EINDHOVEN UNIVERSITY OF TECHNOLOGY

Shortest path queries in rectilinear worlds

Citation for published version (APA):
Berg, de, M. T., Kreveld, van, M. J., Nilsson, B. J., & Overmars, M. H. (1991). Shortest path queries in rectilinear worlds. (Universiteit Utrecht. UU-CS, Department of Computer Science; Vol. 9120). Utrecht: Utrecht University.

Document status and date:
Published: 01/01/1991

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Shortest Path Queries in Rectilinear Worlds

Mark de Berg   Marc van Kreveld   Bengt J. Nilsson   Mark Overmars

RUU-CS-91-20
June 1991

Shortest Path Queries in Rectilinear Worlds

Mark de Berg Marc van Kreveld Bengt J. Nilsson Mark Overmars

Technical Report RUU-CS-91-20
June 1991

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

ISSN:0924-3275

Shortest Path Queries in Rectilinear Worlds*

Mark de Berg[†] Marc van Kreveld[†] Bengt J. Nilsson[‡] Mark Overmars[†]

Abstract

In this paper, a data structure is given for two and higher dimensional shortest path queries. For a set of n axis-parallel rectangles in the plane, or boxes in d-space, and a fixed target, it is possible with this structure to find a shortest rectilinear path avoiding all rectangles or boxes from any point to this target. Alternatively, it is possible to find the length of the path. The metric considered is a generalization of the L_1-metric and the link metric, where the length of a path is its L_1-length plus some (fixed) constant times the number of turns on the path. The data structure has size $O((n \log n)^{d-1})$, and a query takes $O(\log^{d-1} n)$ time (plus the output size if the path must be reported). As a byproduct, a relatively simple solution to the single shot problem is obtained; the shortest path between two given points can be computed in time $O(n^d \log n)$ for $d \geq 3$, and in time $O(n^2)$ in the plane.

1 Introduction

The computation of shortest paths in some geometric space is a topic that has received considerable attention. This is mainly due to the number of applications, such as in robotics, VLSI-design, and Geographical Information Systems (GIS). Shortest path problems have been studied in many different settings. In (almost) all settings, there is a finite set of obstacles, which have to be avoided by any legal path. Instances of shortest path problems include different restrictions on the obstacles, the use of different metrics, the dimension of the space, preprocessing version or single query, and more.

Most work has been done in planar settings, although recently there are some 3-dimensional results. We briefly review some of the more important results that relate to

*Work of the first and fourth author was supported by the Dutch Organisation for Scientific Research (N.W.O.). Work of the first, second and fourth author was supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM). Work of the third author was supported by the Deutsche Forschungs Gemeinschaft under Grant No. Ot 64/5-4.

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

‡Institut für Informatik, Albert-Ludwigs-Universität, Rheinstr. 10–12, D-7800 Freiburg i. Br., Fed. Rep. of Germany, and Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.

the work of this paper. (The time taken to report paths is omitted from the query time bounds; this is always linear in the number of segments of the path.)

Shamos[25] considers the problem of finding the shortest Euclidean path between two points inside a simple polygon with n vertices. This problem is also studied by Chazelle[3] and by Lee and Preparata[17]. They solve it in $O(n \log n)$ time. With the linear time triangulation algorithm of Chazelle[4], this improves to $O(n)$ time. For n parallel line segments, Lee and Preparata[17] obtain an $O(n \log n)$ time algorithm. Lee[16] and Sharir and Schorr[26] consider an environment of arbitrary polygonal obstacles, and they present $O(n^2 \log n)$ time solutions. For k disjoint polygons with n vertices in total, Reif and Storer[24] give an improved $O(nk+n \log n)$ time algorithm. De Rezende, Lee and Wu[10] start out with an environment of non-intersecting rectangles and a fixed target. They preprocess in $O(n \log n)$ time to a linear space data structure, such that rectilinear (i.e., axis-parallel) shortest paths can be retrieved in $O(\log n)$ time. Clarkson, Kapoor and Vaidya[7] study rectilinear paths among non-intersecting polygonal obstacles. Their result is improved by Mitchell[20], who gives an $O(n \log n)$ time algorithm. Furthermore, he presents a data structure of linear size, in which shortest path queries to a fixed target take $O(\log n)$ time.

In the link metric, the cost of a path is the number of turns the path makes, or alternatively, the number of segments on the path. The shortest link path is thus the path with the least number of turns. For this metric there are also a number of results. Lenhart et al.[18], Suri[27] and Ke[13] study several shortest link path problems inside simple polygons. Shortest link paths with respect to non-intersecting polygonal obstacles are considered by Mitchell, Rote and Wöginger[22]. They give an $O(n^2\alpha(n)\log^2 n)$ time algorithm. De Berg[8] studies rectilinear shortest link paths inside rectilinear polygons. He devises a data structure of size $O(n \log n)$, such that a rectilinear shortest link path between any two points can be found in $O(\log n)$ time.

In 3-dimensional space, shortest path problems are considerably harder. The general problem — shortest paths in the Euclidean metric among polyhedral obstacles — is known to be NP-hard[2]. The NP-hardness result holds for any L_p-metric[1], where p is a positive integer. Clarkson, Kapoor and Vaidya[7] give an algorithm for shortest rectilinear paths amidst non-intersecting rectilinear obstacles which runs in $O(n^2 \log^3 n)$ time. The study of shortest paths on the surface of a polyhedron (convex or not) has received considerable attention[6, 21, 23, 26]. Best results so far are of Chen and Han[6], who compute the shortest path between two points in $O(n^2)$ time.

In this paper we solve the following shortest path problem in arbitrary dimensional space. Given a set of n possibly intersecting axis-parallel boxes (hyperrectangles) in d-space ($d \geq 2$), a fixed target (point), and a non-negative real C, build a data structure such that for any point in d-space, a shortest path to the target can be found efficiently. The path must avoid all the boxes, and should be shortest in the *combined metric*. In this metric, the length of a path is the sum of its L_1-length, and C times the number of turns in the path. Notice that if $C = 0$, then the data structure answers shortest path queries in the L_1-metric, and for C large enough, the data structure answers shortest path queries in the link metric (i.e., paths with a minimum number of links are found). See Figure 1 for examples of shortest rectilinear paths in the L_1-metric, the link metric, and the combined metric (for some value of C). We obtain an $O((n \log n)^{d-1})$ size structure, in which queries take $O(\log^{d-1} n)$ time (in d-space). Additionally, it takes $O(k)$ time to report a shortest path with k segments. Preprocessing takes $O(n^d \log n)$ time.

An immediate consequence of our work is a relatively simple algorithm for shortest rectilinear paths (in the combined metric) between two given points in a scene with n boxes in d-space. It runs in $O(n^d \log n)$ time. In the plane, this improves to $O(n^2)$ time. This result is interesting in its own right. It implies that the shortest path problem in 'rectilinear worlds' of higher dimension is solvable in polynomial time, whereas the shortest path problem in 'Euclidean worlds' is NP-hard. As remarked before, this is also true for the L_1-metric. What saves us is having rectilinear obstacles and paths.

