Intersection detection and computation of Manhattan terrains

Binhai Zhu*

School of Computer Science, McGill University, Montreal, Canada.
* Current Address: Los Alamos National Lab, MS265,Los Alamos, NM 87545, USA.

1 Introduction

Intersection detection (computation) is one of the fundamental problems in computational geometry. This problem finds applications in motion planning, collision detection and avoidance, computer graphics, CAD and VLSI [PS85]. For the general intersection detection problem regarding non-convex polyhedra in 3D, a few results are only known very recently. With the results of ray-shooting [dB92], de Berg shows that the intersection detection query of a line (a ray or a line segment) with an arbitrary polyhedron can be answered in $O(\log n)$ time and the data structure can be constructed in $O(n^{4+\epsilon})$ (for any $\epsilon > 0$) time and space. Furthermore, the intersection of a rectilinear line (a ray or a line segment) with an axis-parallel polyhedron can be detected in $O(\log n)$ time with $O(n^{1+\epsilon})$ (for any $\epsilon > 0$) time and space preprocessing. If the space complexity is of more concern, then he shows that the query can be answered in $O(\log n(\log\log n)^2)$ time and the data structure can be constructed in $O(n \log n)$ time and space. Since a Manhattan terrain is a special axis-parallel polyhedron which arises frequently in practice we would like to ask: if the object is a Manhattan terrain instead of an arbitrary axis-parallel polyhedron, can the rectilinear ray shooting be performed more efficiently? We show in this paper that this question can be answered positively.

There are much more results regarding intersection computation. Most of these results before 1988 can be found in Chapter 7 of [PS85] (the second edition). Two of the most famous results of intersection computation after 1988 are the optimal linear time algorithm for computing the intersection of two convex polyhedra by Chazelle [Cha92] and the optimal $O(n \log n + K)$ time algorithm for computing the intersection of n line segments [CE92].

The known results regarding intersection detection and computation of polyhedral terrains are as follows. The problem of computing the shortest vertical distance between two non-intersecting polyhedral terrains (two set of lines or line segments in 3D) has been solved with a randomized $O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$) time algorithm; consequently, the problem of detecting the intersection between two polyhedral terrains can be solved with the same bound [CEGS89]. The problem of computing the longest vertical distance between two polyhedral terrains (two sets of lines or line segments in 3D) has also been solved with a randomized $O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$) time algorithm [GP92]. The problem of computing the intersection (upper envelope) of two polyhedral terrains one of which is convex has been solved in optimal $O(n \log n + K)$ time [Sha88]. The problem of computing the intersection of two polyhedral terrains has been solved with a randomized $O(n^{4/3+\epsilon} + K^{1/3}n^{1+\epsilon} + K \log^2 n)$ (for any $\epsilon > 0$ and K is the size of output) time algorithm [Pel93].

In this paper we consider the intersection detection and computation problems for Manhattan terrains (solid Manhattan terrains). We show that after $O(n \log n)$ time and space preprocessing, the intersection of a rectilinear line segment (ray or line) with a Manhattan terrain can be detected in $O(\log n)$ time. For the dynamic version of this problem, we show that there exists a dynamic data structure with a query and update of $O(\log^2 n)$ time and $O(n \log n)$ space. With these results, we are able to show that:

