Efficient 3-D Range Searching in External Memory

Darren Erik Vengroff*
{Brown, Duke} University
dev@cs.duke.edu

Jeffrey Scott Vitter†
Duke University
jsv@cs.duke.edu

Abstract

We present a new approach to designing data structures for the important problem of external-memory range searching in two and three dimensions. We construct data structures for answering range queries in $O((\log\log\log_B N)\log_B N + K/B)$ I/O operations, where N is the number of points in the data structure, B is the I/O block size, and K is the number of points in the answer to the query. We base our data structures on the novel concept of B-approximate boundaries, which are manifolds that partition space into regions based on the output size of queries at points within the space.

Our data structures answer a longstanding open problem by providing three dimensional results comparable to those provided by [8, 10] for the two dimensional case, though completely new techniques are used. Ours is the first 3-D range search data structure that simultaneously achieves both a base-B logarithmic search overhead (namely, $(\log\log\log_B N)\log_B N$) and a fully blocked output component (namely, K/B). This gives us an overall I/O complexity extremely close to the well-known lower bound of $\Omega(\log_B N + K/B)$. The space usage is more than linear by a logarithmic or polylogarithmic factor, depending on type of range search.

*Supported in part by the U.S. Army Research Office under grant DAAH04–93–G–0076 and by the National Science Foundation under grant DMR–9217290. Portions of this work were conducted while visiting the University of Michigan.

†Supported in part by the National Science Foundation under grant CCR–9522047, and by the U.S. Army Research Office under grant DAAH04–93–G–0076.

1 Introduction

A *range search data structure* is a data structure that stores points in d-dimensional space and answers orthogonal range queries. Each such query is specified as a hyper-rectangle of the form $\rho = [x_1, x_1'] \times [x_2, x_2'] \times \cdots \times [x_d, x_d']$. The output of the query is the set of points in the data structure that are contained in ρ.

Range searching is a fundamental primitive in several large-scale applications, including spatial databases and geographic information systems (GIS) [1, 4, 6, 9, 11], graphics [3], indexing in object-oriented databases [5, 7], and constraint logic programming [7]. When the data are too large to fit in main memory and must reside on disk, the Input/Output (I/O) communication can become a very severe bottleneck.

In d-dimensional space, we define a $(s_1, s_2, \ldots, s_d)$-*sided range query*, where each $s_i \in \{1, 2\}$, to be an orthogonal range query with s_i sides in the x_i dimension. For example, the range query $[3, 5] \times [4, \infty]$ is a $(2, 1)$-sided range query, since there are two sides in the x_1 dimension (namely, $3 \le x_1$ and $x_1 \le 5$) but only one side in the x_2 dimension (namely, $x_2 \ge 4$). In the two-dimensional cases studied in [5, 8, 10], the authors use the terms "two-sided," "three-sided," and "four-sided" range query to mean what we call $(1, 1)$-sided, $(2, 1)$-sided, and $(2, 2)$-sided queries, respectively. Range search data structures, both for the general case of range queries and for the various special cases

defined above, have several important systems and database applications, as discussed in the previous references.

The I/O-efficiency of external-memory query data structures is measured in terms of the number of blocks of data that have to be transferred from the disk to the main memory in order to compute the answer to a query. Each block that is transferred contains B items from contiguous locations within a track on the surface of the disk. We assume that the entire data structure initially resides on disk.

The primary challenge in devising efficient data structures for range searching is twofold; we want a query overhead logarithmic (with base B) in the number of points stored in the data structure, and an output complexity of $O(K/B)$ when K points are output, thus taking full advantage of blocking. Results of this form have been achieved for two dimensions [8, 10], but the techniques used are inherently grounded in the two dimensionality of the problem and do not generalize to three dimensions. It has thus remained an open problem whether I/O-optimal data structures for range searching in higher dimensions exist. In this work, we present a new approach that, for the first time, combines nearly optimal query overhead with fully blocked output. The complexity of processing a query with our new data structures is

$$O\left((\log\log\log_B N)\log_B N + \frac{K}{B}\right),$$

which is within very close range of the well known lower bound of $\Omega(\log_B N + \frac{K}{B})$.

2 Data Structures for Open Queries in Two and Three Dimensions

An *open query* in d dimensions is a range query that is one-sided in all d dimensions. We assume without loss of generality that a query that is open in a dimension x_i is open in the positive x_i direction. A $(1, 1, \ldots, 1)$-sided query can thus be specified by a single corner point q. The answer to the query is the set of all stored points that dominate q. We say that a point $p' = (x'_1, x'_2, \ldots, x'_d)$ *dominates* a point $p = (x_1, x_2, \ldots, x_d)$ if $x'_i \geq x_i$ for all $1 \leq i \leq d$. We say that p' *strictly dominates* p if all the inequalities are strict, that is, if $x'_i > x_i$ for all $1 \leq i \leq d$. As will be seen in Section 3, our data structure for answering all forms of range queries in three dimensions relies heavily on the availability of an efficient data structure for answering $(1, 1, 1)$-sided open queries. Our construction of a data structure for $(1, 1, 1)$-sided queries relies on the fundamental notion of a B-approximate boundary. We discuss the structure, construction, and applications of B-approximate boundaries in Sections 2.1– 2.4.