Let us motivate the combined metric from a practical point of view. If we plan the motion of a mobile robot, we do not want it to make many turns, because making a turn involves slowing down, turning, and gaining speed again. Therefore the L_1-metric may be insufficient to plan good paths for robots. However, one also doesn't want the path with the least number of turns, because this path could make a large detour. Thus it is natural to assign each turn a fixed cost (the parameter C in the combined metric) and to use a combination of the L_1-metric and link metric.

For several reasons, it is difficult to compare our work with previous work. First of all, the combined metric has not been used before. Secondly, we study the preprocessing version, whereas most authors study single shot shortest path problems (generally, the preprocessing versions are at least as difficult as the corresponding single shot problems). Thirdly, this paper gives the first results for shortest paths in dimensions greater than three (and on link distance in dimensions greater than two).

In Section 2, a number of definitions are given, and lemmas proved, upon which our work is based. A set of points, called *induced points*, is defined (as in Clarkson et al.[7]). They play a crucial role, as it is proved that any shortest path to the target will repeatedly visit induced points. Therefore, it is natural to split a shortest path query in two phases. In the first, we compute to which induced point the shortest path from the query point should go first. The second phase consists of going from one induced point to the next, eventually reaching the target.

Section 3 studies the second phase; it solves the problem of finding the shortest path from any induced point to the target. The data structure obtained is basically a shortest path tree on the induced points, such that any node has an outgoing edge to that node that will lead to a shortest path to the target. (This approach is called 'continuous Dijkstra' in [21].) This will also lead to a solution to the single shot problem.

In Section 4, we consider how to find an induced point to which the query point should go first. This is needed for the query problem, contrary to the single shot problem. In higher dimensions, this results in a rather complicated recursive tree structure, in which the query proceeds by repeatedly searching in grids, ray shooting, and continuing in recursively defined structures. This leads to the main result of this paper.

The conclusions and directions for further research are given in Section 5.

2 Geometric preliminaries

Let S be a set of n d-boxes in $\mathbb{R}^d$, which are the obstacles of the problem. An i-face is defined to be an i-dimensional face of a d-box. Let t be the target point, and let C be a non-negative real which represents the cost of making one turn.

Definition 1 *A directed polygonal chain $\vec{s_1}\vec{s_2}\ldots\vec{s_m}$ from a point p to a point q is called a* rectilinear path *from p to q if and only if each link $\vec{s_i}$ of the path is axis-parallel. A rectilinear path from p to q is called* legal *if and only if it does not intersect the interior of any d-box in S. A legal rectilinear path from p to q is called a* simple step *from p to q if and only if it consists of at most d links, no two of which are parallel. p is said to* see *q if and only if there is a simple step from p to q.*

In the following we only consider legal rectilinear paths, so we omit the words 'legal' and 'rectilinear'.

Definition 2 *The* length *of a path $\Pi = \vec{s_1}\ldots\vec{s_m}$ from a point p to q in the combined metric is defined as $C \cdot (m-1) + \sum_{i=1}^{m} |\vec{s_i}|$, where $|\vec{s_i}|$ denotes the length of $\vec{s_i}$. The* distance *from p to q is the minimum length over all paths from p to q.*

Observe that the shortest path from one point to another is not necessarily unique. To see this, consider an obstacle-free environment in the plane. A shortest path is traced along either boundary segment pair of the rectangle defined by the two points. In fact, for any value of C and in any dimension $d \geq 2$, there can be an exponential number of 'different' shortest paths (exponential in n). Also notice that for C large enough, a shortest path in the combined metric will be a shortest path in the link metric, and if $C = 0$, then it is a shortest path in the L_1-metric. See Figure 1 for an illustration of shortest paths in the combined metric.

Definition 3 *The* grid *G_S for a set S of d-boxes is the arrangement of all axis-parallel hyperplanes that contain a $(d-1)$-face of a d-box in S, together with the axis-parallel hyperplanes that contain the target t. The* skeleton *of the grid G_S is the union of the closure of all 1-faces of G_S. The set V_S of* induced points *of S is that subset of the vertices of G_S that lie on the closure of a $(d-2)$-face of some d-box in S (including the target t).*

The grid G_S defines a subdivision of $\mathbb{R}^d$ into cells (d-faces). Note that any cell either lies completely inside a d-box, or completely outside of it.

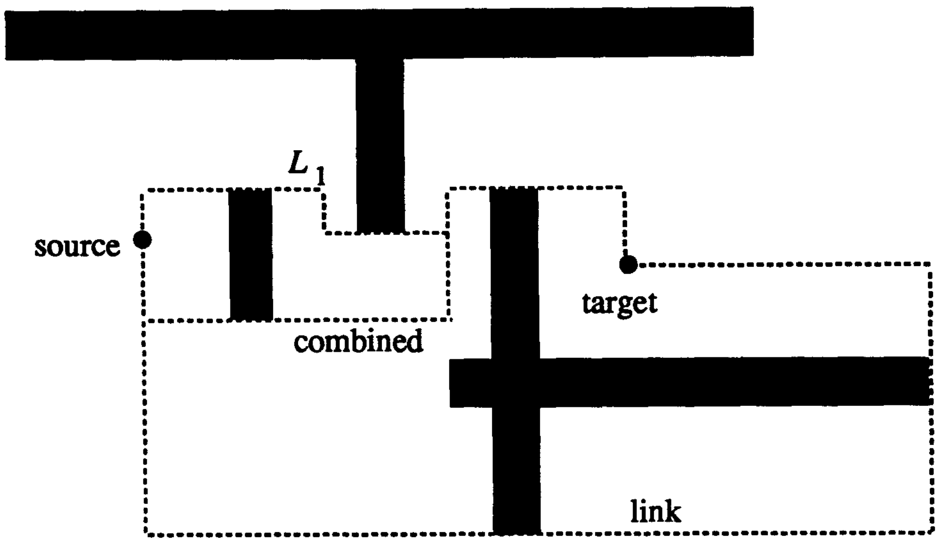


Figure 1: Shortest paths in the combined metric.

In 2-space, the set of induced points is the set of vertices of the rectangles, and in 3-space, all induced points lie on the edges of the boxes (at most $2n+1$ induced points per edge). It is easily seen that the grid G_S contains $O(n^d)$ vertices, whereas the set V_S of induced points has size $O(n^{d-1})$.

It is possible to alter a path by a transformation called *sliding*. Let $\Pi = \vec{s_1} \dots \vec{s_m}$ be a path, and let $\vec{s_i}$ and $\vec{s_j}$ $(i < j)$ be two parallel links of Π. Then, by changing the lengths of $\vec{s_i}$ and $\vec{s_j}$ with equal amounts, the subpath $\vec{s_{i+1}} \dots \vec{s_{j-1}}$ can be moved into direction $+\vec{s_i}$ or $-\vec{s_i}$ while remaining axis-parallel. Notice that sliding the path into direction $-\vec{s_i}$ cannot make the path Π longer.

