Extending Rectangular Range Reporting with Query Sensitive Analysis

Robin Y. Flatland and Charles V. Stewart
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, 12180

Abstract

We present algorithms for reporting the new points incorporated by on-line sequences of nested rectangular range queries where the rectangular regions in a sequence may be aligned with any arbitrary axis. For each larger rectangular query, our algorithm performs partially ordered searches of a box-decomposition subdivision using information acquired from processing previous queries in the sequence. We provide a query sensitive analysis showing that our algorithm is most efficient for "targeted" queries, queries that generally target space containing a high density of data relative to the space immediately surrounding the query, such as are found in region growing surface reconstruction techniques in computer vision.

1 Introduction

We present an algorithm for reporting the new points incorporated by on-line sequences of nested rectangular range queries. We assume the rectangular regions in a sequence have the same orientation but this orientation may be different for each sequence and is not known in advance (see Figure 1). The problem is formally stated as follows:

Extending Rectangular Range Reporting Problem: Given a set of N points in R^d and an on-line sequence of d-dimensional rectangular query regions $Q_1, \ldots, Q_E$ aligned with some arbitrary orthogonal coordinate system and each Q_i completely containing Q_{i-1}, for the i(th) extended query, report the points in Q_i that are not in Q_{i-1}.

Here we consider the problem in two and three dimensions. Because we expect many sequences of extending queries on a static point set, our solution includes a preprocessing stage to organize the points into a search structure which facilitates the processing of extending queries.

Efficient solutions to a restricted version of this problem where all sequences of nested rectangular queries are aligned with a fixed coordinate system are found in [7]. Efficient solutions to the less restrictive problem considered here, however, appear unlikely. Rectangular range reporting (i.e. reporting the points inside a (single) rectangular query region with arbitrary orientation) reduces immediately to the extending rectangular range reporting problem by setting $E = 1$. Lower bound results [3] indicate that any solution to the rectangular range reporting problem either requires a large amount of space or has large query time. In particular, as shown in [3], any algorithm with $O(\log^c N + k)$ query time, for c arbitrarily large, would require $\Omega(N^{d-\epsilon})$ space, for all fixed $\epsilon > 0$.

In many applications, however, "hard" point sets and queries causing these pessimistic time or space bounds may seldom if ever arise. In such situations, the worst case analysis is not a representative measure of an algorithm's true query time. Expected case analysis is one alternative, but for many applications it is often difficult to derive realistic statistical models of the input. A second alternative is query sensitive analysis where the running time is expressed in terms of parameters capturing the local geometric complexity of the query. This has been successfully applied to other "hard" problems such as ray shooting [8] and high dimensional nearest neighbor queries [9].

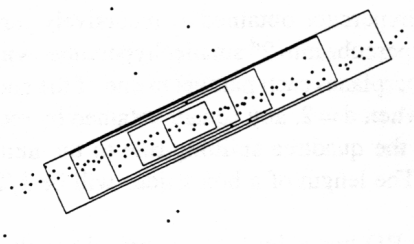

Figure 1:

For the query sensitive analysis, we characterize a query Q by three parameters k, α, and l: k is the number of data points inside Q, α is the distance to the point outside Q that is k(th) farthest from its boundary, and l is Q's diameter. We characterize a sequence of extending rectangular queries by the parameters k, α, and l of the final query. For $d = 2, 3$, using linear space, the overall time for our algorithm to process a sequence of E extending range queries is $O(\log N + C \log C + E)$, where $C = O(k + l/\alpha)$ when $d = 2$ and $C = O(k + (l/\alpha)^2)$ when $d = 3$. As a special case, our algorithm processes single queries ($E = 1$) in $O(\log N + C)$ time.

From the definition of C in the query time, our algorithm is most effective for what we call "targeted" queries. These are queries that target space containing a sufficiently large number of data points relative to the space immediately surrounding the query, making $C = O(k)$. In Section 3, we show further that when the point density is the same both inside and outside Q, $C = O(k)$ in the

expected case as long as Q is not extremely narrow.