2.1 B-Approximate Boundaries in Two Dimensions

A *B-approximate boundary* in d dimensions is a $(d-1)$-dimensional, monotone, orthogonal manifold $\mathcal{M}$ that partitions a set $\mathcal{S}$ of N points into two sets $\mathcal{S}^+$ and $\mathcal{S}^-$, such that

- Every point on $\mathcal{M}$ is dominated by at least B points in $\mathcal{S}^+$.
- Every point on $\mathcal{M}$ is strictly dominated by no more than $2B$ points in $\mathcal{S}^+$.
- Every point in $\mathcal{S}^-$ is dominated by some point on $\mathcal{M}$.

An example of a B-approximate boundary in two dimensions is shown in Figure 1.

Clearly, a B-approximate boundary is not a unique structure. A set of points may be partitioned appropriately by any one of a large number of manifolds. Algorithm 1 is an algorithm which constructs one such boundary for a given set of points. It does so by moving a point in the plane in a stair-step fashion such that it never dominates too many or too few points. The B-approximate boundary shown in Figure 1 was constructed using Algorithm 1 with $B = 4$.

A B-approximate boundary $\mathcal{M}$ in 2-D constructed using Algorithm 1 has a number of important properties, which are summarized in the following lemmas:

Lemma 1 *The number of inward corners in $\mathcal{M}$ is at most $|\mathcal{S}^+|/B$.*

Proof: An inward corner is formed in Algorithm 1 each time we begin to move p in the positive y direction in Step (4) 1 and at the end when $x = 0$.

(1) $p \leftarrow (0,0)$;
(2) Move p in the positive x direction until it dominates $\leq 2B$ points;
(3) **while** $p.x \neq 0$ **do**
(4) Move p in the positive y direction as long as p is dominated by $\geq B$ points;
(5) Let $\mathcal{D}$ be the set of points of the form (x, y');
(6) Move in the negative x direction until p is dominated by $\geq 2B$ points not in $\mathcal{D}$;
(7) **od**
(8) $\mathcal{M} \leftarrow$ the path traced out by p in steps (3)–(7);

Algorithm 1: An algorithm for generating a B-approximate boundary for a set of points $\mathcal{S}$ in two dimensions. We assume, without loss of generality, that all points in $\mathcal{S}$ lie in the first quadrant of the (x, y) plane. If this is not the case, simple translation suffices to make it so.

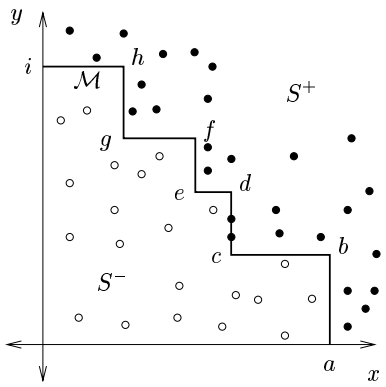

Figure 1: A B-approximate boundary in two dimensions. In this example, $B = 4$. Every point $m \in \mathcal{M}$ is dominated by between four and eight points in $\mathcal{S}^+$. The points of $\mathcal{S}$ "on" or "to the right of" $\mathcal{M}$ are in $\mathcal{S}^+$, and points in $\mathcal{S}$ "to the left of" $\mathcal{M}$ are in $\mathcal{S}^-$. Corners a, c, e, g, and i are inward corners, and the other corners b, d, f, and h are outward corners.

Step (4) can be executed at most once for every B points in $\mathcal{S}^+$. The result follows easily. □

Lemma 2 *We can block the elements of $\mathcal{S}^+$ into blocks of size B such that for any query point $m \in \mathcal{M}$, there is a set of at most $\lceil 2K/B \rceil$ blocks that contain the K points of $\mathcal{S}^+$ that dominate m. The total space used for the blocking is at most $4|\mathcal{S}^+|/B$ blocks.*

Proof: For each inward corner $c = (x, y)$ of $\mathcal{M}$, we group the $\leq 2B$ points that strictly dominate c into at most two blocks. In addition, we block together the points that dominate c and have the same x or y value as c. All such points are associated with only one inward corner w.r.t. x and one w.r.t. y; we block the horizontal points separately from the vertical points, in order of decreasing distance from c. By Lemma 1, there are at most $|\mathcal{S}^+|/B$ inward corners. Thus at most $2|\mathcal{S}^+|/B$ blocks are used for strictly dominating points and another $2|\mathcal{S}^+|/B$ blocks for the dominating points sharing a common x or y value. The total number of blocks is therefore at most $4|\mathcal{S}^+|/B$.