With this transformation of a path, it immediately follows that for any path Π that has its endpoints on the grid G_S, there exists a new path between these endpoints which is no longer than Π, and only uses the skeleton of the grid G_S. But we can prove something stronger, which justifies the use of induced points, and which is crucial for the solution and its efficiency.

Lemma 1 *For any two induced points p and q, there exists a shortest path Π from p to q, such that any subpath of $d-1$ consecutive links of Π contains at least one induced point of V_S (d is the dimension).*

Proof: Let Π' be a shortest path from p to q with a minimum number of links. We will transform Π' into Π without increasing the length.

First, transform Π' by sliding such that all the links are contained in the skeleton of the grid. This is always possible without increasing the length of the path, or intersecting any (interiors of) d-boxes. Let $\Pi'' = \vec{s_1} \dots \vec{s_m}$ be the resulting path. If $m \leq d$, then Π''

satisfies the property of the lemma (because p and q are induced points) and we are done. Otherwise, let $\vec{s_i}$ and $\vec{s_j}$ be two parallel links in Π, such that $i < j$ and j is minimum. Such parallel links must exist when $m > d$. We slide the subpath $\vec{s_{i+1}} \dots \vec{s_{j-1}}$ into direction $-\vec{s_i}$, until some link of this subpath hits a d-box. This must happen before $\vec{s_i}$ or $\vec{s_j}$ has length zero, since that would contradict the choice of Π'. Furthermore, when the subpath hits a d-box, all links will again be contained in the skeleton of the grid.

Let $\vec{s_k}$ be the first link (i.e., the one with minimal k) of the subpath that hits a d-box. Then it not only hits a $(d-1)$-face of the d-box, but also the boundary of the $(d-1)$-face, being a $(d-2)$-face f. This is true because the links $\vec{s_i} \dots \vec{s_{k-1}}$ do not intersect the $(d-1)$-face that $\vec{s_k}$ hits. As $\vec{s_k}$ lies in the skeleton of the grid, there must be a point on the face f that is intersected by this line (and by $\vec{s_k}$). By definition, this point is an induced point in V_S.

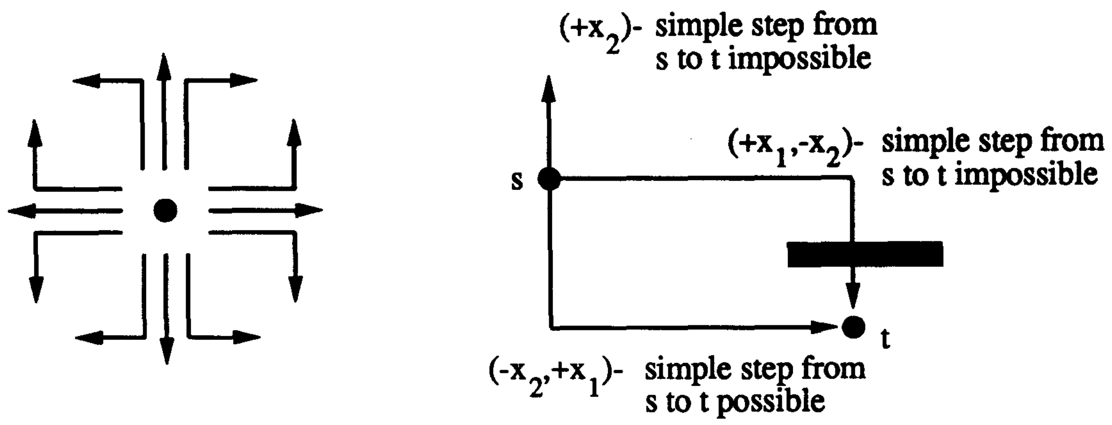
Now $\vec{s_1}$ and $\vec{s_k}$ both contain an induced point, and by the choice of i, j and k, we have $1 < k < j \leq d+1$. If necessary, we repeat the above construction with the path $\vec{s_k} \dots \vec{s_m}$.
□

The above lemma will be used in the following way. For every induced point, we compute and store — as a part of the preprocessing — the next induced point on a shortest path to the target. This gives a tree rooted at the target on the $O(n^{d-1})$ induced points, where every node (excluding the target) has one outgoing edge to its parent in the tree. Every path will (eventually) lead to the target. So from any induced point, we can follow a shortest path from the corresponding node in the graph to the target. Furthermore, at every node we store the distance to the target.

It is natural to split a query with an arbitrary point in two parts. In the first, we compute the best induced point to which we can go with a simple step. (With *best point* we mean the one that results in a shortest path.) The second part consists of following the path from the induced point that we just found, to the target (root) in the shortest path tree.

To decide upon which induced point to go to in the first simple step, it is comfortable to treat all possibilities of simple steps separately. The possibilities concern the directions of the links and their order. We make this more precise. A *direction vector* is a permutation of a non-empty subset of the base vectors of $\mathbb{R}^d$, which are prefixed by either $+$ or $-$. Thus every simple step corresponds to a unique direction vector. For example, we can speak of a $(+x_2, -x_4, -x_1)$-simple step. When we only consider simple steps with one specific direction vector, we say that the direction vector is *fixed*. When we fix the direction vector, the term 'see' only refers to simple steps with the fixed direction vector. If the direction vector is fixed, then a simple step from one point to another is either unique or impossible. See Figure 2 for an illustration of this notion.

Lemma 2 *Let the direction vector be fixed, and let p_1 and p_2 be two induced points. If a point q in $\mathbb{R}^d$ can take a simple step to both p_1 and p_2 (q sees both p_1 and p_2), and the path via p_1 has shorter total length, then for all points that can see both p_1 and p_2, the path via p_1 is shorter.*



(+x2)- simple step from s to t impossible
(+x1,-x2)- simple step from s to t impossible
s
t
(-x2,+x1)- simple step from s to t possible

Figure 2: Twelve different simple steps and direction vectors in the plane.

Proof: Because the direction vector is fixed, all points q that can see p_1 and go via p_1, have the same number of links to the target. Let cl_1 be the cost of the turns on such a path. Let $|qp_1|$ be the L_1-distance of q to p_1, and $|p_1t|$ the L_1-distance of p_1 to the target. Let cl_2, $|qp_2|$ and $|p_2t|$ be defined analogously for p_2. From the assumption in the lemma, $|qp_1| + |p_1t| + cl_1 < |qp_2| + |p_2t| + cl_2$. For any point q' that can see p_1 and p_2, the distance to the target via p_1 (resp. p_2) is $|q'p_1| + |p_1t| + cl_1$ (resp. $|q'p_2| + |p_2t| + cl_2$). Because $|qp_1| - |q'p_1| = |qp_2| - |q'p_2|$, it follows that the path for q' to the target via p_1 is shorter. □

An induced point that can be seen, and which lies on a shortest path to the target (such as p_1 in the lemma), is called the *best* induced point (among some set of points that can be seen).