One application encountering targeted queries is surface reconstruction techniques in computer vision that incrementally "grow" surfaces in three dimensional point data acquired from a range sensor. A surface is grown by collecting the new points included in an incrementally enlarging region which tightly "hugs" a set of points from a (approximately) planar surface. (Figure 1 is a two dimensional example of the type of queries made by region growing techniques.) Typically the free space immediately surrounding the region contains very few points. We suspect that targeted report queries arise in other applications as well since it is not uncommon that apriori information is available to direct queries to interesting data clusters.

2 Box-decomposition Subdivision

Our algorithm processes extending queries using partially ordered searches of a *box-decomposition* of space into rectilinear regions. The basic concept of box-decompositions was introduced by [4] and has since appeared in many different forms (see [2] for a review). We begin by describing the construction of a relatively simple box-decomposition tree adapted from [2] whose leaves define a box-decomposition (BD) subdivision.

WLOG, we assume all data points are contained within an axis-aligned, d dimensional, unit hypercube. The structure of BD subdivisions is defined in terms of *boxes*. A box is any hypercube obtained by recursively partitioning the unit hypercube into 2^d smaller hypercubes with d orthogonal hyperplanes each parallel to one of the coordinate axis (*e.g.*, when $d = 2$, any square obtained by recursively applying the quadtree splitting rule to the unit square is a box). The length of a box's sides will be $1/2^j$, for some $j \geq 0$.

The BD tree is built by recursively applying *split* and *shrink* operations to the unit hypercube. A *split* operation partitions a box b into 2^d smaller boxes with d hyperplanes. A *shrink* operation partitions a box b into an *inner* box and a *doughnut* region; the inner box is some smaller box contained inside b and the doughnut is the set difference of b and the inner box. b is the *outer* box of the doughnut.

The root of the BD tree is associated with the unit hypercube and all points contained inside it. Building the tree begins by recursively splitting box b associated with node v into 2^d smaller boxes. v, in this case a *splitting node*, is assigned 2^d children nodes, each associated with one of the smaller boxes and the points contained inside it. If, however, splitting b will result in all but one of the smaller boxes containing zero points, then a shrink operation is applied instead. The inner box of the shrink operation is chosen to be the smallest box inside b containing all of b's points; the doughnut region contains no points. v, in this case a *shrinking node*, is assigned two children nodes, one associated with the inner box and containing the points inside and the other associated with the doughnut and containing zero points. Splitting and shrinking stops when a node contains one or zero points.

The leaves of the BD tree partition the unit square into boxes and doughnuts, called the *cells* of the subdivision; each cell contains zero or one data points. For uniformity, every cell is thought of as the set difference of an outer box and zero or one inner boxes; for *box cells*, the outer box is the cell itself and it has no inner box, whereas *doughnut cells* have an outer box and one inner box. The *size* of a box is the length of one of its sides. The *size* of a cell is the size of its outer box. For fixed d, this BD subdivision has the following five properties. (See [2] for proofs of similar properties.) These properties are common to many BD subdivisions with the only differences being in the constant factors. We have therefore expressed them more generally than necessary for this subdivision so that they may be attributed to other BD subdivisions as well.

P1 The BD subdivision is of size $O(N)$.

P2 Every cell is the set theoretic difference of a unique outer box and up to a constant number of inner boxes.

P3 There exists a many-to-one mapping from cells to data points such that a cell of size s maps to a data point located within distance $c_1 s$ of its outer box, where c_1 is a constant, such that at most a constant number of cells map to each data point.

P4 The number of cells of size at least s intersecting the boundary of a convex region of diameter l is at most $c_3\lceil l/s\rceil^{d-1}$, where c_3 is a constant.

P5 Every cell of size s occupies a region that includes a box of size at least s/c_2, where c_2 is a constant.

Note that for property [P5], in the BD subdivision described above, the region occupied by a box cell of size s (obviously) includes a box of size s; the region occupied by a doughnut cell of size s includes three boxes of size $s/2$ since its inner box is of size $s/2$ or smaller.