Any query point m in $\mathcal{S}^+$ dominates some inward corner c whose output we have explicitly blocked. To answer the query at m, we read the two blocks of points that strictly dominate c and report those that dominate m. If m shares the same x or y value as c, we also read the appropriate blocks of such points that dominate c, ordered by decreasing distance from c. The number of blocks read is thus at most $\lceil 2K/B \rceil$. □

2.2 A Data Structure for $(1,1)$-Sided 2-D Queries

A single B-approximate boundary can only answer a very specific set of $(1,1)$-sided queries. A carefully chosen set of B-approximate boundaries, however, can be used to answer general $(1,1)$-sided queries. We define a set of *layered approximate boundaries* to be a set of $2^i B$-approximate boundaries, one for each $i \in \{0, 1, 2, \ldots, \lfloor \lg(N/B) \rfloor\}$. Each such boundary is constructed by the method described in the proof of Lemma 2. Let $\mathcal{S}_i^+$ be the set of points in $\mathcal{S}$ that dominate one or more

points on the 2^iB-approximate boundary and let $\mathcal{S}_i^- = \mathcal{S} - \mathcal{S}_i^+$. Let $\mathcal{P}_i^+$ be the region of the plane consisting of all points which dominate some point on the 2^iB-approximate boundary. Let $\mathcal{P}_i^-$ be the set of points in the plane not contained in $\mathcal{P}_i^+$. Note that $\mathcal{S}_i^+ \subset \mathcal{P}_i^+$ and $\mathcal{S}_i^- \subset \mathcal{P}_i^-$.

The total space used by a set of layered approximate boundaries is given by the following lemma:

Lemma 3 *The total space usage for a complete set of layered approximate boundaries in two dimensions is* $O(\frac{N}{B}\log\frac{N}{B})$ *blocks.*

Proof: For each i, $\mathcal{S}_i^+$ can be blocked optimally in $O(|\mathcal{S}_i^+|/2^iB)$ blocks of size 2^iB, as shown by Lemma 2. Each block of size 2^iB can clearly be represented by 2^i blocks each of size B; thus $\mathcal{S}_i^+$ can be represented in $O(|\mathcal{S}_i^+|/B)$ blocks of size B. Any particular point in $\mathcal{S}$ can appear in $\mathcal{S}_i^+$ for any number of the approximate boundaries, but since there are only $O(\log\frac{N}{B})$ approximate boundaries, we have $\sum_i |\mathcal{S}_i^+| = O(N\log\frac{N}{B})$. Thus the total space usage is $O(\frac{N}{B}\log\frac{N}{B})$ blocks of size B. □

We can use a set of layered approximate boundaries to answer a $(1,1)$-sided range query defined by corner point q using a very simple algorithm that performs a search over the layers. The goal of the search is to find a value of i such that $q \in \mathcal{P}_i^+$ and $q \in \mathcal{P}_{i-1}^-$. We denote by $\mathcal{M}^*$ the 2^iB-approximate boundary for such a value of i, as illustrated in Figure 2.

To determine $\mathcal{M}^*$ we first check, via a B-way search using $\log_B N$ I/Os, as described below, whether $q \in \mathcal{P}_i^+$ for $i = \lg\log_B N$. If $q \in \mathcal{P}_i^+$ we do a binary search over the $\lg\log_B N$ layers between the B-approximate boundary and the $(\log_B N)B$-approximate boundary to find $\mathcal{M}^*$. Otherwise, if $q \notin \mathcal{P}_i^+$, we check $i = 1 + \lg\log_B N$, $i = 2 + \lg\log_B N$, $i = 3 + \lg\log_B N$, and so on, until we find a 2^iB-approximate boundary $\mathcal{M}^*$ for which $q \in \mathcal{P}_i^+$.

In order to determine whether $q \in \mathcal{P}_i^+$ for a given approximate boundary $\mathcal{M}$, we use a B-tree over the y coordinates of the corners of $\mathcal{M}$. Using this B-tree, we locate the two consecutive corners between which the y coordinate of q falls, and then test whether the x coordinate of q is greater than their shared x coordinate. If so, then $q \in \mathcal{P}_i^+$.