We conclude this section by discussing shortest paths and their length in the combined metric. A problem we have to face in the link metric and, hence, in the combined metric is that the distance function is not additive in the following sense. The distance from p to r via q is not necessarily equal to the sum of the distances from p to q and from q to r. The problem is that in the latter case, the cost of a possible turn at point q is not counted. This is shown in Figure 3. However, if the rectangle R in Figure 3 was not present, then the shortest path from p to r makes no turn at q. Even worse, it may be the case that a shortest path that arrives at some point q horizontally, continues in a different way than a shortest path that arrives at the point q vertically (see Figure 3), and the induced point to visit after q is not unique.

We overcome these problems in the following way. We d-plicate every induced point q in V_S, to get d versions $q_1, \ldots, q_d$. Every one of the d possible ways of arriving at q corresponds to one version. When a path enters q with a link parallel to the i^{th} coordinate axis, then it arrives at the version q_i. There are two ways to leave q_i. Either use a link parallel to the i^{th} coordinate axis (simply continue in the same direction) and go to the

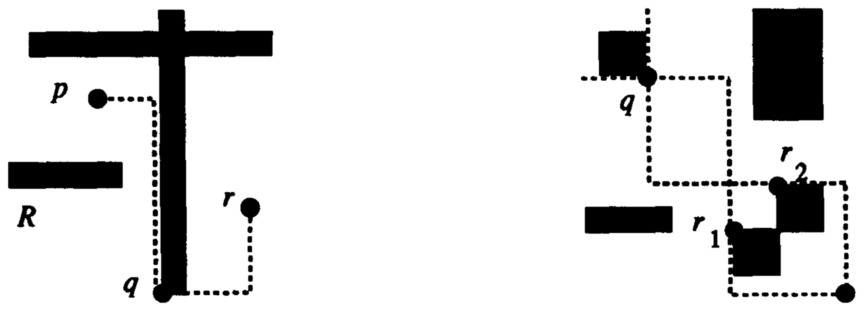


Figure 3: Difficulties in the combined metric.

version q'_i of some $q' \neq q$, or go to another version q_j of q (which corresponds to making a turn at q). The distance from one version of q to another version of q is defined to be C, the cost of making a turn. With this extension the distance function becomes additive. It is easy to see that the number of induced points remains $O(n^{d-1})$, and that any shortest path visits at most two versions of one point. Since we never want to make a turn at the target, only one version of t is needed.

3 The shortest path graph

In this section we concentrate on the structure that allows us to trace the shortest path to the target from any induced point. This structure is a single target shortest path graph, which we call sp-tree for short.

Let the set S of n d-boxes be given, and also the target t and the non-negative real C. The d-boxes and the target together define a grid G_S with $O(n^d)$ vertices, and a set V_S of $O(n^{d-1})$ induced points. Recall that we actually use d versions of each induced point. The sp-tree we aim to construct has one node for every version of each induced point. With this node the distance to the target is stored. Furthermore, a node corresponding to (a version of) some point p has one outgoing edge to a node corresponding to (a version of) a point q, where q is the next induced point on a shortest path from p to the target. There is a simple step from p to q, and the links of this simple step are stored explicitly with the edge in the graph. See Figure 4 for a planar scene with the corresponding sp-tree. We begin with the construction of the sp-tree in the planar case; then we address the higher dimensional case.

For a set S of n rectangles (or line segments) in the plane, the $O(n)$ vertices of S are the induced points. We perform two plane sweeps — one vertically and one horizontally — to compute the full simple step visibility graph on the versions of the induced points. The graph can be constructed in $O(n \log n + |E|)$ time, where $|E|$ is the number of edges, which is $O(n^2)$. Every edge is assigned a weight, which is the L_1-distance between the

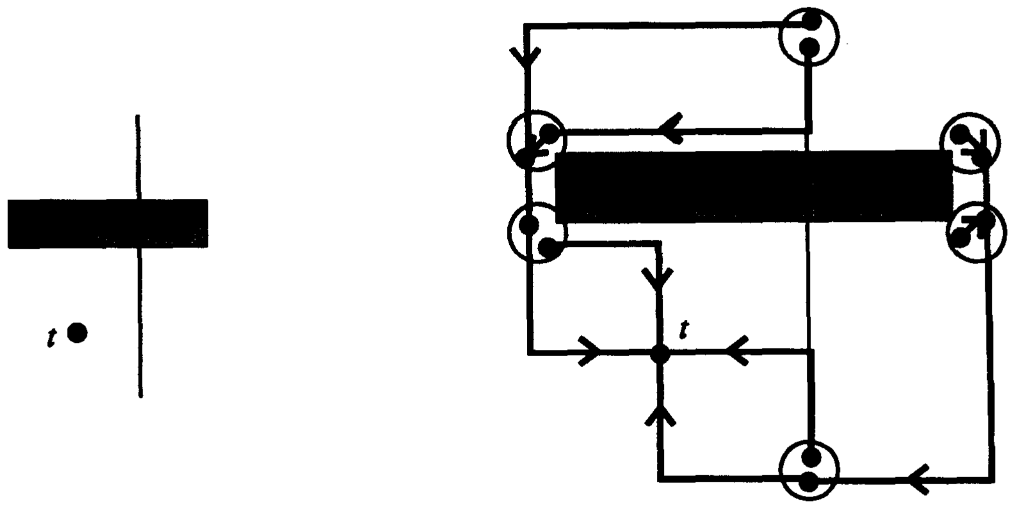


Figure 4: A planar scene with a box, a segment, and a target. Right, a possible sp-tree for the 13 versions of the 7 induced points of the scene.

corresponding points, with the constant C added if the simple step makes a turn. We identify the target in this graph, and perform Dijkstra's algorithm [11] to obtain the sp-tree. The construction takes $O(n^2)$ time. An alternative implementation of Dijkstra's algorithm gives time $((n + |E|)\log n)$, which is better when $|E|$ is small.

In the higher dimensional case, we do not want to compute the full simple step visibility graph on the induced points and then compute the sp-tree, since there can be $\Theta(n^{2d-2})$ edges in the full graph. Instead, we take the grid G_S, and show how to transform it into the sp-tree in $O(n^d \log n)$ time. Consider the grid G_S to be a weighted graph on the $O(n^d)$ vertices of the grid, where we d–plicate the grid vertices in the same way as we did with the induced points. For every version of a vertex of the grid, there is a node in the graph (from now on, we will no longer make a distinction between a node in the graph and the version of the point it represents). Two nodes p_i and q_j in the graph are neighbours if and only if either $p = q$ and $i \neq j$, or $i = j$, p and q are neighbours in the grid G_S and connected by an edge parallel to the i^{th} coordinate axis. The graph is weighted in the usual way: every edge is weighted with the distance between the versions of the points. Notice that every node in the graph has degree at most $d + 1$. The algorithm to change this graph into the sp-tree has three phases.