For our asymptotic analysis, we require the additional bounded adjacency property below, which is not guaranteed for the simple BD subdivision described above, but is for the more sophisticated, "smoothed" versions of the subdivision found in [8] and [6].

P6 Every cell is adjacent to at most a constant number of other cells.

The smoothed subdivision in [8] is only given for $d = 2$, but generalizes to higher dimensions (see [2], revised paper). When $d = 2$, both smoothed subdivisions may be constructed in $O(N \log N)$ time; when $d = 3$, to our knowledge, the best known bound on their preprocessing is $O(N^2)$.

For our asymptotic analysis, we also require a linear space structure for processing $O(\log N)$ point location queries in the smoothed BD subdivision. For $d = 2$, $O(\log N)$ point location queries in a planar subdivision can be done with a linear space data structure requiring $O(N \log N)$ preprocessing (see [10]). For $d = 3$, we first

further partition each doughnut cell into at most a constant number of rectangular regions and use the linear space data structure in [5] for $O(\log N)$ point location queries in 3D rectangular subdivisions requiring $O(N \log N)$ expected preprocessing time.

In practice, one would probably prefer the balanced version of the BD tree and the subdivision defined by its leaves as introduced in [2]. For fixed d, it satisfies properties [P1]-[P5] above, can be built in $O(N \log N)$ time, and handles point location queries in $O(\log N)$ time. Although it does not satisfy property [P6], the average number of adjacent cells *is* bounded by a constant (see [6]).

3 Two Dimensional Extending Queries

In this section we describe the two dimensional version of our extending rectangular range reporting algorithm. We assume that during a preprocessing stage an adjacency graph representation of a two dimensional, smoothed box-decomposition subdivision and a point location data structure satisfying all the properties of the previous section are constructed. We describe our algorithm inductively, concentrating on how to process the i(th) extension following the completion of the $(i-1)$(th) and concluding briefly with a discussion of the initial case.

Let C_i be the set of cells intersecting query Q_i, and let B_i be the set of cells adjacent to cells in C_i but not themselves in C_i. We call the cells in B_i the *surrounding* cells.

To process the i(th) extension, we assume inductively that having completed the $(i-1)$(th) extension we have a data structure S_2 (described below) containing all edges of cells B_{i-1} and all points located in cells C_{i-1} not yet reported (in other words, points contained in cells C_{i-1} but not located in Q_{i-1}). In the i(th) extension, we extract from S_2 points it contains that are in Q_i and edges it contains whose cells intersect Q_i. The extracted points are just reported. The edges extracted are used to start searches of the BD adjacency graph. These searches (collectively) visit each cell in $C_i - C_{i-1}$ and report its point if it is located in $Q_i - Q_{i-1}$. Since every point in $Q_i - Q_{i-1}$ is either a previously unreported point located in a cell of C_{i-1} or a point located in a cell of $C_i - C_{i-1}$, all the appropriate points are reported. In the remainder of this section we detail this algorithm and analyze it.

We begin by describing the searches started from cells B_{i-1} intersecting Q_i. Prior to starting the searches in the i(th) extension, only cells in C_{i-1} have been visited. For each cell c in B_{i-1} intersecting Q_i, c's search is a pruned depth first search of the adjacency graph, starting at c and stopping at visited cells and cells not in $C_i - C_{i-1}$. (Note that c must be in $C_i - C_{i-1}$.) Since each cell has $O(1)$ edges (due to property [P2]), it is easy to determine in constant time if a cell is in $C_i - C_{i-1}$ by just checking if it intersects Q_i but not Q_{i-1}. Upon completing these searches it is not too difficult to see that every cell in $C_i - C_{i-1}$ is visited since there must exist a path in the adjacency graph from each cell in $C_{i-1} - C_i$ to some cell in B_{i-1} that intersects Q_i.

