundary, i.e., on the boundary of a cell.

Finally, property 4 follows from the fact that each slab contains at most $\sqrt{n}$ 1-boundaries and each cell at most $\sqrt{n}$ 2-boundaries. □

We will use the cells of this partition as leaves of the partition tree. For each slab s we construct a binary tree T_s that contains its cells ordered by x_2-coordinate in its leaves. Next we construct a tree T that contains the slab trees T_s ordered by x_1-coordinate in its leaves. See figure 2 for an example. Each node in the tree, consisting of T and the slab trees T_s, has an associated region, being the union of the cells at the leaves in the subtree. It is easy to see that such a region is a (possibly infinite) rectangle.

Lemma 3.2 *Let V be a set of n rectangles in the plane. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n)$ nodes.*

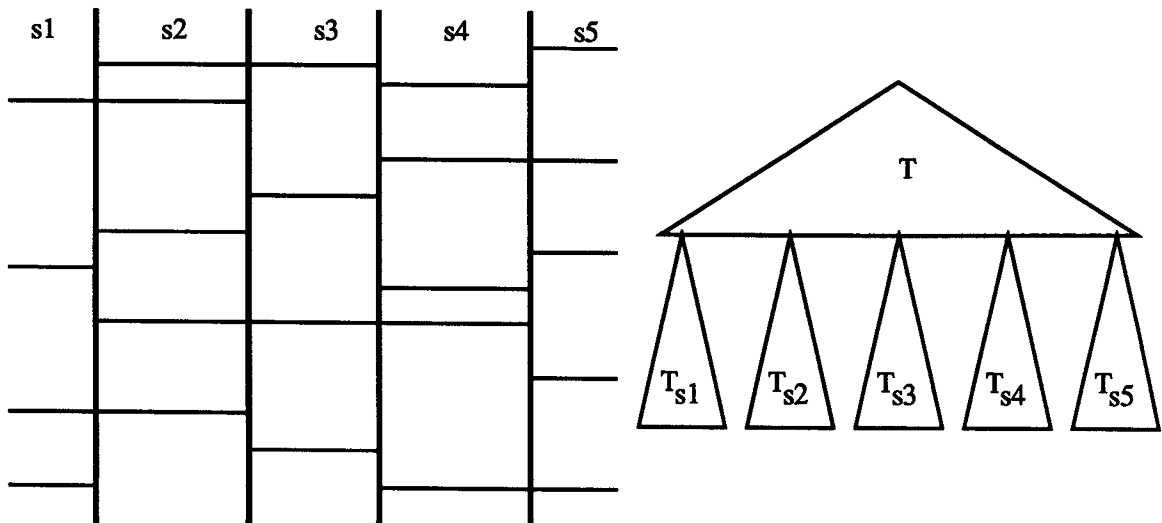

Figure 2: The 2-dimensional partition tree.

2. *Each rectangle is stored in* $O(\sqrt{n})$ *leaves.*

3. *Each rectangle influences* $O(\sqrt{n}\log n)$ *TOT fields.*

4. *No cell of a leaf contains vertices of rectangles in the interior.*

5. *Each leaf stores no more than* $O(\sqrt{n})$ *rectangles.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two. If the tree has $O(n)$ nodes its depth is bounded by $O(\log n)$. When a rectangle influences the TOT field of a node δ it partially covers $C_{father(\delta)}$, and there must be a leaf below $father(\delta)$ that is intersected by the rectangle. Hence, the number of internal nodes intersected by a rectangle is bounded by $O(\log n)$ times the number of leaves where the rectangle is stored. As a result the rectangle can only influence that number of TOT fields. □

It remains to show how the measure at a leaf is maintained when inserting and deleting rectangles. To this end we use the inclusion/exclusion principle. Note that, due to property 4, the rectangles stored at a leaf δ are 1-piles or 2-piles w.r.t. C_δ. In other words, they form a trellis. The measure of such a trellis can be maintained in the following way. Let V_1 be the projection of the 1-piles on the x_1-axis and let V_2 be the projection of the 2-piles on the x_2-axis. Let M_1 be the (1-dimensional) measure of V_1 and M_2 the (1-dimensional) measure of V_2. Assume that the cell C_δ has measure $L_1 \times L_2$. Now it is easy to see that the measure of the trellis is $M_1 \times L_2 + M_2 \times L_1 - M_1 \times M_2$. Hence, we just have to maintain the 1-dimensional

measure of V_1 and V_2. For this we can use a simple segment tree that uses linear storage and maintains the measure in time $O(\log n)$ per insertion and deletion (see [7]). So with each cell (leaf) δ we associate two segment trees S_1 and S_2. S_1 contains the projections of the 1-piles in δ on the x_1-axis and S_2 contains the projections of the 2-piles on the x_2-axis. Inserting (deleting) a rectangle at δ now consists of inserting (deleting) it in S_1 or S_2 (never in both). In this way M_1 and M_2 get updated and we obtain the new measure in the leaf.

Theorem 3.3 *The measure of a set of n 3-boxes in 3-dimensional space can be computed in time $O(n\sqrt{n}\log n)$ using $O(n\sqrt{n})$ storage.*

Proof. We use the plane sweep approach and maintain the partition tree described above. To insert or delete a rectangle we have to update $O(\sqrt{n}\log n)$ TOT fields. This takes time $O(\sqrt{n}\log n)$. Next, we have to insert or delete the rectangle at $O(\sqrt{n})$ leaves. At each such leaf this causes an insertion or deletion in a segment tree which takes $O(\log n)$. Hence, the total update time of the partition tree is $O(\sqrt{n}\log n)$.

The bound on the amount of storage required follows from the fact that the tree itself takes $O(n)$ storage and each leaf stores $O(\sqrt{n})$ information (according to property 5 of the above lemma). □

In section 5 we will show how to reduce the amount of storage used to $O(n)$.

4 Dynamic measure in multi-dimensional space

We will now generalize this method to d-dimensional space. To this end we will describe a d-dimensional partition tree, based on a cell decomposition of the d-dimensional space.

Definition 4.1 *A* slab *at level $i = 1, ..., d$ is a subset of $\mathcal{R}^d$ of the form $I_1 \times I_2 \times ... \times I_i \times \mathcal{R}^{d-i}$ where $I_1, ..., I_i$ are intervals of $\mathcal{R}$.*

Let V be the set of all d-boxes that will be inserted or deleted in the partition tree. We split the x_1-axis in $2\sqrt{n}$ intervals, each of whose interior contains $\leq \sqrt{n}$ 1-boundaries of boxes. This splits the d-dimensional space in $2\sqrt{n}$ slabs at level 1. For each such slab s let V_s be the set of d-boxes that partially cover s. We split V_s in two subsets: V_s^1 of d-boxes that have a 1-boundary inside s and V_s^2 of d-boxes that do not have a 1-boundary inside s. Note that $|V_s^1| \leq \sqrt{n}$. Each slab s we now split with respect to second coordinate. We split it at the 2-boundaries of each d-box in V_s^1 and we split it at every $\sqrt{n}$-th 2-boundary of d-boxes in V_s^2. As a result we split each slab s into $O(\sqrt{n})$ slabs at level 2. For each such slab s' let $V_{s'}$ be the set of d-boxes that partially cover s'. We again split $V_{s'}$ into two subsets: $V_{s'}^1$ of boxes that have a 1- or a 2-boundary intersecting $Int(s')$ and $V_{s'}^2$ of boxes that do not have a

1- nor a 2-boundary intersecting $Int(s')$. (Note that there are no boxes that have both a 1- and 2-boundary intersecting $Int(s')$.) Again $|V_{s'}^1| = O(\sqrt{n})$. We split s' into slabs at level 3 with respect to the third coordinate. Again, we split at each 3-boundary of boxes in $V_{s'}^1$ and at every $\sqrt{n}$-th 3-boundary of boxes in $V_{s'}^2$. In this way we continue for all coordinates.