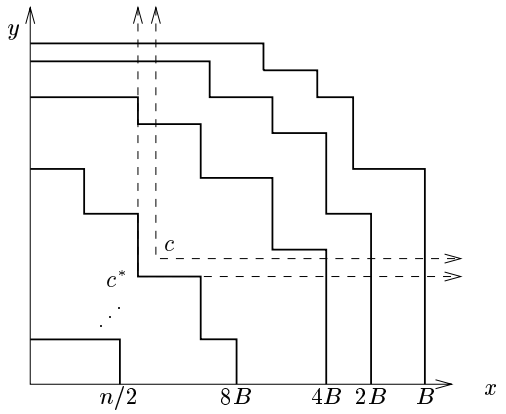

Figure 2: Determining $\mathcal{M}^*$ by locating the smallest i such that c dominates some point on the 2^iB-approximate boundary. In this case, $i = 3$ and $\mathcal{M}^*$ is the $8B$-approximate boundary.

The B-tree search for a given layer takes $O(\log_B N)$ I/Os, since a boundary can have up to N/B corners, and $\Theta(B)$-way branching is used.

Once the appropriate layer $\mathcal{M}^*$ has been located, we pick an inward corner (call it c^*) of $\mathcal{M}^*$ that the query point q dominates, as illustrated in Figure 2. We can determine a suitable c^* based on the outcome of the B-tree search. Given c^*, we produce a list of all points in $\mathcal{S}_i^+$ that are in the answer to a query with corner c^*. We then examine each of these points, reporting only those that lie inside the original query region (with corner q). Because $q \in \mathcal{P}_{i-1}^-$, the answer to the query at q contains at least $2^{i-1}B$ points. We can thus examine all 2^iB points that strictly dominate c^*, and at least half of them will be outputs to the query at q. In addition, if q has the same x value (respectively, y value) as c^*, we examine the points in $\mathcal{S}_i^+$ with the same x value (respectively, y value) as c^* that dominate c^*. Furthermore, all outputs to the query at q will be outputs to a query at c^*; thus there is no need to look elsewhere.

The following lemma summarizes the resulting time and space bounds:

Lemma 4 *A* $(1,1)$*-sided range query can be answered in* $O((\log\log\log_B N)\log_B N + K/B)$ *I/Os using* $O(\frac{N}{B}\log\frac{N}{B})$ *blocks of space.*

Proof: The space usage follows from Lemma 3.

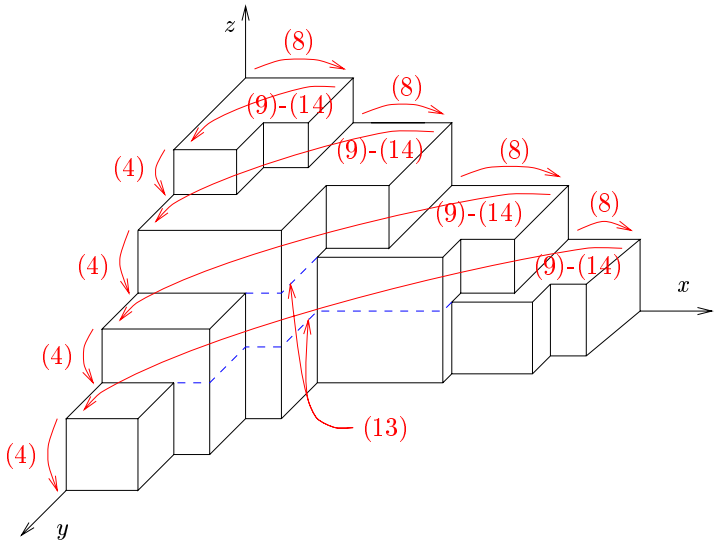

Figure 3: Building a B-approximate boundary in three dimensions. Portions of the boundary are labeled to indicate which step(s) of Algorithm 2 produced them. For clarity, the points outside the boundary are not shown.

If $q \in \mathcal{P}_i^+$ for $i = \lg\log_B N$ then the binary search uses $O((\log\log\log_B N)\log_B N)$ I/Os. Reporting the output takes $O(K/B)$ I/Os, as mentioned above.

If instead $q \notin \mathcal{P}_i^+$ for $i = \lg\log_B N$ then we search iteratively for the smallest $j \geq 1$ for which $q \in \mathcal{P}_{i+j}^+$. Each of the j additional searches requires a B-tree search; thus, all j searches take a total of $O(j \log_B N)$ I/Os. Reporting takes $\Theta(K/B) = \Theta(2^j \log_B N)$ I/Os. Because the cost of the additional searches is smaller than the cost of reporting the output, it does not enter into the overall I/O complexity of answering the query. □

2.3 B-Approximate Boundaries in Three Dimensions

In three dimensions, the construction of the B-approximate boundary consists of forming a series of level ridges. Each ridge is a zigzag path in a plane parallel to the x, y plane. Unlike the case in 2-D, the boundaries in 3-D can have $O(|\mathcal{S}^+|)$ inward corners and thus we cannot store the outputs associated with each inward corner explicitly, or else we would use $O(|\mathcal{S}^+|)$ disk blocks, which is excessive by a factor of B. The fundamental challenge is to develop an efficient mechanism for storing the outputs in some other manner.