The heart of our algorithm is the structure of S_2, but before describing it, we first motivate why we need it. An obvious solution to the extending rectangle reporting problem is, in the i(th) extension, to start from any cell intersecting the boundary of Q_{i-1} and search the adjacency graph visiting all cells intersecting $Q_i - Q_{i-1}$ and report their points if they are located in $Q_i - Q_{i-1}$. This solution, however, is inefficient when queries grow by small enough increments to cause the same cells to be visited on many extensions. Our data structure S_2 avoids this problem by imposing a partial order on the search so that only the new cells (and new points) intersecting the current query are examined. It is not immediately obvious how to order the search since each query in the sequence may have a different aspect ratio and hence there is no single distance function for ordering the cells.

S_2 is organized to efficiently find the cells B_{i-1} intersecting Q_i in the i(th) extension. Since cells B_{i-1} do not intersect Q_{i-1}, if they intersect Q_i then some part of their boundary (one or more of their edges) must intersect Q_i. Therefore, we only need to determine the edges of cells B_{i-1} that intersect Q_i. Let x' and y' be the axes of a coordinate system aligned with the sequence of queries (see Figure 2), and let $P_{x'}(r)$ be the projection of any region $r \subset R^2$ onto the axis x', let $P_{y'}(r)$ be r's projection onto y', and let $P_\theta(r)$ be r's projection onto a line with orientation θ. The following lemma will help us find these edges and therefore dictate the organization of S_2.

Lemma 3.1 *An edge e with orientation $\theta + \pi/2$ intersects rectangular query Q_i iff*

1. $P_{x'}(e) \cap P_{x'}(Q_i) \neq \emptyset$,
2. $P_{y'}(e) \cap P_{y'}(Q_i) \neq \emptyset$, and
3. $P_\theta(e) \cap P_\theta(Q_i) \neq \emptyset$.

Proof: Proving that the three conditions hold if e intersects Q_i is trivial. Now suppose (for the sake of contradiction) that the three conditions hold but e does not intersect Q_i. Conditions 1 and 2 imply that e has at least one point in region $R_{y',u}$ or $R_{y',l}$ and at least one point in region $R_{x',u}$ or $R_{x',l}$ (see Figure 2 where e has a point in $R_{y',u}$ and one in $R_{x',l}$) and the closest point to e in Q_i is the corner of Q_i, call it c, located on the border of both regions (again, see Figure 2). c is located at some distance $\delta > 0$ from e. No point of Q_i may be located less than distance δ from the line containing e (or else c would not be the closest point to e). But then condition 3 must not hold, resulting in a contradiction. □

Now we can define S_2. It is a three stage "pipeline," each stage corresponding to one of the three conditions above. We assume inductively that having processed the $(i-1)$(th) extension, S_2 contains the edges of cells B_{i-1}, and these edges are stored in S_2 as follows: The first stage of S_2, call it H_1, holds all edges whose projections on the x' axis do not intersect the projection of Q_{i-1}. It consists of two heaps of edges; one heap holds those edges whose

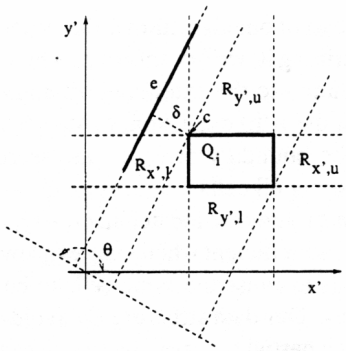

Figure 2:

projections have smaller x' coordinates than the projection of the centroid of Q_1; the other heap holds those with larger x' coordinate. Both heaps are ordered by the minimum distance of the projected edges to the projection of Q_1's centroid. The second stage of S_2, call it H_2, is analogous except that it holds all edges not in H_1 whose projections on the y' axis do not intersect the projection of Q_{i-1}. The third stage of S_2, call it H_3, is also analogous except that it holds all edges not in H_1 or H_2 whose projections onto a line perpendicular to their orientations do not intersect the projection of Q_{i-1} onto that line. For each of the two edge orientations, two heaps (as described for H_1 but w.r.t. this projection) are required. Only two edge orientations are possible since each edge of a box-decomposition is aligned with one of the two (original) coordinate axes.