The first phase serves to delete all nodes that lie inside a d-box. Consider the $O(n^{d-1})$ vertical lines of the skeleton of G_S, that is, the grid lines parallel to the x_d–axis. For each line we determine all relevant d-boxes (the ones that properly intersect the line). When we have for each line a sorted list of the x_d–coordinates of the relevant boxes, it is straightforward to remove the vertices lying inside a box in linear time per line. Deletion of one node from the graph can be done in constant time, because every node has constant degree. The sorted lists of x_d–coordinates of the relevant boxes can be obtained in $O(n \log n + n^d)$ time in total, by first presorting the x_d–coordinates and then testing each box with each line. Hence, the first phase takes $O(n^d)$ time.

The second phase consists of the computation of the directed sp-tree on the (remaining) grid vertices. Note that the removal of the part of the grid lying inside d-boxes may have caused the graph to become disconnected. Trivially, we only have to consider the connected component that contains the target. Computing the directed sp-tree can be done using Dijkstra's algorithm in $O(|E| \log n) = O(n^d \log n)$ time.

In the third phase all grid vertices that are not induced points are removed. This is done in the following straightforward way. Take an induced point p, trace its shortest path until the next induced point q is reached. Delete the outgoing edge of p, and insert an edge from p to q. After doing this for every induced point, all grid vertices that are not induced points are removed from the graph. The resulting graph is the sp-tree on the induced points. The third phase takes $O(n^d)$ time, because the path traced from any induced point to the next, contains at most a linear number of grid vertices.

We conclude with the main result of this section.

Theorem 1 *Given a set S of n d-boxes, a target and a non-negative real C, the shortest path tree for the combined metric can be constructed in $O(n^d \log n)$ time and $O(n^d)$ space. The graph has size $O(n^{d-1})$, and allows for retrieval of the distance to the target in constant time for any induced point. A path to the target can be reported in $O(k)$ time, where k is the number of links on the path. For $d = 2$, $O(n^2)$ time and space suffices to construct the tree.*

The method to compute the sp-tree can also be used to solve the single shot problem. To this end, we add the axis-parallel hyperplanes containing the source point to the grid. After the first phase of the computation of the sp-tree, we have to compute a shortest path between two nodes in a graph with $O(n^d)$ edges, which takes $O(n^d \log n)$ time.

Corollary 1 *Given a set of n d-boxes and two points, a shortest rectilinear path between them that does not cross any d-box can be computed in $O(n^d \log n)$ time. This result holds in the L_1-metric, the link metric and the combined metric.*

4 Finding the first simple step

In the previous section we solved the problem of finding the shortest path to the target from any induced point. However, a query point can be any point in d-space. In this section we consider how to find the first induced point on a shortest path to the target from any point q. We will first consider the case in which q is a vertex of the grid. At the end of this section we will generalize the solution to arbitrary query points. It follows from the proof of Lemma 1 that from any grid vertex, there is a shortest path to the target, which starts with a simple step to an induced point.

We take the following approach. Fix the direction vector of the simple step and solve the problem using a recursive tree structure called a d-dimensional slab tree. This is done for every direction vector, thus a constant number of answers is found. An answer consists of some induced point p that is the best if the path must start with a simple step with the specified direction vector. We can also compute the length of this path in constant time, by adding the cost of the first simple step from the query point q to the point p, to the distance from p to the target (which we have precomputed and stored in the sp-tree). Of all the direction vectors, the one resulting in a shortest path to the target is the actual shortest path.

Observe that if the locus approach is used (i.e., d-space is divided in regions, in which the best induced point to go to is constant), every cell of the grid G_S belongs completely to one region. Thus the subdivision has complexity $O(n^d)$. Unfortunately, this bound is tight, even if we fix the direction vector. In other words, there exist scenes for which this subdivision has size $\Omega(n^d)$. In this section we show how to reduce the space requirements by storing the subdivision implicitly. To facilitate the description, we assign to every induced point a unique colour (label). We wish to colour d-space with these $O(n^{d-1})$ colours. The colour *white* is used as a special colour, indicating that no induced point can be reached at all (with this direction vector). An explicit colouring may have size $\Omega(n^d)$ as already noted; below we show that an implicit colouring with a recursive data structure uses $O((n \log n)^{d-1})$ space. This structure is called a d-dimensional slab tree.

Before we start the description of the slab tree, we first study the problem of axis-parallel ray shooting in the set S of boxes. This will prove to be necessary for the query algorithm in the slab tree. Next, because the slab tree is quite complicated in higher dimensions, we continue with the description of the planar slab tree, and then extend to three and higher dimensions.

4.1 Ray shooting in boxes

We study the following problem: Given a set S of n boxes in d-space, preprocess it such that for any axis-parallel query ray, the first intersection point with a box can be found efficiently. In the query algorithms for the slab tree, to be given later, we will repeatedly need a quick answer to such a query, to determine how far we can go with the next link of the simple step without crossing any box.

Assume without loss of generality that the query ray has direction $-\vec{x_1}$; for the other possibilities we define separate structures of the same type. First, consider the planar case. Here we have a set of rectangles and a query ray, directed to the left. Such a query ray can only hit a vertical edge of a rectangle. Therefore, we solve the problem of horizontal ray shooting in a set of vertical segments. We use the locus approach, that is, the plane is partitioned in maximal regions in which the answer is fixed. This partitioning has linear size. It is preprocessed for efficient point location [12, 14], resulting in a structure of linear size, which answers queries in $O(\log n)$ time.

We wish to solve the higher dimensional version of this problem with $O(\log n)$ query time as well. Therefore, we extend the solution in the following way. If $d > 2$, then we construct a balanced binary tree on the different x_d-coordinates of the faces of the boxes. With each leaf, we have a $(d-1)$-dimensional problem of the same type, on the set of faces whose x_d–interval contains the x_d–coordinate corresponding to the leave. This $(d-1)$-dimensional problem is solved with a recursively defined structure. If $d = 2$, we use the solution for the planar case. To query with a point $q = (q_1, \ldots, q_d)$, we search in the first $d-2$ layers of trees with the values $q_d, \ldots, q_3$. At the leaf where the search ends, we have a 2-dimensional ray shooting structure on the set of faces whose x_3- up to x_d-intervals contain the x_3- up to x_d-coordinates of the query point. In this structure we perform a point location query with the point (q_1, q_2). (This solution is not very efficient in terms storage requirements, but it is good enough for our purposes, and it achieves the fast query time that we need.)