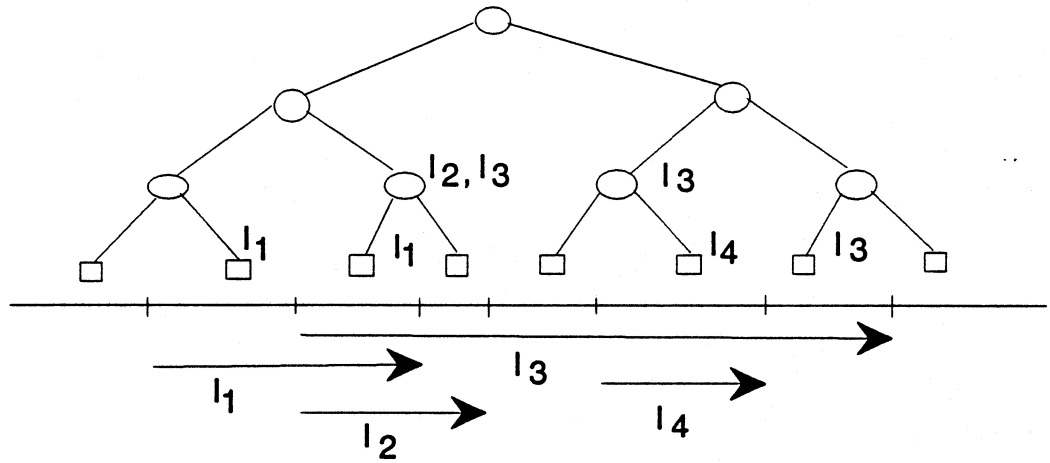

Figure 1: A segment tree of four line segments.

(1) Given two Manhattan terrains with a total of $O(n)$ vertices, we can either compute the shortest (longest) vertical distance between them or report their intersection in $O(n \log n)$ time. Equivalently, given two sets of rectilinear line segments, the shortest (longest) distance between them can be computed in $O(n \log n)$ time.

(2) Given two Manhattan terrains with a total of $O(n)$ vertices, we can compute their intersection (upper envelope) in $O(n \log n + K)$ time, where K is the combinatorial complexity of the envelope.

The techniques and data structures we use include: *multi-layer tree*, *segment tree* [Ben77, VW82], *symmetric order heap* [HT84], *fractional cascading* [CG86] and any one of the techniques and data structures supporting planar point location queries in $O(\log n)$ time with $O(n)$ time and space preprocessing [Kir83, EGS86, ST86]. In order to support the dynamic version of the above problems, we also use the 2D dynamic point location algorithm of Preparata and Tamassia [PT89].

We begin by recalling some elementary definitions. A Manhattan terrain $\mathcal{M}$ with n vertices is a connected 3-D rectilinear polyhedral surface such that the intersection of any vertical line with $\mathcal{M}$ is either empty, a point, or a vertical line segment. A solid Manhattan terrain $\mathcal{M}$ is a simple rectilinear polyhedron such that there exists a face f of $\mathcal{M}$ and the intersection of $\mathcal{M}$ with any line perpendicular to f is either empty, or a line segment with one endpoint lying on f.

A segment tree is a data structure that is used to store a set of intervals on the real line. Segment trees are introduced by Bentley in 1977 [Ben77]. Since then, they have found a lot of applications, especially for axis-parallel and c-oriented geometric objects [PS85]. Let S be a set of n possibly overlapping half-open intervals I_i's on the real line, i.e., $S = \{I_1, ..., I_n\}$. Let $I_i = [x_i, x_i')$. The $m \leq 2n$ different endpoints of I_i's partition the real line into $m + 1$ half-open *elementary intervals*. The segment tree that stores S is a balanced binary tree with $m + 1$ leaves, which correspond to the $m + 1$ elementary intervals. Each internal node v of T has an interval associated with it that is the union of the intervals associated with its two children. In other words, each node v of T has an interval associated with it that is the union of all elementary interval leaves of $T(v)$. We denote this interval by I_v. An interval $I_j \in S$ is stored at those nodes v such that $I_v \subseteq I_j$, but $I_{parent(v)} \not\subseteq I_j$. One can check that I_j is precisely the disjoint union of all the intervals I_v over all node v where I_j is stored.