In the i(th) extension, to determine those cells in S_2 intersecting Q_i, we propagate edges though S_2. First, all edges whose projections on the x' axis intersect Q_i are removed from H_1 and inserted into H_2. For the heap in H_1 containing edges whose projections have smaller x' coordinates than the projection of Q_1's centroid, removing these edges just involves repeatedly removing the minimum item until the minimum item is farther from the projection of Q_1's centroid than the projected point of Q_i with smallest x' value; the other heap is handled similarly. Second, all edges whose projections on the y' axis intersect Q_i are removed from H_2 and inserted into H_3. And finally, all edges whose projections on a line orthogonal to their orientation intersect the projection of Q_i on that line are removed from H_3.

Edges removed from S_2 had to satisfy conditions 1-3 for some queries Q_j, $j \leq i$. But, it is easy to see that if a condition is satisfied for Q_j, then it is also satisfied for Q_i. Hence, if e is removed from S_2 in the i(th) extension, then it satisfies the three conditions for Q_i and by Lemma 3.1 the edge, and hence its cell, intersects Q_i.

We have described how the intersecting cells are found, but we have not mentioned how the previously unreported points in cells C_{i-1} located in Q_i are reported. It is not difficult to see that they can be propagated though S_2 just as the edges are since a lemma similar to Lemma 3.1 but using only conditions 1 and 2 applies to points. The third stage of S_2 is then not actually necessary for points, and points can be removed from S_2 as soon as they are removed from the second stage of S_2.

To update S_2 in preparation for the $(i+1)$(th) extension, observe first that for any cell that is in both B_{i-1} and B_i, all its edges will still be in S_2 after processing the i(th) extension. For any edge removed from S_2 during the i(th) extension, we remove from S_2 all other edges associated with its cell. Therefore the edges of every cell in B_{i-1} that is not in B_i (because it intersects Q_i) will not be in S_2 after the i(th) extension. Any cell b in $B_i - B_{i-1}$ must be adjacent to a cell c in $C_i - C_{i-1}$ so it is easy to insert b's edges into S_2 when c is visited during the i(th) extension's depth first searches. Any still unreported point in a cell C_{i-1} will still be in S_2. Points in cells $C_i - C_{i-1}$ not reported in the i(th) extension are found when the cells are visited and they are added to S_2. Whenever a cell edge or an unreported point is added to S_2, the proper stage of S_2 to add it is easy to determine in constant time.

Query Q_1 is handled uniquely. Using the point location data structure for the subdivision, a cell intersecting Q_1 is determined in $O(\log N)$ time. Starting from that cell, a pruned depth first search like those described above visits all cells in C_1, reports the appropriate points contained in cells C_1, and builds S_2 appropriately.

3.1 Asymptotic Analysis

We now bound asymptotically the overall time to process a sequence of two dimensional extending rectangular queries in terms of N, E and the number of cells C intersecting query Q_E.

Theorem 3.1 *With $O(N)$ storage, an on-line sequence of E extending two dimensional rectangular range reporting queries can be processed in $O(\log N + C \log C + E)$ time, where C is the total number of subdivision cells intersecting Q_E. As a special case, a single query ($E = 1$) can be processed in $O(\log N + C)$ time.*

Proof: We first bound the total number of cell edges and points inserted into (and extracted from) S_2 over the course of all E extensions. Each cell intersecting Q_E can have its point inserted into S_2 at most one time making the total number of points inserted into S_2 $O(C)$. A cell's edges are only inserted into S_2 when the cell first becomes a surrounding cell for one of the E queries. Any cell considered a surrounding cell at some time during the E extensions must either intersect Q_E or be adjacent to a cell in Q_E, thus there are $O(C)$ surrounding cells and, since each cell's boundary can be represented by $O(1)$ edges (following from property [P2]), $O(C)$ edges are inserted into S_2. Hence, the size of each heap comprising S_2 is $O(C)$ and each heap operation takes $O(\log C)$ time.