First, let us consider how we will construct a B-approximate boundary in 3-D. This is done by running Algorithm 2 on $\mathcal{S}$. Algorithm 2 creates a series of ridges, each with a flat top, as shown in Figure 3. Portions of each ridge may coincide with the ridge of the previous level, as indicated by the dashed parts of Figure 3 created during Step 13. We call each such portion, which starts at an outward corner and extends to an inward corner, a *crooked* horizontal edge. The points shown in the shaded region of Figure 4 are associated with the *crooked* inward corner c at the end of a crooked horizontal edge; the points along the top portion of the shaded wedge region are not included, but the points along the vertical edge are included.

As mentioned above, we cannot efficiently represent the outputs of the queries at all inward corners of the B-approximate boundary by storing each corner's outputs explicitly, because the storage space may be nonoptimal by the large factor of B. Our only hope, therefore, to obtain optimal linear storage is to use an implicit representation, where duplication is avoided by use of pointers.

We store the output associated with each inward corner in one of two ways:

(1) { Let $\mathcal{R}$ be the current ridge. }
(2) $\mathcal{R} \leftarrow$ the single point $(0, 0, \infty)$;
(3) **while** *true* **do**
(4) Let $\mathcal{R}$ descend in the $-z$ direction until some point on $\mathcal{R}$ is dominated by $3B$ points in $\mathcal{S}$ or $z = 0$;
(5) **if** $\mathcal{R}.z = 0$ **then return fi**;
(7) $p \leftarrow$ the point on $\mathcal{R}$ with $y = 0$;
(8) Move p in the positive x direction until it dominates $\leq 2B$ points;
(9) **while** $p.x \neq 0$ **do**
(10) Move p in the positive y direction as long as p is dominated by $\geq B$ points;
(11) Let $\mathcal{D}$ be the set of points of the form (x, y');
(12) Move in the negative x direction until p hits $\mathcal{R}$ or p is dominated by $\geq 2B$ points not in $\mathcal{D}$;
(13) **if** p hit $\mathcal{R}$ **then**
(14) p follows $\mathcal{R}$ until it hits the x, z plane or dominates $\geq 2B$ points **fi**
(15) **od**;
(16) $\mathcal{R} \leftarrow$ the path traced out by p in Steps (8)–(14)
(17) **od**

Algorithm 2: An algorithm for generating a B-approximate boundary for a set of points $\mathcal{S}$ in three dimensions.

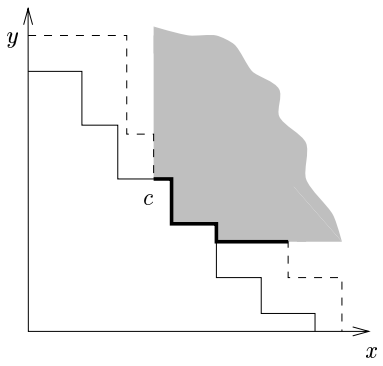

Figure 4: A crooked corner c. The solid line represents the $(i-1)$st ridge, and the dashed line represents the edge being constructed on ridge i. The bold path is the crooked horizontal edge of the ith ridge, which meets and runs along the $(i-1)$st ridge until the crooked corner c. The shaded area is the output area associated with the crooked corner c.

1. An explicit blocking of the outputs.
2. A pointer to an inward corner on a previous ridge whose outputs are explicitly blocked. Each inward corner that is explicitly blocked also has an extra block $\mathcal{A}$ containing auxiliary points, for use as described below.

Usually the output size of an inward corner is $2B$, but if there are several points with the same x value, this value can be arbitrarily high.

When constructing ridge i, we must decide which storage method to use for each inward corner. When considering an inward corner c on ridge i, let c^* be the inward corner on ridge $i-1$ with maximum y value less than or equal to that of c. If c^* does not have its outputs explicitly stored, we redefine c^* to be the corner on a previous ridge pointed to by c^*. Let $\mathcal{D}$ be the outputs of c that are not included in c^*'s output or in the auxiliary block $\mathcal{A}$ for c^*.

If there is more than one inward corner c that maps to the same c^*, we explicitly block the outputs of all such c. Otherwise, if there is room, we add $\mathcal{D}$ to the auxiliary block $\mathcal{A}$ and set c to point to c^*. If there isn't enough room then we explicitly block the outputs of c.