Lemma 3 *A set S of n boxes in d-space can be preprocessed into a structure of size $O(n^{d-1})$, such that for any axis-parallel query ray, the first box hit by the ray can be found in $O(\log n)$ time. Preprocessing takes $O(n^{d-1} \log n)$ time.*

4.2 The planar slab tree

The query object for the slab tree is an interval, which is obtained as follows. Let us assume that the direction vector is fixed to be $(-x_2, -x_1)$. (We use similar structures for the other possible direction vectors.) A query with a point q begins by ray shooting in $-\vec{x_2}$-direction. This gives us the rectangle that is hit first, and thus a vertical interval I_q which is the maximal length of the first link. We will use the interval I_q to query in the slab tree. All induced points that can be reached with the specified simple step will have their x_2-coordinate in the interval.

As we remarked before, the slab tree is an implicit colouring of the plane. The idea of the implicit colouring is as follows. Consider the intersection of a horizontal line with the (coloured) subdivision. The colouring of this line tells us were to go when the query point lies on that line. But by Lemma 2, the colouring of this line gives us even more information: it tells us what the best induced point below the line is for the points that lie above it (provided that they can see it). In other words, if the interval I_q of a query point intersects this line in a red region, then the best induced point that lies below the line is the red induced point. This red induced point is visible from the query point, because the interval I_q intersects the horizontal line. Of course, the best induced point need not lie below this line. Therefore we do the following. We divide the plane in horizontal *slabs* and we colour the top boundaries of these slabs with the colours of the induced points in that slab. (The fact that we only use colours of induced points in the slab is crucial for the reduction in storage.) Then, for a query interval, we select a number of slabs that together cover the interval. The top boundaries of these slabs are all intersected by the interval and thus give us a number of colours. Of the corresponding induced points, we select the best one. We next give a more precise description of the slab tree.

Draw horizontal lines through all induced points. This results in a number of *elementary slabs.* Associate the elementary slabs from bottom to top with the leaves of a balanced binary tree. Every internal node δ corresponds to a slab which is the union of the elementary slabs that are associated with the leaves of the subtree rooted at δ. We will colour the top bounding line of the slab of δ with the colours of induced points that lie in the slab. (Points lying on the boundary between two slabs belong to the bottommost of these two slabs.) This results in a list ordered on x_1-coordinate, such that for any query interval that crosses the slab completely, the best induced point in this slab can be found by searching in this list with the x_1-coordinate of the vertical interval. This list is called the *associated list* of δ. Although one colour can appear more than once, the size of an associated list is linear in the number of induced points in the corresponding slab. This can be seen as follows. From Lemma 2 it follows that the colouring of the line can only change at a point corresponding to the x_1-coordinate of an induced point inside the slab or at a point where a vertical obstacle that crosses the slab completely intersects the line. In the latter case, there is no induced point in the slab that can be reached, to the right of the vertical obstacle. So the colour will be white. Because adjacent white regions are merged into one region, there can only be a linear number of x_1-coordinates at which there is a change in colour. Thus the size of the associated list at some node δ is linear in the number of induced points in the slab. Hence, the slab tree uses $O(n)$ space at every level of the tree which sums up to $O(n \log n)$ space in total. In Figure 5, an example of a slab tree is given.

A query in the slab tree is performed in the following way. The query object is a vertical interval that defines the region in which the induced point via which we have to go to the target must be contained.

Move with the interval down the slab tree to the node δ where the paths to its lower endpoint and upper endpoint split. From the left son of δ follow the path to the upper endpoint of the interval; for every node where we go left, search in the associated structure of its right son to find a colour. Similarly, from the right son of δ, follow the path to the lower endpoint, and search in the left son of each node were we turn right. Finally, search for a colour in the two leaves where the paths end. Searching in the associated structures is done with the x_1-coordinate of the interval. In this way we search in the associated structures of $O(\log n)$ nodes whose slabs together cover the vertical query interval. Thus the query time is $O(\log^2 n)$, which can be reduced to $O(\log n)$ by applying fractional cascading ([5]) to the slab tree. (It is straightforward to apply this technique, because all associated lists are ordered on x_1-coordinate, and when we search, we always search with the same x_1-coordinate in the associated structures.) Of the $O(\log n)$ colours found, choose the one that gives a shortest path to the target. This colour represents the induced point which yields a shortest path to the target if the path must start with a $(-x_2, -x_1)$-step. It can easily be decided which induced point results in the shortest path, because for each induced point we have the distance to the target, and we can calculate the distance from the source to any induced point by adding C to the L_1-distance. (C has to be added because a $(-x_2, -x_1)$-simple step makes one turn.)

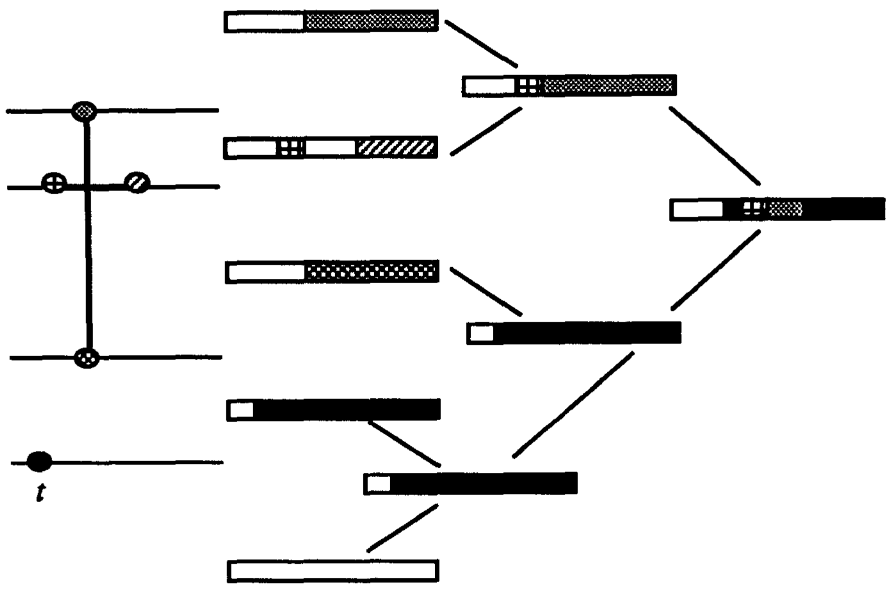


Figure 5: An example of a slab tree for the $(-x_2, -x_1)$-simple step. The colour white corresponds to areas where no induced point can be reached (with a $(-x_2, -x_1)$-simple step).

Lemma 4 *The 2-dimensional slab tree has size $O(n \log n)$, and queries take time $O(\log n)$.*

4.3 The higher dimensional slab tree

We next address the description of higher dimensional slab trees. The d-dimensional slab tree is basically a balanced binary tree on the d-th coordinate, where every node in the main tree stores a $(d-1)$-dimensional grid of appropriate size, and a number of $(d-1)$-dimensional slab trees. We first give a description of the 3-dimensional slab tree, and then we generalize to higher dimensions.

Assume without loss of generality that the direction vector is $(-x_3, -x_2, -x_1)$, and assume that we have the axis-parallel ray shooting structures of Lemma 3.