The total amount of time spent in propagating each of the $O(C)$ edges and points through S_2 is $O(\log C)$ since it is inserted into (and removed from) at most 3 heaps. Determining which heaps in S_2 require processing takes only constant time per extension. For the searches started from cells in B_{i-1} intersecting Q_i, only a constant amount of

time is spent at each cell in $C_i - C_{i-1}$ determining if its point should be reported or added to S_2, checking its $O(1)$ neighboring cells to see if their edges should be added to S_2, and determining the adjacent cells from which to continue the search. Since $\sum_{i=1}^{E} |C_i - C_{i-1}| = C$, this takes $O(C)$ time. An additional $O(\log N)$ is required by the first query to identify a cell to begin the search. Therefore, the total time spent performing all E extensions is $O(\log N + C \log C + E)$. In the special case when $E = 1$, the first query takes only $O(\log N + C)$ time since S_2 does not have to be built.

The storage is bounded by $O(N)$ since the box-decomposition is of size $O(N)$, the point location data structure is of size $O(N)$, and S_2 consists of 8 heaps each of size $O(C)$ which is also $O(N)$. □

3.2 Query Sensitive Analysis

Our goal here is to express the number of cells C intersecting Q_E in terms of Q_E's characterizing parameters k, α, and l. This analysis is similar to the analysis of the approximate range searching algorithm in [1] which performs a partial top down search of a box-decomposition tree. Our analysis depends only on the subdivision's properties listed in Section 2.

Lemma 3.2 *The number of BD subdivision cells intersecting any d-dimensional rectangular region Q characterized by k, α and l is* $O(k + (l/\alpha)^{d-1})$.

Proof: By property [P3], there exists an association (or a mapping) between cells and data points such that a cell of size s is associated with a data point located within distance $c_1 s$ of the cell's outer box, where c_1 is a constant, and at most a constant number of cells are associated with each data point. For the following proof, we do not actually need to know the association, just that it exists.

The cells intersecting Q can be partitioned into two types: type (1) cells intersect Q and have associated data points in Q or within distance α of Q's boundary, and type (2) cells intersect Q and have associated data points located farther than α from Q's boundary.

For type (1) cells, since there are $2k$ data points located in Q or within distance α of Q's boundary and every data point is associated with at most a constant number of cells, the number of type (1) cells is $O(k)$.

For type (2) cells, we first observe that their outer boxes must be larger than $\alpha/(c_1 + \sqrt{d})$ for their cells (and thus their outer boxes) to intersect Q and their associated data points (contained within $c_1 s$ of their outer boxes) to be farther than α from Q's boundary. To count cells of type (2), we partition them into those that intersect Q's boundary and those that do not. Because they are of size at least $\alpha/(c_1 + \sqrt{d})$, the number of type (2) cells intersecting Q's boundary is at most $c_3 \lceil l(c_1 + \sqrt{d})/\alpha) \rceil^{d-1} = O((l/\alpha)^{d-1})$ by property [P4].

Now consider type (2) cells not intersecting Q's boundary. These cells, located entirely inside Q, cannot be too far from Q's boundary since their associated points are located outside Q. Specifically, for cells of size s, their outer boxes cannot be farther than $c_1 s$ from Q's boundary. Therefore, their outer boxes (and hence the cells themselves) are located completely within $c_1 s + \sqrt{d}s$ of Q's boundary. But this means all these cells are packed within a volume A of size $A \leq P(c_1 s + \sqrt{d}s)$, where P is Q's surface area. Since $P \leq 2dl^{d-1}$, we have $A \leq 2dl^{d-1}(c_1 s + \sqrt{d}s)$. Cells of size s are pairwise disjoint and each cell occupies a volume of at least s^d/c_2^d by property [P5], so at most $Ac_2^d/s^d \leq 2dl^{d-1}(c_1 s + \sqrt{d}s)c_2^d/s^d = c_4(l/s)^{d-1}$ cells can be packed into A, where $c_4 = 2dc_2^d(c_1 + \sqrt{d})$ is a constant. Therefore, $c_4(l/s)^{d-1}$ bounds the number of type (2) cells not intersecting the boundary of Q and of size s.