Lemma 4.1 *The partition has the following properties:*

1. *There are $O(n^{d/2})$ cells.*
2. *Each d-box of V partially covers at most $O(n^{(d-1)/2})$ cells.*
3. *Each cell only contains piles in its interior.*
4. *Each cell has at most $O(\sqrt{n})$ d-boxes partially covering it.*

Proof. For each coordinate, every slab at level i is split into $O(\sqrt{n})$ slabs at level $i+1$. Hence the total number of cells we obtain is $O(\sqrt{n}^d) = O(n^{d/2})$.

If a d-box R partially covers a cell C then an i-boundary B of R cuts through C for some i ($1 \leq i \leq d$). At the moment we split slabs at level $i-1$ with respect to the ith coordinate there are $O(n^{(i-1)/2})$ slabs. Each of these slabs is split into $O(\sqrt{n})$ slabs at level i at this moment but B can cut through only one of them (because the cutting is done with respect to the ith coordinate axis). So after the ith step B still cuts through at most $O(n^{(i-1)/2})$ slabs at level i. In the next $(d-i)$ steps each slab at level i is cut into $O(n^{(d-i)/2})$ cells. So B will cut through at most $O(n^{(i-1)/2} \times n^{(d-i)/2}) = O(n^{(d-1)/2})$ cells.

Property 3 follows from the fact that no d-box can have both an i_1- and an i_2-boundary intersecting a slab at level i with $i_1 < i_2 \leq i$. Hence, no d-box has boundaries in more than one coordinate intersecting a slab at level d, i.e., a cell. So each d-box forms a pile in a cell.

The last property follows immediately from the way we split the slabs. □

We will use the cells of this partition as leaves of the partition tree. It is easy to see how the rest of the tree can be built on top of it. The tree consists of d "stages", where each stage consists of $O(\log n)$ levels of the tree. The top stage consists of a tree T that stores the $2\sqrt{n}$ slabs at level 1 in its leaves, sorted on x_1-coordinate. Each slab is represented by a slab tree that stores its slabs at level 2 (created in the second step) sorted by x_2-coordinate. For each of these slabs there is again a slab tree that stores its subdivision by x_3-coordinate, etc.

Lemma 4.2 *Let V be a set of n d-boxes in d-dimensional space. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n^{d/2})$ nodes.*

2. *Each d-box is stored in* $O(n^{(d-1)/2})$ *leaves.*

3. *Each d-box influences* $O(n^{(d-1)/2}\log n)$ *TOT fields.*

4. *Each cell of a leaf only contains piles.*

5. *Each leaf stores no more than* $O(\sqrt{n})$ *d-boxes.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two as the depth is again bounded by $O(\log n)$. □

It remains to show how the measure at a leaf is maintained when inserting and deleting d-boxes. To this end we again use the inclusion/exclusion principle. As stated in property 4, the d-boxes stored at a leaf δ are piles and form a d-dimensional trellis. Let V_i be the projection of the i-piles on the x_i-axis for each $1 \leq i \leq d$. Let M_i be the 1-dimensional measure of V_i. Let L_i be the length of C_δ in direction x_i. The following result is easy to proof:

Lemma 4.3 *The measure of the trellis is*

$$\sum_{1\leq k\leq d} (-1)^{k+1} Z_k$$

where

$$Z_k = \sum_{1\leq j_1<\ldots<j_k\leq d} \left(\prod_{1\leq i\leq k} M_{j_i} \prod_{l\neq j_i \text{ for any } i} L_l \right)$$

Although this might look quite complicated it is simply the inclusion/exclusion principle. E.g., for $d = 3$ the formula shows that the measure is

$$M_1L_2L_3 + L_1M_2L_3 + L_1L_2M_3 - M_1M_2L_3 - M_1L_2M_3 - L_1M_2M_3 + M_1M_2M_3.$$

When M_i is known for each i the measure can be computed in constant time (assuming d is a constant).

Hence, we just have to maintain the 1-dimensional measure of V_i for each i. For this we use d segment trees $S_1 \ldots S_d$, one for each dimension. An insertion or deletion in a leaf means inserting or deleting the i-pile in the correct segment tree S_i. In this way we obtain the updated measure M_i and we can recompute the above formula to obtain the new measure in the cell. This will take time $O(\log n)$ (assuming that d is a constant).

Lemma 4.4 *Updates in the d-dimensional partition tree take time* $O(n^{(d-1)/2}\log n)$ *and the tree uses* $O(n^{(d+1)/2})$ *storage.*

Proof. Follows from the above lemmas. □

Theorem 4.5 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n^{d/2})$ storage.*

Proof. We use the plane sweep approach and maintain a $(d-1)$-dimensional partition tree. So we have to perform $O(n)$ updates, each taking time $O(n^{(d-1-1)/2} \log n)$. The time bound follows. According to the preceding lemma, the structure uses $O(n^{(d-1+1)/2})$ storage. □

5 Reducing the amount of storage

In this section we will show how the amount of storage required can be reduces to $O(n)$. To this end we use an instance of the streaming technique introduced in Edelsbrunner and Overmars[3].

The idea of streaming is the following: Before hand we know what updates have to be performed and in what order. We can view the space sweep method as traversing in time (being the d-th coordinate). Each update in the structure has to be performed at a specific moment in time. Before each update we check what the current measure is and we multiply it by the time passed since the last update. Rather than building the structure and performing the updates one after the other, we will perform them simultaneously and construct parts of the data structure when we need them. When we are ready with the part we discard it again to free memory.

To formalize this, at any moment we are given a sequence of updates L over time and a region of the space C. This region corresponds to some node in the tree and L is the sequence of updates that will pass through this node. With each update in L we have stored the time at which it has to be performed. In the beginning C is the whole $(d-1)$-dimensional space and L is the complete list of updates, time being the d-th coordinate. A counter $MEAS$ will be used to collect all the measure found. In the beginning it will be set to 0.

The technique now works as follows: When all $(d-1)$-boxes in L are piles with respect to C (i.e., we are at a leaf in the partition tree) we construct $d-1$ segment trees. We perform all the updates on the segment trees and compute the $(d-1)$-measure in the cell after each update. These measures we multiply with the time period to the next update to obtain the d-measure in C. This d-measure we add to $MEAS$. This will take time $O(|L| \log n)$ and $O(|L|)$ storage. Afterwards we destroy all the structures.

When not all boxes are piles (i.e., we are at an internal node) we first compute during which periods of time C will be completely covered by one box. (This corresponds to the time when $TOT \neq 0$.) This can be done by simply walking along the list of updates and maintaining the number of boxes that cover C. Whenever

this number is larger than 0 C is covered. This takes time $O(|L|)$. We multiply the $(d-1)$-measure of C with the total amount of time C is covered and add it to $MEAS$. Next we change time by collapsing the covered periods into a single moment, performing all the updates in that period at the same moment. (This is necessary to avoid that lower in the tree measure will be found during these periods again and counted twice.) Boxes that are now inserted and deleted at the same moment are removed from L. Again this takes time $O(|L|)$ only.