Consider the $O(n)$ horizontal planes (perpendicular to the x_3-axis) of the grid G_S. These planes partition space into a number of *elementary slices*, which we associate with the leaves of a balanced binary tree T. Every internal node δ of T corresponds to a slice which is the union of all elementary slices that are associated with the leaves of the subtree rooted at δ. Any box b either does not intersect the slice of δ, or it has vertices in the slice (a so-called x_3-*short* box at δ), or it completely cuts through the slice (a so-called x_3-*long* box at δ). Clearly, if n_δ denotes the number of leaves in the subtree rooted at δ, there are $O(n_\delta)$ x_3-short boxes at δ.

With δ, we store a 2-dimensional grid G_δ. G_δ is a grid on the front plane of the slice

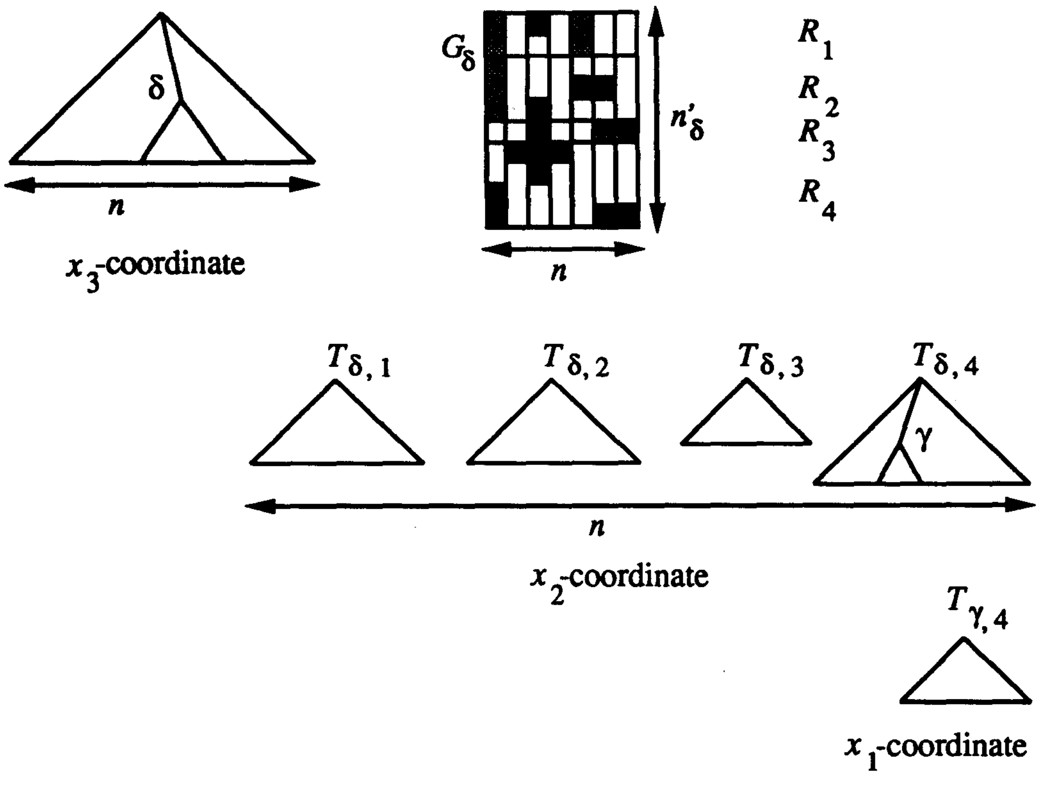


Figure 6: A 3-dimensional slab tree.

corresponding to δ. However, G_δ is not the intersection of the front plane with the full grid: we do not take a horizontal line for every different x_2-coordinate, but we only use x_2-coordinates that correspond to vertices of boxes that are x_3-short at δ. This reduces the size of G_δ from $n \times n$ to $n \times n'_\delta$, where $n'_\delta \leq 2n_\delta$. In the projection of all x_3-short and x_3-long boxes onto the grid G_δ, the first ones have all edges projected onto the set of edges of G_δ, and the edges of x_3-long boxes that are parallel to the x_1-axis may intersect edges of G_δ that are parallel to the x_2-axis. See Figure 6, where the x_3-short boxes are shaded, and the x_3-long boxes are black. G_δ can been seen as to consist of n'_δ 1-dimensional subgrids of size n called rows. Every row R_i is stored in two ways. First, we colour each row R_i implicitly using a 2-dimensional slab tree $T_{\delta,i}$, in which only the colours of induced points in the row R itself are used. Secondly, we colour R_i explicitly (i.e., we give each point of R_i the colour of the best induced point in the slice), but only colours are used of induced points that lie in a different row R_j. Note that since we consider the $(-x_3, -x_2, -x_1)$-simple step, R_j must lie below R_i. The 3-dimensional slab tree is shown schematically in Figure 6.

Before we continue to describe the structure in more detail, we show how a query is done thusfar, to improve intuition for the structure. When querying, we shoot from the query point q in $-\vec{x_3}$-direction to see how far we can go with the first link. This gives an x_3-interval I_3, with which we search in the 3-dimensional slab tree. The query interval I_3 selects $O(\log n)$ nodes that together span I_3 (as in the planar case). The query continues at each one of these nodes. Suppose that δ is such a node. The query point q projects somewhere on the grid G_δ, and we know that it cannot be in the projection of an x_3-long or x_3-short box. (Otherwise, this node would not be selected by querying with the obstacle-free x_3-interval). With q, we query in both the grid, and in the 2-dimensional slab tree corresponding to the row in which q lies. So we get two colours at δ, one corresponding to the best point that can be reached in a different row, and the other corresponding to the best point that can be reached in the row itself. The best one is selected as the answer to the query at δ. The best induced point among all the ones found at the $O(\log n)$ selected nodes, is the answer to the query in the slab tree.

We continue to describe the grid G_δ more carefully. Let m_δ be the number of induced points in the subtree rooted at δ. Let p be any such induced point. Define for p its histogram H_p (or visibility region) to be the set of points in 3-space for which p is visible with respect to the simple step $(-x_2, -x_1)$. Intuitively, consider the set of boxes and p in 3-space, and shoot from p in $+\vec{x_1}$-direction. This gives an x_1-interval. With this interval, shoot in $+\vec{x_2}$-direction, giving a possibly unbounded histogram. By definition, the query point q can see p if and only if the x_3-interval I_3 of q intersects H_p. Note that the histogram H_p of p is obstructed by all of the x_3-long boxes, and by those x_3-short boxes for which the x_3-coordinate of p lies in the x_3-interval of the x_3-short box. H_p is split into two parts: H_p^1 is the part of H_p that lies in the same row as p, and $H_p^2 = H_p - H_p^1$. The part H_p^2 is used to colour the grid G_δ explicitly, and H_p^1 is used in the 2-dimensional slab tree corresponding to the row in which p lies. The following lemma shows that the explicit colouring of G_δ using the parts H_p^2 has small size.