This bound applies specifically to cells of size s so we now sum over all possible sizes of s. Noting that $s = 1/2^i$ for some integer i, $s \geq \alpha/(c_1 + \sqrt{d})$, and s cannot be larger than 1 since the largest box, the unit hypercube, is of size 1, we have the following geometric series,

$$\sum_{i=0}^{\lfloor \log((c_1+\sqrt{d})/\alpha) \rfloor} c_4(l2^i)^{d-1} \qquad (1)$$

which is $O((l/\alpha)^{d-1})$. Therefore the total number of type (2) cells not intersecting Q's boundary is $O((l/\alpha)^{d-1})$. Combining this with the number of type (2) cells intersecting Q's boundary and the number of type (1) cells gives us our result. □

Our bound on C is obtained by substituting $d = 2$ into the preceding lemma. With any type of query, we cannot avoid the k term in the definition of C since we must report the points in Q, so ideally we would like the l/α term to be $O(k)$. For what we might think of as a worst case targeted query — when the data points inside and outside Q have the same density — we have $k \geq l/\alpha$ for reasonable values of k. To see this, for query Q, let $h_1 \leq h_2$ be the lengths of its two sides. Suppose the data points outside Q but within α of its boundary are uniformly distributed with density $k/h_1 h_2$, the same as the average density of data points in Q. In this case, we can show that for any k and l, the expected value of α is larger than $h_1/8$. Hence, $l/\alpha < 8l/h_1 \leq 8\sqrt{2}h_2/h_1$. Therefore, if $k \geq 8\sqrt{2}h_2/h_1$, then $k \geq l/\alpha$ and $C = O(k)$. Note that for a more typical targeted query where the density outside Q is much less than the average density of points inside Q, Q may contain fewer data points and still $C = O(k)$.

4 Three Dimensional Extending Queries

Processing extending rectangular queries in three dimensions is similar to that in two dimensions. The main difference is that instead of propagating cell edges through S_2, we propagate cell faces through a data structure we call S_3. As before, we assume that during a preprocessing stage an adjacency graph representation of a three dimensional, smoothed box-decomposition subdivision and

a point location data structure satisfying all the properties of Section 2 are constructed.

To process the i(th) extension, we assume inductively that having completed the $(i-1)$(th) extension we have a data structure S_3 (described below) containing all points located in cells C_{i-1} that have not yet been reported and all faces of cells B_{i-1}. Analogous to the case of two dimensions, in the i(th) extension, we extract from S_3 points it contains that are in Q_i and faces it contains whose cells intersect Q_i. Extracted points are just reported. Faces extracted are used to start pruned depth first searches of the BD adjacency graph (analogous to those in two dimensions) which (collectively) visit each cell in $C_i - C_{i-1}$ and report its point if it is located in $Q_i - Q_{i-1}$.

S_3 is organized to efficiently find the cells B_{i-1} intersecting Q_i in the i(th) extension. Since cells B_{i-1} do not intersect Q_{i-1}, if they intersect Q_i then some part of their boundary (one or more of their faces) must intersect Q_i. Therefore, we only need to determine the faces of cells B_{i-1} that intersect Q_i. Let x', y', and z' be the axes of a coordinate system aligned with the sequence of queries, let $P_{x'y'}(r)$ be the projection of any region $r \subset R^3$ onto the x', y' plane, and let $P_{x'z'}(r)$ and $P_{y'z'}(r)$ be defined similarly. Let $P_{\theta,\phi}(r)$ be r's projection onto a line with orientation determined by angles θ and ϕ. The following lemma determines the structure of S_3; we omit the proof since it is similar to Lemma 3.1.