It is known that I can be stored with at most two nodes at each level of T. Therefore an interval is stored at most $O(\log n)$ times in T, which implies that the space for storing T is $O(n \log n)$ [Ben77, VW82]. We denote the subset of intervals that are stored at some node v by $L(v)$, and call $L(v)$ the *associated list of*

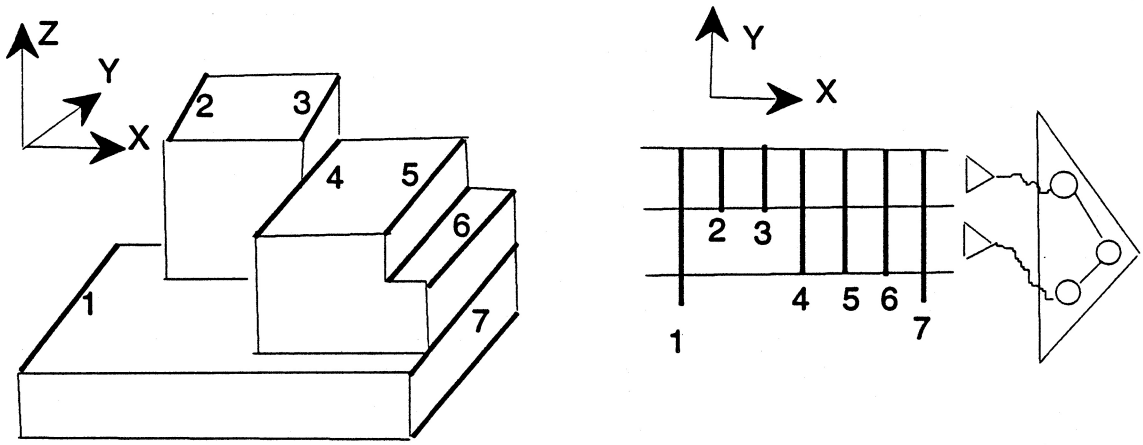

Figure 2: A Manhattan terrain and its two-layer segment tree.

v. In Figure 1, we show a segment tree storing four segments. In practice, to solve more complex problems we might need to store each associated list as some other data structure instead of a linked list. This gives us *multi-layer* data structures [WL85].

The intersection detection problem regarding Manhattan terrains is to design data structures to support efficient queries about the intersection between a preprocessed Manhattan terrain $\mathcal{M}$ and an arbitrary rectilinear line segment or another Manhattan terrain. We also consider the dynamic version of the above problems, i.e., when only a constant number of changes occurred on the vertices, edges and faces of $\mathcal{M}$. (In the worst case, adding or deleting a new face to $\mathcal{M}$ could cause a linear number of changes on vertices, edges and faces of $\mathcal{M}$.)

2 Detecting the intersection of two Manhattan terrains

In this section we show how to detect the intersection between two Manhattan terrains in $O(n \log n)$ time. We first show how to report the intersection between a Manhattan terrain $\mathcal{M}$ and a query rectilinear line segment in $O(\log n)$ time after $O(n \log n)$ time and space preprocessing. We begin with the following definition. A line segment is *rectilinear* if it is perpendicular to either the XY-, or YZ- or XZ-planes.

Definition: Given two rectilinear line segments t_1 and t_2 in 3-D, if there is a vertical line l such that $t_1 \cap l \neq \emptyset$, $t_2 \cap l \neq \emptyset$, then the vertical distance between t_1 and t_2 (denoted by $d(t_1, t_2)$) is the difference between the z-coordinate of $t_1 \cap l$ and $t_2 \cap l$. Otherwise the vertical distance between the two rectilinear segments is infinity.

We first give an $O(\log^2 n)$ solution to compute the shortest vertical distance between a rectilinear line segment $\overline{ab}$ and a Manhattan terrain $\mathcal{M}$. It is easy to see that the shortest vertical line segment $\overline{uv}$, with $v \in \overline{ab}$, $u \in \mathcal{M}$, must satisfy one of the following properties:

(1) v is either a or b;
(2) u lies on an edge of $\mathcal{M}$ and v lies on $\overline{ab}$.