The following key lemma, based on a neat combinatorial analysis, shows that our method for storing the outputs of the B-approximate boundary is optimal for storage and query I/Os:

Lemma 5 *The outputs for all queries on a B-approximate boundary in three dimensions can be represented in $O(|\mathcal{S}^+|/B)$ blocks of size B so that all points in $\mathcal{S}$ that dominate any given point on*

the boundary are contained in $O(1)$ blocks.

Proof: For each inward corner c, the elements of $\mathcal{D}$ (the outputs of c not previously encountered on the preceding level) are fully blocked, either explicitly or in an auxiliary block. Therefore, for the purposes of proving our storage bound, it suffices to show that the sum over all inward corners c of the sizes of the sets $\mathcal{D}$ is $O(|\mathcal{S}^+|)$. Let us call the sum Σ; we need to show that $\Sigma = O(|\mathcal{S}^+|)$.

Consider each point p when it first "appears" in an output of some point on a ridge. Let ridge i be the first ridge for which p is an output point. Suppose that p appears in the outputs of k different inward corners on ridge i, as in in Figure 5; that is, p contributes k to Σ in order to represent p in k different outputs. If $k < 5$, then p contributes less than 5 to Σ. If $k \geq 5$, then we call p a "heavy contributor."

Our goal now is to prove that the total number of points contributed to Σ by heavy contributors is always small enough that Σ remains within the $O(|\mathcal{S}^+|)$ bound that we are trying to prove. The lemma below, which we prove later, shows that the existence of a heavy contributer implies the existence of many other points in $\mathcal{S}^+$:

Lemma 6 *Suppose that ridge i is the first ridge for which point p appears as an output, and suppose the p appears in the output of k inward corners on ridge i. Then the number of points in $\mathcal{S}^+$ whose x, y projection lies in the shaded region pictured in Figure 5 is at least $(k-3)B$. When $k \geq 5$, the number of points is at least $\frac{2}{5}kB$.*

Assuming for the moment the correctness of Lemma 6 (its proof will be given after the rest of the proof of Lemma 5 is completed), let $r \geq \frac{2}{5}kB$ be the number of points in the shaded region defined by p, as in Lemma 6. We charge each of the r points the value $k/r \leq 5/2B$, which pays for p's contribution of k to Σ. We now claim that each point accumulates at most five units of charge, and thus $\Sigma = O(|\mathcal{S}^+|)$.

Suppose some point q accumulates more than five units of charge. Then the number of times it was charged is $> 5/\frac{5}{2B} = 2B$. Consider the last ridge of $\mathcal{M}$ for which q was an output. The fundamental observation is that the ridge's inward corner that is dominated by q contains q and all the points p that caused q to be charged, which consists of more than $2B$ points. This is a contradiction by the construction of the ridge. (This yields a contradiction even for those inward corners that have more than $2B$ points, that is, when several points have the same x and z value.) □

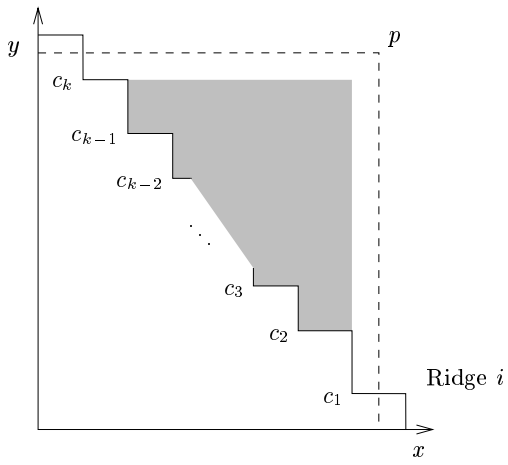

Figure 5: A point p (appearing for the first time as an output on ridge i) that dominates $k \geq 5$ inward corners on ridge i. See Lemmas 5 and 6.

Now all that remains to be done in order to complete the proof of Lemma 5 is to prove Lemma 6. This is done as follows:

Proof of Lemma 6: Suppose we extend the shaded region of Figure 5 in the positive y direction to infinity. Now we divide it into vertical strips S_i at the x coordinates of the corners $c_1, \ldots c_{k-1}$. This is shown in Figure 6. Each strip contains at least B points. Thus, all the strips together contain at least $(k-2)B$ points. Now consider the portion of the strips whose y coordinate exceeds that of c_k, namely, the portion above the dotted line in Figure 6. This entire region contains no more than B points that dominate the outward corner labeled c^* in Figure 6. Since the shaded region contains all the points in the strips that do not dominate c^*, it contains at least $(k-3)B$ points. Whenever $k \geq 5$, we have $(k-3)B \geq \frac{2}{5}kB$. □

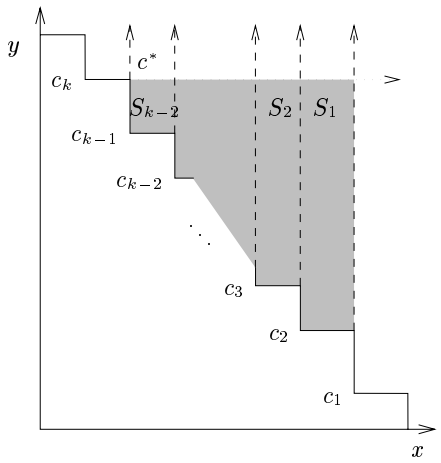

Figure 6: Strips used to compute a lower bound on the number of points that must exist to engender a single "heavy contributor." See Lemma 6.