Next we split C into two cells C_1 and C_2 in a way similar to as it would have been split in the partition tree. This can be done in the following way. Remember that in the first stage of the tree we split on x_1-coordinate, in the next stage on x_2-coordinate etc. until, in the last stage we split on x_{d-1}-coordinate. Hence, it is easy to remember on which coordinate we have to split at a particular moment. So assume we have to split on the i-th coordinate. There are two different splits we make: splits along i-boundaries in V_C^1 or splits along i-boundaries of boxes in V_C^2 (see the previous section). There is no problem in first making the splits along i-boundaries of V_C^1 and after that along i-boundaries in V_C^2. (The tree will get a depth that is at most twice as large.) So making a split can be done as follows:

- Let i be the current splitting coordinate. Split L into V_C^1 and V_C^2.
- If $V_C^1 \neq \emptyset$ then split along the median i-boundary in V_C^1.
- Else, if V_C^2 contains more than $\sqrt{n}$ i-boundaries split along the median i-boundary in V_C^2.
- Else, increase i and repeat the procedure.

Finding the splitting line can easily be done in time $O(|L|)$. It is easy to see that the resulting partition tree will still satisfy the properties in lemma 4.2.

Now we construct the list L_1 out of L containing the updates that influence C_1. In L we only keep the other updates. Hence, each update is either stored in L_1 or in L. We recursively call the routine for C_1 and L_1. When we get back from the recursive calls, we join L_1 and L to reconstruct L in its original form. Now we determine the list L_2 of updates that influence C_2, again leaving in L the other updates. We now recursively call the routine with C_2 and L_2. When we get back we again reconstruct L (to be used one level higher in the recursion). Note that during the whole process we never copy updates. We simply take a part of list L and send it down the recursion. When we get back we take another part of L and go again in recursion. As a result each update is stored at at most one place.

The method does essentially the same work as the original technique in which all updates are performed one after the other. In fact, it is more efficient because of two reasons. When the whole list consists of piles we immediately solve the problem rather than splitting till the list contains less than $\sqrt{n}$ boxes. Secondly, we don't consider boxes anymore when during their whole period of existence they are covered by some other box.

Theorem 5.1 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n)$ storage.*

Proof. The amount of time used is essentially the same as in the case we performed the updates one after the other.

To estimate the amount of storage, note that each update is stored at most once in a list L. The bound follows. □

6 Extensions

The partition tree and method described above can also be used to solve a number of related problems. In this section we will briefly mention some of them.

It is well-known that the perimeter of the union of n rectangles in the plane can be computed in time $O(n \log n)$. (See e.g. [7, 9].) Computing the perimeter generalizes to computing the $(d-1)$-dimensional measure of the contour of the union of a set of d-boxes in d-dimensional space. The contour consists of parts of i-boundaries of boxes that do not lie in the interior of the union. We will only show how to compute the measure of the parts of d-boundaries of the contour. The measure of the i-boundaries for other i can easily be obtained by renumbering coordinates. The total measure of the boundary is obviously the sum of the measures of the i-boundaries in the contour for all $1 \leq i \leq d$.

To compute the measure of the d-boundaries of the contour we use exactly the same method as in section 4. We move a sweep plane along the d-th coordinate axis and maintain the measure of the intersection. At any d-boundary where the sweep-plane halts we update the $(d-1)$-dimensional measure as in section 4. The part of this d-boundary that is part of the contour is obviously the absolute value of the difference of the old and the new measure. (Except when more boundaries have the same d'th coordinate value. In this case some care has to be taken. The procedure below correctly treats those cases.) To be precise, the main algorithm (as described in section 2) is changed as follows:

S:=$\emptyset$;
$MEAS$:=0;
for i:=1 **to** n' **do**
  M:=$(d-1)$-dimensional measure of boxes in S;
  Insert all d-boundaries of boxes that start at a_i in S;
  M_+:=$(d-1)$-dimensional measure of boxes in S;
  Delete all d-boundaries of boxes that end at a_i from S;
  M_-:=$(d-1)$-dimensional measure of boxes in S;
  $MEAS$:=$MEAS + (M_+ - M) + (M_+ - M_-)$
end;

S is again stored as a partition tree and maintained in exactly the same way. The correctness of the method is easily established. This lead to the following result:

Theorem 6.1 *The measure of the contour of the union of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n^{d/2})$ storage.*

The method can also be used to compute the measure of lower-dimensional parts of the contour. It is unclear how streaming can be applied here to reduce storage.

As a second application consider the following query problem: Given a set of d-boxes in d-dimensional space, store them such that for a given query box R it can efficiently be determined whether R is completely covered by the d-boxes.

To solve this problem we store the d-boxes in a d-dimensional partition tree. A query is performed using the procedure filled, described below. It gets two arguments, a node δ and the query rectangle R and returns whether the part of R inside C_δ is fully covered by d-boxes. Calling the routine with δ the root of the tree gives the required answer.

```
procedure filled (δ, R):boolean;
    if R completely covers C_δ then
        return M_δ = measure of C_δ
    else if R partially covers C_δ then
        if δ is a leaf then
            search the segment trees to see whether in at least
            one of them the projection of R is fully covered;
            return the result
        else
            return filled(lson_δ,R) and filled(rson_δ,R)
    else
        return true
end;
```

The correctness of the method can easily be established. Searching the segment trees in a leaf takes time $O(\log n)$. This has to be done in at most $O(n^{(d-1)/2})$ leaves. The total number of internal nodes visited is bounded by $O(n^{(d-1)/2} \log n)$. The following theorem follows.

Theorem 6.2 *Let V be a set of n d-boxes in d-dimensional space. One can store V using $O(n^{(d+1)/2})$ storage, such that for a given d-box R one can determine in time $O(n^{(d-1)/2} \log n)$ whether R is completely covered by the boxes in V.*

The method can easily be extended to compute the measure of the union of the d-boxes restricted to a given box R, in the same time bounds. Updates from a fixed set of boxes can be performed in time $O(n^{(d-1)/2} \log n)$ using the same method as described for maintaining the measure.

Other applications exist. For example, it is possible to use the techniques given here to determine contours and i-contours (contour of the area covered by at least or precisely i d-boxes).

7 Conclusions

We have given a new solution to Klee's measure problem that is much more efficient than previously known results, improving the time bound from $O(n^{d-1})$ to $O(n^{d/2} \log n)$. The technique uses some new ideas, including a result on partitioning space, a new type of partition tree and the use of the inclusion/exclusion principle. Streaming was applied to reduce the amount of storage used to $O(n)$.

The dynamic data structure we presented for dynamically maintaining the measure can be used for other problems as well. As we have shown it is very simple to compute e.g. the perimeter. Moreover, the structure gives a compact representation of the shape of the set of d-boxes. This can be used to answer certain classes of queries efficiently.

Some open problems remain. First of all, it might be possible to shave off the factor of $\log n$. But, in fact, there is no reason to believe that the method is even near optimal. Improvements or lower bounds should be worked on. It is also interesting to look at the measure of other objects. For example the best bound known for computing the measure of the union of a set of triangles is $O(n^2)$.

References

[1] Bentley, J.L., Multidimensional binary search trees used for associated searching, *Comm. ACM* **18** (1975), 509–517.

[2] Bentley, J.L., Algorithms for Klee's rectangle problem, Unpublished notes, Dept. of Computer Science, CMU, 1977.

[3] Edelsbrunner, H., and M.H. Overmars, Batched dynamic solutions to decomposable searching problems, *J. Algorithms* **6** (1985), 515–542.

[4] Fredman, M.L., and B. Weide, The complexity of computing the measure of $\cup[a_i, b_i]$, *Comm. ACM* **21** (1978), 540–544.

[5] Klee, V., Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps?, *Amer. Math. Montly* **84** (1977), 284–285.