The first case can be dealt with using planar point location (for this reason, we ignore the case when $\overline{ab}$ is a vertical line segment, i.e., it is vertical to the XY-plane). For the second case we just consider the case when $\overline{ab}$ is perpendicular to the YZ-plane. Without loss of generality, assume $a = (x_a, y), b = (x_b, y)$. The idea is to construct a *two-layer tree* such that the first layer is a segment tree and the second layer is a balanced binary tree. We first build a segment tree T for all of the edges of $\mathcal{M}$ which are perpendicular to the

XZ-plane (Figure 2). A node v in T has a certain y-interval I_v associated with it and v can be considered to represent the horizontal slab $[-\infty, +\infty] \times I_v$. When considering a vertical line segment inside the horizontal slab corresponding to v, we always restrict our attention to the part of the vertical line segment inside the slab corresponding to v. A search with y in T gives us $O(\log n)$ associated lists $L(v)$ such that the shortest vertical distance between $\overline{ab}$ and $\mathcal{M}$ is equal to the shortest vertical distance between $\overline{ab}$ and one of the elements in these lists.

We fix a plane $\mathcal{P}$ parallel to the XY-plane which is above $\mathcal{M}$. We store each associated list $L(v)$ in a balanced search tree T_v according to the x-coordinates of these $O(n)$ line segments such that a leaf corresponds to a line segment in $L(v)$ and each leaf store its distance to the plane $\mathcal{P}$; furthermore, the parent of two nodes is the one whose distance to $\mathcal{P}$ is smaller. With such a T_v, we can compute the shortest vertical distance between $\mathcal{P}$ and those segments between $X = x_a, X = x_b$, which is equal to the distance between $\mathcal{P}$ and the *nearest-common-ancestor* of x_a and x_b (denoted by $nca(x_a, x_b)$), in the time proportional to the height of T_v (which is $O(\log n)$). Then the shortest vertical distance between $\overline{ab}$ and these segments can be computed in an extra $O(1)$ time: it is equal to $d(\mathcal{P}, nca(x_a, x_b)) - d(\mathcal{P}, \overline{ab})$, if $\overline{ab}$ is below $\mathcal{P}$; otherwise, it is equal to $d(\mathcal{P}, nca(x_a, x_b)) + d(\mathcal{P}, \overline{ab})$. In total, we have $O(\log n)$ associated lists for an elementary interval (which are the associated lists stored at the nodes on the path from the root to the leaf representing that elementary interval). For each of these $L(v)$ (and the corresponding T_v) it takes $O(\log n)$ time to compute the shortest vertical distance between $\overline{ab}$ and $nca(x_a, x_b)$ in T_v. Therefore the complexity for answering such a query is $O(\log^2 n)$ and the preprocessing time and space is $O(n \log n)$. Consequently we have the following theorem.

Theorem 1: After $O(n \log n)$ time and space preprocessing, the shortest vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be answered in $O(\log^2 n)$ time [1].

Furthermore, the above data structure also solves the dynamic version of the problem. The point is that the second layer is a balanced binary search tree which supports an $O(\log n)$ time update [Tar83]. Combining this with the results of Preparata and Tamassia [PT89] (which supports $O(\log^2 n)$ time for a query or an update after $O(n)$ time and space preprocessing), we have:

Corollary 2: After $O(n \log n)$ time and space preprocessing, the shortest (longest) vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be solved in $O(\log^2 n)$ time. A Manhattan terrain of constant size can be inserted into or deleted from the structure representing $\mathcal{M}$ in $O(\log^2 n)$ time.