2.4 A Data Structure for $(1,1,1)$-Sided 3-D Queries

To convert the B-approximate boundary into a full data structure for answering $(1,1,1)$-sided queries in three dimensions, we can use an approach similar to that used for two dimensions, namely forming a set of layered approximate boundaries. The difference is that when we attempt to generalize the binary search over the set of $O(\log \frac{N}{B})$ boundaries we find that determining whether or not the query corner q lies inside or outside a given approximate boundary is more difficult.

In 2-D we used a B-tree to locate q in one of the two dimensions of the layer, and then tested to see which side of the layer boundary it fell on. In 3-D, we need to locate q in two of the three dimensions before we can use a simple test in the third dimension.

The problem of locating q in the first two dimensions can be reduced to the problem of point location in a rectilinear decomposition of the plane. This can be done by projecting the edges of each ridge in an approximate boundary onto the x, y plane. Each region in the projected planar decomposition corresponds to a specific ridge, against whose z value we can test q's z value to determine whether q is above or below the ridge. It follows from the proof of Lemma 5 that the total number of non-crooked inward corners on a B-approximate boundary is $O(\mathcal{S}^+) = O(N)$. Thus the number of outward corners is also $O(N)$. Every corner of our rectilinear decomposition is located at the projection onto the x, y plane of one of these corners; thus the complexity of the decomposition is $O(N)$. The point location problem can be thus solved in $O(\log_B N)$ I/Os with a persistent B-tree [2] occupying $O(N/B)$ blocks.

Using techniques analogous to those of Section 2.2, we can build a set of layered approximate boundaries in three dimensions and to answer $(1,1,1)$-sided queries. Following the proof techniques used for two dimensions, we can prove the following lemma:

Lemma 7 *$(1,1,1)$-sided queries can be answered in*

$$O\left((\log\log\log_B N)\log\frac{N}{B} + K/B\right)$$

I/Os, where K is the number of points in the query region. The space used is $O(\frac{N}{B}\log\frac{N}{B})$ blocks.

3 Closed Queries

Up to this point, we have only considered queries that are bounded on one side in each dimension. We now show that with additional space, using techniques of [10], we can handle queries that bounded on both sides in one or more dimensions without additional asymptotic complexity.

The data structure we use for closed queries is called a *segmented layer tree* because it resembles a segment tree. A segmented layer tree for $(2,1,1)$-sided queries is a balanced binary tree over the x coordinates of the points is contains. Each leaf is a block containing B points with consecutive x coordinates. Internal nodes are grouped together into blocks of B nodes descending from a single node and occupying $\lg B$ levels of the tree. Each internal node v of the binary tree contains a pointer to two layered B approximate boundary data structures, one of which answers queries open in the positive x, y, and z directions and one which answers queries open in the negative x and positive y, and z directions. The points represented in these structures are those appearing in leaves below v. A segmented layer tree is illustrated in Figure 7.

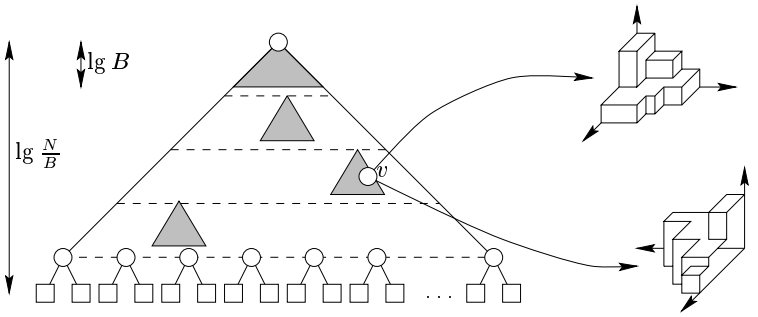

Figure 7: A segmented layer tree for answering $(2,1,1)$-sided queries. Each leaf holds B points and is stored in a single block. The shaded regions are examples of sets of B internal nodes that are stored together in the same block. Each internal node v points to two layered B-approximate boundary data structures over the points in leaves below v. One of them is open in the positive x direction and one in the negative x direction.