[6] Overmars, M.H., and C.K. Yap, New upper bounds in Klee's measure problem (extended abstract), *Proc. 29th IEEE Symp. on Foundations of Computer Science,* 1988, pp. 550–556.

[7] Preparata, F.P., and M.I. Shamos, *Computational Geometry,* Springer-Verlag, 1985.

[8] van Leeuwen, J., and D. Wood, The measure problem for rectangular ranges in d-space, *J. Algorithms* **2** (1980), 282–300.

[9] Vitányi, P.M.B. and D. Wood, Computing the perimeter of a set of rectangles, Techn. Rep. 79-CS-23, Unit for Computer Science, McMaster University, 1979.

[10] Welzl, E., Partition trees for triangle counting and other range searching problems, *Proc. 4th ACM Symp. on Computational Geometry,* 1988, pp. 23–33.

[11] Willard, D.E., Polygon retrieval, *SIAM J. Computing* **11** (1982), 149–165.

New upper bounds in Klee's measure problem

Mark H. Overmars and Chee-Keng Yap

RUU-CS-89-28
November 1989

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

New upper bounds in Klee's measure problem

Mark H. Overmars and Chee-Keng Yap

Technical Report RUU-CS-89-28
November 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

New upper bounds in Klee's measure problem*

Mark H. Overmars† Chee-Keng Yap‡

Abstract

We give new upper bounds for the measure problem of Klee which significantly improve the previous bounds for dimensions greater than 2. We obtain an $O(n^{d/2}\log n, n)$ time-space upper bound to compute the measure of a set of n boxes in Euclidean d-space. The solution is based on a new data structure that we call an orthogonal partition tree that has other applications as well. The method involves several new ideas including application of the inclusion/exclusion principle, the concept of trellises and streaming.

1 Introduction

Around 1977, Klee [5] posed the *measure problem*: given a set of n intervals (of the real line), find the length of their union. He gave an $O(n \log n)$ time solution and asked if this was optimal. This generated considerable interest in the problem, and shortly after, Fredman and Weide [4] proved that $\Omega(n \log n)$ is a lower bound under the usual model of computation. Bentley [2] considered the natural extension to d-dimensional space where we ask for the d-dimensional measure of a set of d-rectangles. He showed that the $O(n \log n)$ bound holds for $d = 2$ as well, and, for $d > 2$, the result generalizes to an upper bound of $O(n^{d-1}\log n)$. Thus the results are optimal for $d = 1, 2$. We refer to the book [7] for an account. Concerning these results for $d \geq 3$, Preparata and Shamos remarked in their book ([7] pp.328-9):

> What is grossly unsatisfactory about the outlined method for $d \geq 3$ is the fact that there is a "coherence" between two consecutive sections in

*Research of the first author was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM). Research of the second author has been supported by ONR grant N00014-85-K-0046, and NSF grants DCR-84-01898 and CCR-87-03458.

†Dept. of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

‡Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA.

the sweep that we are unable to exploit. ... Although it seems rather difficult to improve on this result, no conjecture about its optimality has been formulated.

The only progress made since was a small improvement by van Leeuwen and Wood [8] who removed the $\log n$ factor from Bentley's upper bound for $d \geq 3$. The test case seems to be $d = 3$: is $O(n^2)$ really necessary for computing the volume of a set of n boxes in 3-space? In this paper we show that $O(n^{1.5} \log n)$ suffices. This immediately implies that in d-dimensions ($d \geq 3$), the bound becomes $O(n^{d-1.5} \log n)$.

The idea is to use a plane-sweep approach and dynamically maintain the measure of a set of 2-dimensional rectangles in time $O(\sqrt{n} \log n)$ per update.

Such a result means that we can maintain the area of a set of rectangles *implicitly* without having to represent the full boundary structure. This is because any explicit representation of the boundary of n rectangles requires $\Omega(n^2)$ time in the worst case because of the simple 'trellis' example (see figure 1): it consists of n long vertical rectangles which are pairwise disjoint, superposed on n long horizontal rectangles also disjoint among themselves.

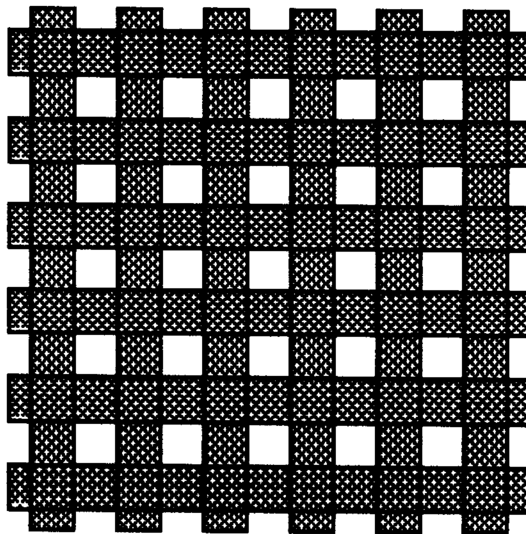

Figure 1: Trellis

The first idea is to exploit the regularity of such trellis structures by maintaining only $O(n)$ amount of information (at the boundary of the box containing the trellis) to keep track of the area of the trellis rectangles. Of course, a union of rectangles is too irregular to be consistently exploited in this way, so the next idea is to partition the plane into a collection of trellises. Using a generalization of the k-d tree of Bentley [1], we are able to form such a partition with only $O(n)$ trellises each of size

$O(\sqrt{n})$. As we shall see, extending this to higher dimensions requires a partition with interesting properties that might be useful for other applications as well.

The rest of this paper is organized as follows: Section 2 describes the basic space sweep algorithm we use and introduces the generalized k-d tree, that we will call an orthogonal partition tree, for storing the boxes. Section 3 contains the solution to the 3-dimensional measure problem. Section 4 generalizes this solution to a d-dimensional method, using an interesting partition scheme of the d-dimensional space. This results in a $O(n^{d/2} \log n, n^{d/2})$ time-space upper bound. In section 5 we exploit a streaming technique of Edelsbrunner and Overmars [3] to reduce the amount of storage required to $O(n)$ only, for any dimension d. In section 6 we briefly mention some other applications of the method like e.g. computing the measure of the boundary of the union of a set of boxes. Finally, in section 7, some conclusions, extensions and open problems are given.

Throughout the paper we will use the following terminology. A *d-box* is the cartesian product of d intervals in d-dimensional space. The *i-boundaries* of a d-box are the parts of the boundary that are perpendicular to the i-th coordinate axis. Each d-box has two i-boundaries for $1 \leq i \leq d$. We refer to them as the left and right i-boundary. The *i-interval* is the projection of the d-box on the x_i-axis. For a d-box R we denote with $Int(R)$ the interior of R.

Definition 1.1 *A d-box R_1 is said to* partially cover *R_2 if the boundary of R_1 intersects $Int(R_2)$. R_1 is said to* (completely) cover *R_2* if *$R_2 \subseteq R_1$.*

Definition 1.2 *For two d-boxes R_1 and R_2 we say that R_1 is an i-*pile *w.r.t. R_2 if R_1 partially covers R_2 and for all $1 \leq j \leq d$ with $j \neq i$ the j-interval of R_2 is fully contained in the j-interval of R_1.*

In other words, in each direction, except for direction i, R_1 completely covers R_2. i-piles will play an important role in this paper.

An extended abstract of this paper appeared in [6].