We show that for the static version of the problem, we can improve the query time to $O(\log n)$ without increasing the time and space for preprocessing. The crucial point is that instead of storing a balanced binary tree for $L(v)$, we can store a special data structure called *symmetric order heap* [HT84], such that the *nearest-common-ancestor* of two nodes in the heap can be answered in $O(1)$ time. We call the resulting data structure a two-layer *hybrid segment tree.*

Let A[1..n] be an array of n real numbers. A *Symmetric Order Heap* (*SH*) is a binary tree that holds the entries of A[1..n]. *SH* has n nodes and each A[i] appears in only one node of *SH*. If we denote the node that holds A[i] by w_i, then *SH* has the following properties:

1. *SH* is a heap, i.e., if w_i is the parent of w_j then A[i] $\leq$ A[j].
2. The symmetric order (inorder) traversal of nodes of *SH* is $w_1, w_2, ..., w_n$, i.e., nodes in symmetric order contain A[1], A[2],...,A[n].

In [HT84], it is shown that a symmetric order heap can be constructed in linear time and after an additional linear time preprocessing on the heap we can answer the nearest-common-ancestor query in $O(1)$ time.

[1] Throughout this paper, all the results regarding computing the shortest vertical distance can be generalized to computing the longest vertical distance.

Let $val(i) = d(\mathcal{P}, w_i)$ be the shortest vertical distance between w_i and $\mathcal{P}$, where $w_i \in L(v)$. We can use the $val(i)$ information for each w_i to construct a a *Symmetric Order Heap* for every associated list. Then by [HT84], we can answer the nearest-common-ancestor query between any w_k and w_j in $O(1)$ time such that w_k is the leftmost interval intersecting with (the XY-projection of) $\overline{ab}$ and w_j is the rightmost interval intersecting with (the XY-projection of) $\overline{ab}$. However, there are $O(\log n)$ associated lists for an elementary interval and for each associated list $L(v)$ finding the leftmost (rightmost) interval intersecting with $\overline{ab}$ by binary search takes $O(\log n)$ time. Consequently the total time complexity could be $O(\log^2 n)$, which is no better than the binary tree implementation. Nevertheless we can apply the *fractional cascading* technique of Chazelle and Guibas [CG86] to improve the bound to $O(\log n)$. The fractional cascading technique is essentially copying certain elements of $L(left(v))$ into $L(v)$ and vice versa [CG86]. It turns out that the copying can be done in such a way that the search in $L(left(v))$ can be done in constant time if we already know the position of the query value in $L(v)$; furthermore, the asymptotic preprocessing time and space is not affected. Consequently we perform a binary search in the associated list which is stored at the root of the tree T, and then we can do the searches in the lists stored at the remaining nodes on the search path in $O(1)$ time per list. (We can also say that the amortized time for this search is $O(1)$ per list). Therefore we have:

Theorem 3: After $O(n \log n)$ time and space preprocessing, the shortest vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be computed in $O(\log n)$ time.

With this result we can proceed to obtain the following theorem.

Theorem 4: The (all) shortest vertical distance problem between two non-intersecting Manhattan terrains with a total of $O(n)$ vertices can be computed in $O(n \log n)$ time.

It is easy to generalize the definition of the shortest vertical distance between an edge $\overline{xy}$ of $\mathcal{M}$ and $\overline{ab}$ so that $\overline{ab}$ intersects with $\mathcal{M}$ if and only if the shortest vertical distance between $\overline{ab}$ and $\mathcal{M}$ is negative. Therefore after $O(n \log n)$ time and space preprocessing, we can either report the intersection of $\overline{ab}$ (or a rectilinear line) and $\mathcal{M}$ (by giving a witness), or return the shortest vertical distance between $\overline{ab}$ and a Manhattan terrain $\mathcal{M}$, or report that the shortest vertical distance between $\overline{ab}$ and $\mathcal{M}$ is infinity in $O(\log n)$ time. Consequently given two Manhattan terrains with a total of $O(n)$ vertices we can either report their intersection by (giving a witness) or return the shortest vertical distance between them in $O(n \log n)$ time.

3 Computing the intersection of two Manhattan terrains

With the above data structure, we can actually compute the intersection of a Manhattan terrain $\mathcal{M}$ and a query line segment $\overline{ab}$ in $O(\log n + k_{ab})$ time, where k_{ab} is the combinatorial complexity of the intersection of $\mathcal{M}$ and $\overline{ab}$. With this result we can compute all the edge/face intersections between the two Manhattan terrains. Consequently, the intersection (*upper envelope*) of two Manhattan terrains can be computed in $O(n \log n + K)$ time, where K is the combinatorial complexity of the envelope.