Lemma 4.1 *A face f with normal direction (θ, ϕ) intersects rectangular query Q_i iff*

1. $P_{x'y'}(f) \cap P_{x'y'}(Q_i) \neq \emptyset$,
2. $P_{x'z'}(f) \cap P_{x'z'}(Q_i) \neq \emptyset$,
3. $P_{y'z'}(f) \cap P_{y'z'}(Q_i) \neq \emptyset$, and
4. $P_{\theta,\phi}(f) \cap P_{\theta,\phi}(Q_i) \neq \emptyset$.

S_3 is a four stage pipeline, each stage corresponding to one of the four conditions above. The first stage of S_3 holds all faces whose projections on the x', y' plane do not intersect the projection of Q_{i-1}. Since three dimensional BD subdivision cells are aligned with the three original coordinate axes, the face of any cell is a planar polygon with sides parallel to two of three orthogonal directions (in R^3). Parallel projection preserves incidence and parallelism, so the face's projection onto the x', y' plane is also a polygon with sides parallel to two directions (although not necessarily orthogonal). The sequence of E nested, three dimensional, rectangular queries projected onto the x', y' plane is just a sequence of nested, two dimensional, rectangular queries. In the i(th) extension, we will want to determine those faces contained in the first stage of S_3 that intersect the projection of Q_i so that they may be propagated to stage 2. But then we can just implement the first stage as an instance of S_2 for the x', y' plane, the projected queries, and the projected faces. (Note that the last stage of this instance of S_2 requires three sets of heaps, two for each of the three possible face's edge orientations.) The next two stages are analogous. The last stage, which is a one-dimensional problem since the projection is onto a line, consists of three sets (one for each face orientation) of two heaps analogous to those used in the stages of S_2.

In the i(th) extension, to determine those cells in S_3 intersecting Q_i, we propagate faces though S_3 similarly as was done in two dimensions. Unreported points contained in cells C_{i-1} are handled similarly but may be removed after stage 2 of S_3. S_3 is updated in preparation for the next extension when the cells in $C_i - C_{i-1}$ are visited. Query Q_1 is handled uniquely, as was done in two dimensions. The following theorem summarizes the asymptotic analysis; we omit the proof since it is similar to to Theorem 3.1.

Theorem 4.1 *With $O(N)$ storage, an on-line sequence of E extending three dimensional rectangular range reporting queries can be processed in $O(\log N + C \log C + E)$ time, where C is the total number of subdivision cells intersecting Q_E. As a special case, a single query ($E = 1$) can be processed in $O(\log N + C)$ time.*

By substituting $d = 3$ into Lemma 3.2, we obtain $C = O(k + (l/\alpha)^2)$. Generalizing the analysis at the end of Section 3, if $h_1 \leq h_2 \leq h_3$ are the lengths of Q's sides, we can show that (under conditions defined in Section 3) when $k \geq 3(8)^2 h_3^2/h_1^2$, we have $C = O(k)$.

References

[1] S. Arya and D. M. Mount. Approximate range searching. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 172–181, 1995.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994. In revised form at ftp://ftp.cs.umd.edu/pub/faculty/mount/Papers/dist.ps.gz.

[3] B. Chazelle and B. Rosenberg. Lower bounds on the complexity of simplex range reporting on a pointer machine. In *Proc. 19th Intl. Colloq. on Automata, Lang., and Prog.*, volume 623 of *Lect. Notes in Comp. Sci.*, pages 439–449. Springer-Verlag, 1992.

[4] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983.

[5] M. de Berg, M. van Kreveld, and J. Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *J. Algorithms*, 18:256–277, 1995.

[6] R. Y. Flatland. PhD dissertation. Department of Computer Science, Rensselaer Polytechnic Institute. To appear 1996.

[7] R. Y. Flatland and C. V. Stewart. Extending range queries and nearest neighbors. In *Proc. of the Seventh Canad. Conf. on Comput. Geom.*, pages 267–272, 1995.

[8] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *To appear in Intl. Journal of Comp. Geom. and App.*

[9] D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. Chromatic nearest neighbour searching: a query sensitive approach. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 261–266, 1995.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.