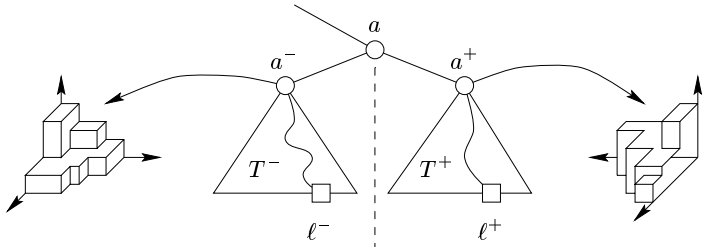

Figure 8: Searching a segmented layer tree to answer a $(2,1,1)$-sided query. Searching the layered approximate boundaries pointed to by a^- and a^+ find all the points in the subtrees T^- and T^+ respectively that are in the original query. All points in the original query are found in either T^- or T^+, thus all are found.

In order to answer a $(2,1,1)$-sided query using a segmented layer tree, we locate a pair of layered approximate boundaries that cover the query region and then search both of them. This is done by searching the main binary tree for the upper and lower bounds of the x range of the query. Let ℓ^- and ℓ^+ respectively be the leaves these searches travel to. Let a be the least common ancestor of ℓ^- and ℓ^+, and let a^- and a^+ be the left and right children of a. The layered approximate boundaries we search are the one open in the positive x direction pointed to by a^- and the one open in the negative x direction pointed to by a^+. The query process is illustrated in Figure 8. The I/O complexity of performing a search is given by the following lemma:

Lemma 8 *We can perform $(2,1,1)$-sided range queries in $O((\log\log\log_B N)\log_B N + K/B)$ I/Os using a segmented layer tree. The space required is $O(\frac{N}{B}\log^2\frac{N}{B})$.*

We can further extend the idea of a segmented layer tree to handle $(2,2,1)$-sided queries by replacing the pointers to layered approximate boundaries by pointers to segmented layer trees for $(2,1,1)$-sided queries. The space requirement increases by another factor of $O(\log_B N)$, but the asymptotic I/O complexity increases by only a constant factor. In a similar manner, we can extend the result to $(2,2,2)$-sided queries with another factor of $O(\log_B N)$ space overhead.

Lemma 9 *$(2,2,1)$-sided range queries can be performed in $O((\log\log\log_B N)\log_B N + K/B)$ I/Os using a segmented layer tree. The space required is $O(\frac{N}{B}\log^3\frac{N}{B})$.*

Lemma 10 *$(2,2,2)$-sided range queries can be performed in $O((\log\log\log_B N)\log_B N + K/B)$ I/Os using a segmented layer tree. The space required is $O(\frac{N}{B}\log^4 \frac{N}{B})$.*

4 Conclusions and Open Problems

We have, for the first time, presented data structures capable of answering range queries in three dimensions using

$$O\left((\log\log\log_B N)\log_B N + \frac{K}{B}\right)$$

I/Os. These results rely on a new data structuring technique, B-approximate boundaries. Previously known results typically required either an $O(\log N)$ query overhead or a $O(K)$ output term.

The most obvious open problems are whether our techniques can be augmented in order to remove the small $O(\log\log\log_B N)$ multiplicative factor on the query time overhead in order to make the search complexity optimal and whether the space usage can be reduced to linear. Additional open problems include dynamizing these data structures, which are all currently static, and extending the results to higher dimensions.

References

[1] R. F. Cromp. An intellegent information fusion system for handling the archiving and querying of terabyte-sized spatial databases. Technical Report TR–93–99, CESDIS, 1993.

[2] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Sys. Sci.*, 38:86–124, 1989.

[3] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, second edition, 1990.

[4] L. M. Haas and W. F. Cody. Exploiting extensible DBMS in integrated geographic information systems. In O. Günther and H.-J. Schek, editors, *Advances in Spatial Databases, Lecture Notes in Computer Science 525*, pages 423–450. Springer-Verlag, 1991.

[5] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. Indexing for data models with constraints and classes. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 233–243, 1993.

[6] R. Laurini and A. D. Thompson. *Fundamentals of Spatial Information Systems.* A.P.I.C. Series, Academic Press, New York, NY, 1992.

[7] S. Ramaswamy and P. C. Kanellakis. OODB indexing by class division. In *Proc. 1995 ACM International Conference on Management of Data*, 1995.

[8] S. Ramaswamy and S. Subramanian. Path caching: a technique for optimal external searching. *Proc. 13th ACM Symposium on Principles of Database Systems*, 1994.

[9] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison Wesley, MA, 1989.

[10] S. Subramanian and S. Ramaswamy. The p-range tree: A new data structure for range searching in secondary memory. In *Proc. Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, Jan. 1995.

[11] M. J. van Kreveld. Geographic information systems. Technical Report INF/DOC–95–01, Utrecht University, 1995.