Theorem 5: The intersection of $\mathcal{M}$ and a query line segment $\overline{ab}$ can be computed in $O(\log n + k_{ab})$ time, where k_{ab} is the combinatorial complexity of the intersection of $\overline{ab}$ and $\mathcal{M}$.

Proof: We preprocess $\mathcal{M}$ as we have done in Theorem 3, that is, the second layer is a set of *symmetric order heaps*. Once we locate the position of x_a and x_b in $L(v)$ (which is stored as a symmetric order heap, each node has additional pointers to its next level for fractional cascading), we can compute the nearest-common-ancestor of them, $nca(x_a, x_b)$, in $O(1)$ time. Then we start a preorder traversal at the subtree rooted at $nca(x_a, x_b)$. During the traversal, if a node w representing $\overline{cd}$ is traversed such that $\overline{ab}$ is above $\overline{cd}$ then discard the subtree rooted at w (since all these nodes rooted at w are below $\overline{ab}$ and there is no intersection between $\overline{ab}$ and all these nodes). Suppose there are k_i intersections between $\overline{ab}$ and the edges stored in (the symmetric order heap of) $L(v)$. Since the number of leaves in a tree is at most the number of internal nodes

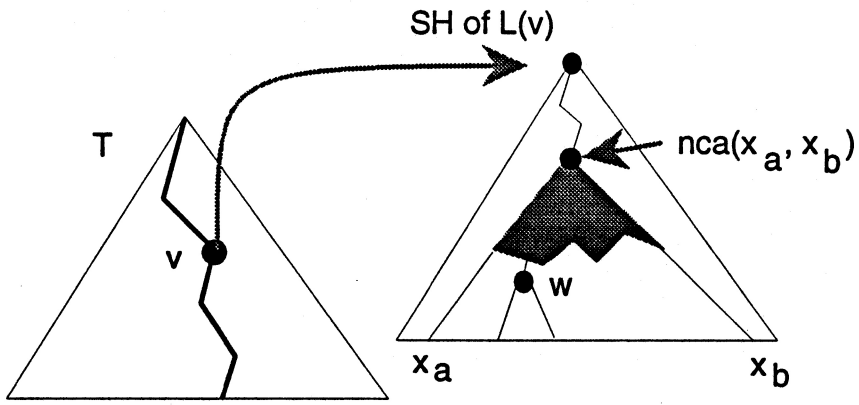

Figure 3: Computing the intersections of a line with a Manhattan terrain.

plus one, it is clear that we visit at most $2k_i+1$ nodes in $L(v)$ to compute all the intersections between $\overline{ab}$ and the edges stored in $L(v)$ (Figure 3). Consequently it takes $\sum_{1\leq i\leq \log n}(2k_i+1) = O(\log n + 2\sum_{1\leq i\leq \log n} k_i)$ $= O(\log n + k_{ab})$ time to compute all the intersections between $\overline{ab}$ and $\mathcal{M}$. □

With this result, we can solve the following problem. Given two Manhattan terrains $\mathcal{M}_1, \mathcal{M}_2$, compute the upper envelope of $\mathcal{M}_1, \mathcal{M}_2$ (i.e., viewing $\mathcal{M}_1, \mathcal{M}_2$ as two functions $z = f_1(x,y)$ and $z = f_2(x,y)$ [2], the *upper envelope* of $\mathcal{M}_1, \mathcal{M}_2$ is the graph of the pointwise maximum of f_1, f_2). Clearly each vertex of the upper envelope is either:

(1) a vertex of $\mathcal{M}_1$ lying below $\mathcal{M}_2$,
(2) a vertex of $\mathcal{M}_2$ lying below $\mathcal{M}_1$,
(3) an intersection of an edge of $\mathcal{M}_1$ with a face of $\mathcal{M}_2$, or
(4) an intersection of an edge of $\mathcal{M}_2$ with a face of $\mathcal{M}_1$.