2 General framework

The basic method for solving the d-dimensional measure problem is as follows. Let V be the set of n d-boxes for which we want to compute the measure. Let $V' = \{a_1, ..., a_{n'}\}$ be the set of all different x_d-coordinates of vertices of the boxes, i.e., all different endpoints of d-intervals. We sort the boxes both by left and right d-boundary. We solve the measure problem using a space sweep approach turning the static d-dimensional problem into a dynamic $(d-1)$-dimensional problem. We sweep a hyperplane along the d-th coordinate axis stopping at each value in V'. During the sweep we maintain the $(d-1)$-dimensional measure of the boxes intersected by the sweep plane. At each step of the sweep we multiply this $(d-1)$-dimensional measure by the distance traveled with the sweep hyperplane and add this to the measure found so far. The algorithm looks as follows.

```
S:=∅;
MEAS:=0;
for i:=1 to n′ − 1 do
    Insert all d-boundaries of boxes that start at a_i in S;
    M:=(d − 1)-dimensional measure of boxes in S;
    MEAS:=MEAS + (a_{i+1} − a_i) × M;
    Delete all d-boundaries of boxes that end at a_{i+1} from S
end;
```

At termination $MEAS$ will contain the measure of the set of boxes. S will be a dynamic data structure for maintaining the $(d-1)$-dimensional measure. If insertions and deletions in S can be performed in time $F_{d-1}(n)$ the method will take time $O(n \log n + nF_{d-1}(n))$. This approach is due to Bentley.

To maintain the measure of the set of boxes intersected by the sweep-plane we introduce a generalization of the k-d tree.

Definition 2.1 *A d-dimensional orthogonal partition tree is a balanced binary tree. With each internal node δ is associated a region C_δ of the d-dimensional space, with the following properties:*

- *C_{root} is the whole d-dimensional space.*
- *For each node δ C_δ is a (possibly unbounded) d-box.*
- *For each node δ with sons δ_1 and δ_2, $Int(C_{\delta_1}) \cap Int(C_{\delta_2}) = \emptyset$ and $C_{\delta_1} \cup C_{\delta_2} = C_\delta$.*

C_δ will be called the *region associated with δ*. When δ is a leaf we refer to C_δ as a *cell.* It immediately follows that for each full level of the orthogonal partition tree all regions are essentially disjoint and their union is the d-dimensional space. From now on we drop the qualifying word "orthogonal" and speak only of "partition trees", which is not to be confused with the "non-orthogonal" partition trees of e.g. Willard [11] and Welzl [10].

To use partition trees for maintaining the measure of a set of d-boxes we store the following extra information in the partition tree: With each leaf δ we store all boxes that intersect $Int(C_\delta)$ but do not cover the region associated with the father of δ. For each internal node δ we store a counter TOT_δ that contains the number of d-boxes that completely cover C_δ but only partially cover $C_{father(\delta)}$. Finally, with each node δ we associate a field M that is defined as follows: If δ is a leaf M contains the measure of the boxes stored at δ restricted to C_δ. Otherwise, if $TOT_\delta > 0$ then M is the measure of C_δ, otherwise $M = M_{lson(\delta)} + M_{rson(\delta)}$. It is easy to verify that M_{root} is the measure of the set of d-boxes.

To maintain the measure in a dynamically changing set we have to be able to insert and delete d-boxes in the partition tree. The basic insertion algorithm is the following:

```
procedure Insert(box,δ);
    if δ is a leaf then
        Store box at δ;
        Recompute M_δ
    elsif box covers C_δ then
        TOT_δ:=TOT_δ + 1;
        M_δ:=measure of C_δ
    elsif box partially covers C_δ then
        Insert(box,lson(δ));
        Insert(box,rson(δ));
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    end;
```

The routine is invoked via Insert(*box*,*root*). The deletion routine is similar:

```
procedure Delete(box,δ);
    if δ is a leaf then
        Remove box at δ;
        Recompute M_δ
    elsif box covers C_δ then
        TOT_δ:=TOT_δ − 1;
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    elsif box partially covers C_δ then
        Delete(box,lson(δ));
        Delete(box,rson(δ));
        if TOT_δ > 0 then M_δ:= measure of C_δ else M_δ:=M_lson(δ) + M_rson(δ) end
    end;
```

The routine is invoked via Delete(*box*,*root*). Note the similarity with the methods of Bentley [2] and van Leeuwen and Wood [8] for the 1- and 2-dimensional case. The main difference is that we no longer insist that the leaves are fully covered by the boxes that intersect them. It is immediately clear that the amount of time required depends on the number of nodes visited and the amount of time required for computing the measure at the leaves. In the sequel of this paper we will show that partition trees exist in which both are small.

3 Dynamic measure problem in two dimensions

To illustrate the general solution we will develop in the next section, we first solve the 3-dimensional measure problem. Solving the 3-dimensional problem means that we have to design a 2-dimensional partition tree with good performance. To obtain such a partition tree we first define a subdivision of the plane into rectangular cells with some interesting properties.

Let V be the set of the rectangles that will be inserted and deleted in the partition tree. First we split the x_1-axis into $2\sqrt{n}$ intervals such that each interval contains $\leq \sqrt{n}$ 1-boundaries of rectangles. This defines $2\sqrt{n}$ slabs in the plane. Each slab s will be split by horizontal line segments into a number of cells. Let V_s^1 be the set of rectangles that have a 1-boundary inside s. Let V_s^2 be the set of rectangles that only have a 2-boundary intersecting s. (Note that the size of V_s^1 is bounded by $\sqrt{n}$ but the size of V_s^2 can be almost n.) We draw a line segment along each 2-boundary of a rectangle in V_s^1. Moreover, we draw a line segment along each $\sqrt{n}$-th 2-boundary of a rectangle in V_s^2. In this way s is partitioned into $\leq 4\sqrt{n}$ rectangular cells.

Lemma 3.1 *The partition has the following properties:*

1. *There are $O(n)$ cells.*
2. *Each rectangle of V partially covers at most $O(\sqrt{n})$ cells.*
3. *No cell contains vertices in its interior.*
4. *Each cell has at most $O(\sqrt{n})$ rectangles partially covering it.*

Proof. Property 1 follows from the fact that there are $2\sqrt{n}$ slabs, each with $4\sqrt{n}$ cells.

To show property 2 note that each vertical line cuts through one slab, i.e., through at most $4\sqrt{n}$ cells. Each horizontal line cuts in each slab through one cell. Hence, through at most $2\sqrt{n}$ cells in total. V partially covers a cell only if its boundary cuts through the cell. This boundary consists of two vertical and two horizontal line segments. Hence, it cuts through at most $12\sqrt{n}$ cells.

Property 3 follows from the fact that, if a vertex lies in the interior of a slab s, it belongs to a rectangle in V_s^1 and, hence, it will lie on the horizontal segment drawn through its 2-boundary, i.e., on the boundary of a cell.

Finally, property 4 follows from the fact that each slab contains at most $\sqrt{n}$ 1-boundaries and each cell at most $\sqrt{n}$ 2-boundaries. □

We will use the cells of this partition as leaves of the partition tree. For each slab s we construct a binary tree T_s that contains its cells ordered by x_2-coordinate in its leaves. Next we construct a tree T that contains the slab trees T_s ordered by x_1-coordinate in its leaves. See figure 2 for an example. Each node in the tree, consisting of T and the slab trees T_s, has an associated region, being the union of the cells at the leaves in the subtree. It is easy to see that such a region is a (possibly infinite) rectangle.