The first two types of vertices can be found in $O(n \log n)$ time using planar point location. The last two types of vertices can be found in $O(n \log n + K)$ time by Theorem 5. After all these vertices have been computed, we have the following theorem:

Theorem 6: The upper envelope of two Manhattan terrains can be computed in $O(n \log n + K)$ time, where K is the combinatorial complexity of the upper envelope.

Similarly, given two solid Manhattan terrains such that their valid bases are on the same plane we can compute their intersection or union in $O(n \log n + K)$ time.

4 Some remarks

In this paper, we have actually shown that the rectilinear ray shooting problem for a Manhattan terrain can be solved more efficiently than that for an arbitrary axis-parallel polyhedron. However, our data structure for a Manhattan terrain does not support efficient ray shooting queries for an arbitrary ray. We list this as an open problem: is it possible to solve the intersection detection (ray shooting) problem of a Manhattan terrain with an arbitrary line in 3D in $O(\log n)$ time with $o(n^2)$ time and space preprocessing?

Chazelle et al. [CEGS89] have an $O(\log^2 n)$ time solution to solve the ray shooting problem between a polyhedral terrain and an arbitrary line with $O(n^{2+\epsilon})$ time and space preprocessing. de Berg [dB92] has

[2] For a point (x,y) on the XZ-plane or the YZ-plane, f_1 (f_2) is defined as the maximal length vertical line segment on $\mathcal{M}_1$ ($\mathcal{M}_2$) such that the lower endpoint of the line segment is (x,y).

an $O(\log n)$ time solution to solve the ray shooting problem between a 3D axis-parallel polyhedron and an arbitrary line with $O(n^{2+\epsilon})$ time and space preprocessing. de Berg's result gives us a better solution to the above problem. But can we do better? (Note that a Manhattan terrain is a special polyhedral terrain as well as a special 3D axis-parallel polyhedron.)

References

[Ben77] J. Bentley. Algorithms for Klee's rectangle problems. unpublished manuscript, Department of Computer Science, Carnegie-Mellon University, 1977.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[CEGS89] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Lines in space—combinatorics, algorithms and applications. In *Proc. 21st STOC*, pages 382–393, 1989.

[CG86] B. Chazelle and L. Guibas. Fractional cascading, Part I: A data structuring technique. *Algorithmica*, 1(3):133–162, 1986.

[Cha92] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[dB92] M. de Berg. *Efficient algorithms for ray shooting and hidden surface removal.* PhD thesis, Department of Computer Science, Utrecht University, 1992.

[EGS86] H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[GP92] L. Guibas and M. Pellegrini. New algorithmic results for lines-in-3-space problems. Technical Report TR-92-005, International Computer Science Institute, 1992.

[HT84] D. Harel and R. Tarjan. Fast algorithm for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[Kir83] D.G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[Pel93] M. Pellegrini. On lines missing polyhedral sets in 3-space. In *Proc. 9th ACM Symp. on Computational Geometry*, pages 19–28, 1993.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[PT89] F. Preparata and R. Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM J. Comput.*, 18(4):811–830, 1989.

[Sha88] M. Sharir. The shortest watchtower and related problems for polyhedral terrains. *Inform. Process. Lett.*, 29(5):265–270, 1988.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Comm. ACM*, 29:669–679, 1986.

[Tar83] R.E. Tarjan. *Data Structures and Network Algorithms.* SIAM monograph, Philadelphia, PA, 1983.

[VW82] V. Vaishnavi and D. Wood. Rectilinear line segment intersection, layered segmentation, and dynamization. *J. Algorithms.*, 3(2):160–176, 1982.

[WL85] D. Willard and G. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.