Lemma 3.2 *Let V be a set of n rectangles in the plane. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n)$ nodes.*

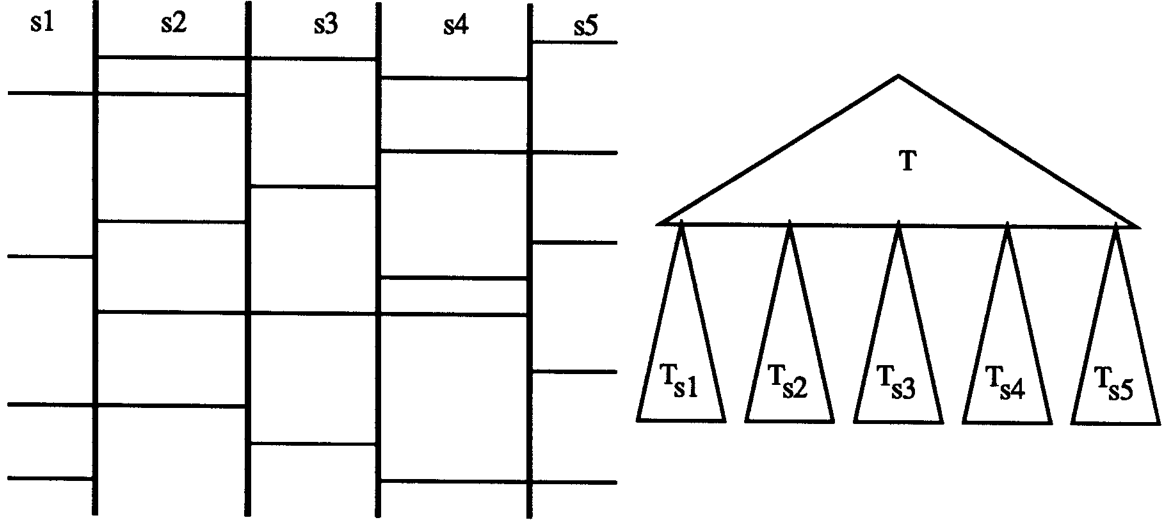

Figure 2: The 2-dimensional partition tree.

2. *Each rectangle is stored in* $O(\sqrt{n})$ *leaves.*

3. *Each rectangle influences* $O(\sqrt{n}\log n)$ TOT *fields.*

4. *No cell of a leaf contains vertices of rectangles in the interior.*

5. *Each leaf stores no more than* $O(\sqrt{n})$ *rectangles.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two. If the tree has $O(n)$ nodes its depth is bounded by $O(\log n)$. When a rectangle influences the TOT field of a node δ it partially covers $C_{father(\delta)}$, and there must be a leaf below $father(\delta)$ that is intersected by the rectangle. Hence, the number of internal nodes intersected by a rectangle is bounded by $O(\log n)$ times the number of leaves where the rectangle is stored. As a result the rectangle can only influence that number of TOT fields. □

It remains to show how the measure at a leaf is maintained when inserting and deleting rectangles. To this end we use the inclusion/exclusion principle. Note that, due to property 4, the rectangles stored at a leaf δ are 1-piles or 2-piles w.r.t. C_δ. In other words, they form a trellis. The measure of such a trellis can be maintained in the following way. Let V_1 be the projection of the 1-piles on the x_1-axis and let V_2 be the projection of the 2-piles on the x_2-axis. Let M_1 be the (1-dimensional) measure of V_1 and M_2 the (1-dimensional) measure of V_2. Assume that the cell C_δ has measure $L_1 \times L_2$. Now it is easy to see that the measure of the trellis is $M_1 \times L_2 + M_2 \times L_1 - M_1 \times M_2$. Hence, we just have to maintain the 1-dimensional

measure of V_1 and V_2. For this we can use a simple segment tree that uses linear storage and maintains the measure in time $O(\log n)$ per insertion and deletion (see [7]). So with each cell (leaf) δ we associate two segment trees S_1 and S_2. S_1 contains the projections of the 1-piles in δ on the x_1-axis and S_2 contains the projections of the 2-piles on the x_2-axis. Inserting (deleting) a rectangle at δ now consists of inserting (deleting) it in S_1 or S_2 (never in both). In this way M_1 and M_2 get updated and we obtain the new measure in the leaf.

Theorem 3.3 *The measure of a set of n 3-boxes in 3-dimensional space can be computed in time $O(n\sqrt{n}\log n)$ using $O(n\sqrt{n})$ storage.*

Proof. We use the plane sweep approach and maintain the partition tree described above. To insert or delete a rectangle we have to update $O(\sqrt{n}\log n)$ TOT fields. This takes time $O(\sqrt{n}\log n)$. Next, we have to insert or delete the rectangle at $O(\sqrt{n})$ leaves. At each such leaf this causes an insertion or deletion in a segment tree which takes $O(\log n)$. Hence, the total update time of the partition tree is $O(\sqrt{n}\log n)$.

The bound on the amount of storage required follows from the fact that the tree itself takes $O(n)$ storage and each leaf stores $O(\sqrt{n})$ information (according to property 5 of the above lemma). □

In section 5 we will show how to reduce the amount of storage used to $O(n)$.

4 Dynamic measure in multi-dimensional space

We will now generalize this method to d-dimensional space. To this end we will describe a d-dimensional partition tree, based on a cell decomposition of the d-dimensional space.

Definition 4.1 *A* slab at level $i = 1, ..., d$ *is a subset of* $\mathcal{R}^d$ *of the form* $I_1 \times I_2 \times ... \times I_i \times \mathcal{R}^{d-i}$ *where* $I_1, ..., I_i$ *are intervals of* $\mathcal{R}$.

Let V be the set of all d-boxes that will be inserted or deleted in the partition tree. We split the x_1-axis in $2\sqrt{n}$ intervals, each of whose interior contains $\leq \sqrt{n}$ 1-boundaries of boxes. This splits the d-dimensional space in $2\sqrt{n}$ slabs at level 1. For each such slab s let V_s be the set of d-boxes that partially cover s. We split V_s in two subsets: V_s^1 of d-boxes that have a 1-boundary inside s and V_s^2 of d-boxes that do not have a 1-boundary inside s. Note that $|V_s^1| \leq \sqrt{n}$. Each slab s we now split with respect to second coordinate. We split it at the 2-boundaries of each d-box in V_s^1 and we split it at every $\sqrt{n}$-th 2-boundary of d-boxes in V_s^2. As a result we split each slab s into $O(\sqrt{n})$ slabs at level 2. For each such slab s' let $V_{s'}$ be the set of d-boxes that partially cover s'. We again split $V_{s'}$ into two subsets: $V_{s'}^1$ of boxes that have a 1- or a 2-boundary intersecting $Int(s')$ and $V_{s'}^2$ of boxes that do not have a

1- nor a 2-boundary intersecting $Int(s')$. (Note that there are no boxes that have both a 1- and 2-boundary intersecting $Int(s')$.) Again $|V_{s'}^1| = O(\sqrt{n})$. We split s' into slabs at level 3 with respect to the third coordinate. Again, we split at each 3-boundary of boxes in $V_{s'}^1$ and at every $\sqrt{n}$-th 3-boundary of boxes in $V_{s'}^2$. In this way we continue for all coordinates.

Lemma 4.1 *The partition has the following properties:*

1. *There are $O(n^{d/2})$ cells.*
2. *Each d-box of V partially covers at most $O(n^{(d-1)/2})$ cells.*
3. *Each cell only contains piles in its interior.*
4. *Each cell has at most $O(\sqrt{n})$ d-boxes partially covering it.*

Proof. For each coordinate, every slab at level i is split into $O(\sqrt{n})$ slabs at level $i+1$. Hence the total number of cells we obtain is $O(\sqrt{n}^d) = O(n^{d/2})$.

If a d-box R partially covers a cell C then an i-boundary B of R cuts through C for some i ($1 \leq i \leq d$). At the moment we split slabs at level $i-1$ with respect to the ith coordinate there are $O(n^{(i-1)/2})$ slabs. Each of these slabs is split into $O(\sqrt{n})$ slabs at level i at this moment but B can cut through only one of them (because the cutting is done with respect to the ith coordinate axis). So after the ith step B still cuts through at most $O(n^{(i-1)/2})$ slabs at level i. In the next $(d-i)$ steps each slab at level i is cut into $O(n^{(d-i)/2})$ cells. So B will cut through at most $O(n^{(i-1)/2} \times n^{(d-i)/2}) = O(n^{(d-1)/2})$ cells.

Property 3 follows from the fact that no d-box can have both an i_1- and an i_2-boundary intersecting a slab at level i with $i_1 < i_2 \leq i$. Hence, no d-box has boundaries in more than one coordinate intersecting a slab at level d, i.e., a cell. So each d-box forms a pile in a cell.

The last property follows immediately from the way we split the slabs. □

We will use the cells of this partition as leaves of the partition tree. It is easy to see how the rest of the tree can be built on top of it. The tree consists of d "stages", where each stage consists of $O(\log n)$ levels of the tree. The top stage consists of a tree T that stores the $2\sqrt{n}$ slabs at level 1 in its leaves, sorted on x_1-coordinate. Each slab is represented by a slab tree that stores its slabs at level 2 (created in the second step) sorted by x_2-coordinate. For each of these slabs there is again a slab tree that stores its subdivision by x_3-coordinate, etc.

Lemma 4.2 *Let V be a set of n d-boxes in d-dimensional space. There exists a partition tree for storing any subset of V such that*

1. *The tree has $O(n^{d/2})$ nodes.*

2. *Each d-box is stored in* $O(n^{(d-1)/2})$ *leaves.*

3. *Each d-box influences* $O(n^{(d-1)/2}\log n)$ *TOT fields.*

4. *Each cell of a leaf only contains piles.*

5. *Each leaf stores no more than* $O(\sqrt{n})$ *d-boxes.*

Proof. Properties 1, 2, 4 and 5 follow immediately from the above lemma. The third property follows from the first two as the depth is again bounded by $O(\log n)$. □

It remains to show how the measure at a leaf is maintained when inserting and deleting d-boxes. To this end we again use the inclusion/exclusion principle. As stated in property 4, the d-boxes stored at a leaf δ are piles and form a d-dimensional trellis. Let V_i be the projection of the i-piles on the x_i-axis for each $1 \le i \le d$. Let M_i be the 1-dimensional measure of V_i. Let L_i be the length of C_δ in direction x_i. The following result is easy to proof:

Lemma 4.3 *The measure of the trellis is*

$$\sum_{1\le k\le d} (-1)^{k+1} Z_k$$

where

$$Z_k = \sum_{1\le j_1<\ldots<j_k\le d} \left(\prod_{1\le i\le k} M_{j_i} \prod_{l\ne j_i \text{ for any } i} L_l \right)$$

Although this might look quite complicated it is simply the inclusion/exclusion principle. E.g., for $d = 3$ the formula shows that the measure is

$$M_1L_2L_3 + L_1M_2L_3 + L_1L_2M_3 - M_1M_2L_3 - M_1L_2M_3 - L_1M_2M_3 + M_1M_2M_3.$$

When M_i is known for each i the measure can be computed in constant time (assuming d is a constant).

Hence, we just have to maintain the 1-dimensional measure of V_i for each i. For this we use d segment trees $S_1 \ldots S_d$, one for each dimension. An insertion or deletion in a leaf means inserting or deleting the i-pile in the correct segment tree S_i. In this way we obtain the updated measure M_i and we can recompute the above formula to obtain the new measure in the cell. This will take time $O(\log n)$ (assuming that d is a constant).

Lemma 4.4 *Updates in the d-dimensional partition tree take time* $O(n^{(d-1)/2}\log n)$ *and the tree uses* $O(n^{(d+1)/2})$ *storage.*

Proof. Follows from the above lemmas. □

Theorem 4.5 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n^{d/2})$ storage.*

Proof. We use the plane sweep approach and maintain a $(d-1)$-dimensional partition tree. So we have to perform $O(n)$ updates, each taking time $O(n^{(d-1-1)/2} \log n)$. The time bound follows. According to the preceding lemma, the structure uses $O(n^{(d-1+1)/2})$ storage. □

5 Reducing the amount of storage

In this section we will show how the amount of storage required can be reduces to $O(n)$. To this end we use an instance of the streaming technique introduced in Edelsbrunner and Overmars[3].

The idea of streaming is the following: Before hand we know what updates have to be performed and in what order. We can view the space sweep method as traversing in time (being the d-th coordinate). Each update in the structure has to be performed at a specific moment in time. Before each update we check what the current measure is and we multiply it by the time passed since the last update. Rather than building the structure and performing the updates one after the other, we will perform them simultaneously and construct parts of the data structure when we need them. When we are ready with the part we discard it again to free memory.

To formalize this, at any moment we are given a sequence of updates L over time and a region of the space C. This region corresponds to some node in the tree and L is the sequence of updates that will pass through this node. With each update in L we have stored the time at which it has to be performed. In the beginning C is the whole $(d-1)$-dimensional space and L is the complete list of updates, time being the d-th coordinate. A counter $MEAS$ will be used to collect all the measure found. In the beginning it will be set to 0.

The technique now works as follows: When all $(d-1)$-boxes in L are piles with respect to C (i.e., we are at a leaf in the partition tree) we construct $d-1$ segment trees. We perform all the updates on the segment trees and compute the $(d-1)$-measure in the cell after each update. These measures we multiply with the time period to the next update to obtain the d-measure in C. This d-measure we add to $MEAS$. This will take time $O(|L| \log n)$ and $O(|L|)$ storage. Afterwards we destroy all the structures.

When not all boxes are piles (i.e., we are at an internal node) we first compute during which periods of time C will be completely covered by one box. (This corresponds to the time when $TOT \neq 0$.) This can be done by simply walking along the list of updates and maintaining the number of boxes that cover C. Whenever

this number is larger than 0 C is covered. This takes time $O(|L|)$. We multiply the $(d-1)$-measure of C with the total amount of time C is covered and add it to $MEAS$. Next we change time by collapsing the covered periods into a single moment, performing all the updates in that period at the same moment. (This is necessary to avoid that lower in the tree measure will be found during these periods again and counted twice.) Boxes that are now inserted and deleted at the same moment are removed from L. Again this takes time $O(|L|)$ only.

Next we split C into two cells C_1 and C_2 in a way similar to as it would have been split in the partition tree. This can be done in the following way. Remember that in the first stage of the tree we split on x_1-coordinate, in the next stage on x_2-coordinate etc. until, in the last stage we split on x_{d-1}-coordinate. Hence, it is easy to remember on which coordinate we have to split at a particular moment. So assume we have to split on the i-th coordinate. There are two different splits we make: splits along i-boundaries in V_C^1 or splits along i-boundaries of boxes in V_C^2 (see the previous section). There is no problem in first making the splits along i-boundaries of V_C^1 and after that along i-boundaries in V_C^2. (The tree will get a depth that is at most twice as large.) So making a split can be done as follows:

- Let i be the current splitting coordinate. Split L into V_C^1 and V_C^2.
- If $V_C^1 \neq \emptyset$ then split along the median i-boundary in V_C^1.
- Else, if V_C^2 contains more than $\sqrt{n}$ i-boundaries split along the median i-boundary in V_C^2.
- Else, increase i and repeat the procedure.

Finding the splitting line can easily be done in time $O(|L|)$. It is easy to see that the resulting partition tree will still satisfy the properties in lemma 4.2.

Now we construct the list L_1 out of L containing the updates that influence C_1. In L we only keep the other updates. Hence, each update is either stored in L_1 or in L. We recursively call the routine for C_1 and L_1. When we get back from the recursive calls, we join L_1 and L to reconstruct L in its original form. Now we determine the list L_2 of updates that influence C_2, again leaving in L the other updates. We now recursively call the routine with C_2 and L_2. When we get back we again reconstruct L (to be used one level higher in the recursion). Note that during the whole process we never copy updates. We simply take a part of list L and send it down the recursion. When we get back we take another part of L and go again in recursion. As a result each update is stored at at most one place.

The method does essentially the same work as the original technique in which all updates are performed one after the other. In fact, it is more efficient because of two reasons. When the whole list consists of piles we immediately solve the problem rather than splitting till the list contains less than $\sqrt{n}$ boxes. Secondly, we don't consider boxes anymore when during their whole period of existence they are covered by some other box.

Theorem 5.1 *The measure of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n)$ storage.*

Proof. The amount of time used is essentially the same as in the case we performed the updates one after the other.

To estimate the amount of storage, note that each update is stored at most once in a list L. The bound follows. □

6 Extensions

The partition tree and method described above can also be used to solve a number of related problems. In this section we will briefly mention some of them.

It is well-known that the perimeter of the union of n rectangles in the plane can be computed in time $O(n \log n)$. (See e.g. [7, 9].) Computing the perimeter generalizes to computing the $(d-1)$-dimensional measure of the contour of the union of a set of d-boxes in d-dimensional space. The contour consists of parts of i-boundaries of boxes that do not lie in the interior of the union. We will only show how to compute the measure of the parts of d-boundaries of the contour. The measure of the i-boundaries for other i can easily be obtained by renumbering coordinates. The total measure of the boundary is obviously the sum of the measures of the i-boundaries in the contour for all $1 \leq i \leq d$.

To compute the measure of the d-boundaries of the contour we use exactly the same method as in section 4. We move a sweep plane along the d-th coordinate axis and maintain the measure of the intersection. At any d-boundary where the sweep-plane halts we update the $(d-1)$-dimensional measure as in section 4. The part of this d-boundary that is part of the contour is obviously the absolute value of the difference of the old and the new measure. (Except when more boundaries have the same d'th coordinate value. In this case some care has to be taken. The procedure below correctly treats those cases.) To be precise, the main algorithm (as described in section 2) is changed as follows:

S:=$\emptyset$;
$MEAS$:=0;
for i:=1 **to** n' **do**
  M:=$(d-1)$-dimensional measure of boxes in S;
  Insert all d-boundaries of boxes that start at a_i in S;
  M_+:=$(d-1)$-dimensional measure of boxes in S;
  Delete all d-boundaries of boxes that end at a_i from S;
  M_-:=$(d-1)$-dimensional measure of boxes in S;
  $MEAS$:=$MEAS + (M_+ - M) + (M_+ - M_-)$
end;

S is again stored as a partition tree and maintained in exactly the same way. The correctness of the method is easily established. This lead to the following result:

Theorem 6.1 *The measure of the contour of the union of a set of n d-boxes in d-dimensional space can be computed in time $O(n^{d/2} \log n)$ using $O(n^{d/2})$ storage.*

The method can also be used to compute the measure of lower-dimensional parts of the contour. It is unclear how streaming can be applied here to reduce storage.

As a second application consider the following query problem: Given a set of d-boxes in d-dimensional space, store them such that for a given query box R it can efficiently be determined whether R is completely covered by the d-boxes.

To solve this problem we store the d-boxes in a d-dimensional partition tree. A query is performed using the procedure filled, described below. It gets two arguments, a node δ and the query rectangle R and returns whether the part of R inside C_δ is fully covered by d-boxes. Calling the routine with δ the root of the tree gives the required answer.

```
procedure filled (δ, R):boolean;
    if R completely covers C_δ then
        return M_δ = measure of C_δ
    else if R partially covers C_δ then
        if δ is a leaf then
            search the segment trees to see whether in at least
            one of them the projection of R is fully covered;
            return the result
        else
            return filled(lson_δ,R) and filled(rson_δ,R)
    else
        return true
end;
```

The correctness of the method can easily be established. Searching the segment trees in a leaf takes time $O(\log n)$. This has to be done in at most $O(n^{(d-1)/2})$ leaves. The total number of internal nodes visited is bounded by $O(n^{(d-1)/2} \log n)$. The following theorem follows.

Theorem 6.2 *Let V be a set of n d-boxes in d-dimensional space. One can store V using $O(n^{(d+1)/2})$ storage, such that for a given d-box R one can determine in time $O(n^{(d-1)/2} \log n)$ whether R is completely covered by the boxes in V.*

The method can easily be extended to compute the measure of the union of the d-boxes restricted to a given box R, in the same time bounds. Updates from a fixed set of boxes can be performed in time $O(n^{(d-1)/2} \log n)$ using the same method as described for maintaining the measure.

Other applications exist. For example, it is possible to use the techniques given here to determine contours and i-contours (contour of the area covered by at least or precisely i d-boxes).

7 Conclusions

We have given a new solution to Klee's measure problem that is much more efficient than previously known results, improving the time bound from $O(n^{d-1})$ to $O(n^{d/2} \log n)$. The technique uses some new ideas, including a result on partitioning space, a new type of partition tree and the use of the inclusion/exclusion principle. Streaming was applied to reduce the amount of storage used to $O(n)$.

The dynamic data structure we presented for dynamically maintaining the measure can be used for other problems as well. As we have shown it is very simple to compute e.g. the perimeter. Moreover, the structure gives a compact representation of the shape of the set of d-boxes. This can be used to answer certain classes of queries efficiently.

Some open problems remain. First of all, it might be possible to shave off the factor of $\log n$. But, in fact, there is no reason to believe that the method is even near optimal. Improvements or lower bounds should be worked on. It is also interesting to look at the measure of other objects. For example the best bound known for computing the measure of the union of a set of triangles is $O(n^2)$.

References

[1] Bentley, J.L., Multidimensional binary search trees used for associated searching, *Comm. ACM* **18** (1975), 509–517.

[2] Bentley, J.L., Algorithms for Klee's rectangle problem, Unpublished notes, Dept. of Computer Science, CMU, 1977.

[3] Edelsbrunner, H., and M.H. Overmars, Batched dynamic solutions to decomposable searching problems, *J. Algorithms* **6** (1985), 515–542.

[4] Fredman, M.L., and B. Weide, The complexity of computing the measure of $\cup[a_i, b_i]$, *Comm. ACM* **21** (1978), 540–544.

[5] Klee, V., Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps?, *Amer. Math. Montly* **84** (1977), 284–285.

[6] Overmars, M.H., and C.K. Yap, New upper bounds in Klee's measure problem (extended abstract), *Proc. 29th IEEE Symp. on Foundations of Computer Science,* 1988, pp. 550–556.

[7] Preparata, F.P., and M.I. Shamos, *Computational Geometry,* Springer-Verlag, 1985.

[8] van Leeuwen, J., and D. Wood, The measure problem for rectangular ranges in d-space, *J. Algorithms* **2** (1980), 282–300.

[9] Vitányi, P.M.B. and D. Wood, Computing the perimeter of a set of rectangles, Techn. Rep. 79-CS-23, Unit for Computer Science, McMaster University, 1979.

[10] Welzl, E., Partition trees for triangle counting and other range searching problems, *Proc. 4th ACM Symp. on Computational Geometry,* 1988, pp. 23–33.

[11] Willard, D.E., Polygon retrieval, *SIAM J. Computing* **11** (1982), 149–165.