Lemma 5 *Any cell c in G_δ has only one non-white colour. Thus, the colouring of G_δ has size $O(n \cdot n_\delta)$.*

Proof: A simple step from a point in a cell c to another row must leave c with the first link. The only boxes that intersect c properly are x_3-long boxes, because they do not project onto the edges of the grid. Therefore, c is divided into at most two regions by some x_2-coordinate. One region lies in $-\vec{x_2}$-direction of the x_3-long box that intersects c and has minimal x_2-coordinate, and the rest of c is the second region. From the second region, it is not possible to leave cell c to another row, and it is coloured white. Since no box intersects the first region, it can either be reached completely, or not at all. Consequently, there is only one colour in the first region by Lemma 2, which is the colour of the best induced point of another row. It follows that the colouring of cell c has constant size. □

Next we describe more carefully what is done with the parts H_p^1 of histograms, which are restricted to the same row as the corresponding induced points p. Let m_δ be the

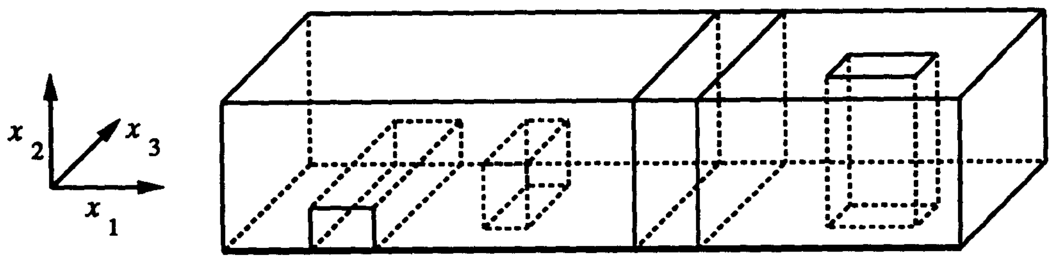


Figure 7: The subgrid G_γ with intersecting boxes that are (from left to right) x_2-short and x_3-long, x_2- and x_3-short, x_2- and x_3-long, and x_2-long and x_3-short.

number of induced points in the subtree rooted at δ. Let R_i be any one of the n'_δ rows. Let $n_{\delta,i}$ be the number of different x_2-coordinates of boxes that intersect R_i, and let $m_{\delta,i}$ be the number of induced points that project into R_i. Note that we have

$$\sum_{i=1}^{n'_\delta} n_{\delta,i} \leq 2n, \text{ and } \sum_{i=1}^{n'_\delta} m_{\delta,i} = m_\delta.$$

For row R_i, a balanced binary tree $T_{\delta,i}$ is built on the $n_{\delta,i}$ different x_2-coordinates. This tree is similar to the 2-dimensional slab tree. For a node γ in $T_{\delta,i}$, let m_γ be the number of induced points in the subtree rooted at γ. Node γ corresponds to some subgrid G_γ of G_S that projects inside R_i. This subgrid corresponds to the x_3-interval of δ, the x_2-interval of γ itself, and the full x_1-interval (the same as G_S, see Figure 7). The number of induced points in G_γ is m_γ. The colouring of this grid is stored implicitly in a 1-dimensional tree on x_1-coordinate. In fact, we colour the line that bounds the grid G_γ and has smallest x_2- and x_3-coordinate (the bottommost front line in Figure 7). One search in such a 1-dimensional tree suffices to find the best induced point of the subtree T_γ, or equivalently, of the subgrid G_γ.

Lemma 6 *The tree T_γ has size $O(m_\gamma)$.*

Proof: Consider the x_1-axis to be left–right, and sweep a plane rightward from $x_1 = -\infty$. We consider at what moments the colouring of T_γ changes, and we will charge each change to some induced point in G_γ. Any change is either caused by the appearance of an induced point, or of a box, because by Lemma 2 a change in colour can only happen when visibility regions start or stop. When a new induced point is encountered at some moment, it may change the colour of T_γ (not necessarily to its own colour). This is charged to the induced point that is encountered. Secondly, the plane can reach a box that is both x_3-long and x_2-long. Now the visibility region changes, without having a new induced point in G_γ at that x_1-coordinate. Such a box necessarily stops all visibility regions, in particular, of the induced point of current colour. We charge this change to this induced point, and that will be the last time it is charged, because its colour cannot reappear in T_γ. If the colour was white already, there isn't any change in T_γ, and nothing has to be charged. Clearly, every induced point is charged at most twice. $\square$

We return to the query again. Recall that we selected $O(\log n)$ nodes in the primary tree, and for each such node δ, we searched in the grid G_δ, and still had to query in some secondary tree of δ. If the projection q' of q is in a row R_i, then the secondary tree to search in is $T_{\delta,i}$. Therefore, we need an x_2-interval I_2 with which we should search in $T_{\delta,i}$. Thus we perform a ray shooting query in $-\vec{x_2}$-direction, starting at any point on the first link that lies within the slice of δ. Within the row R_i we can only hit an x_3-long box, thus we find either an x_3-long box, or no obstruction at all in R_i. In both cases, an x_2-interval I_2 is found with which we search in $T_{\delta,i}$. Again, this results in a subset of $O(\log n)$ nodes, which form a partition of the x_2-interval I_2. For each of these nodes, we search in the associated 1-dimensional tree.

Lemma 7 *During a search with a query point q, $O(\log^2 n)$ colours of induced points are found, and all these induced points are seen by q. The best point that can be seen from q is among these.*

Proof: For the first part, observe that in the 3-dimensional slab tree, $O(\log n)$ nodes are selected in which to proceed. For each one, one colour is found in the grid, and one 2-dimensional slab tree is searched. In the 2-dimensional slab tree, again $O(\log n)$ nodes are selected in which to proceed. For each one, we find one colour in the 1-dimensional tree. This adds up to $O(\log^2 n)$ colours.

To prove that these induced points are visible from q, consider the two possible cases. If a colour is found in the grid G_δ, then no box can obstruct the first link from q in $-\vec{x_3}$-direction by the choice of the x_3-interval I_3. By the definition of the histograms, the second and third links of the simple step are obstacle-free. If a colour is found in a tree T_γ, then the first link can be made for the same reason as above. The second link can be made by the choice of nodes in the tree $T_{\delta,i}$ after ray shooting in $-\vec{x_2}$-direction, and the third link is legal by the way T_γ is coloured.

For the third claim in the lemma, observe that the search always proceeds by using the longest x_3- and x_2-intervals that are obstacle-free. Furthermore, by definition, the histograms also give the largest possible region in which an introduced point is visible. Therefore, if some induced point is visible and the best, its colour will be found as one of the $O(\log^2 n)$ colours. $\square$

Lemma 8 *The 3-dimensional slab tree has size $O((n \log n)^2)$.*

Proof: It is easy to see that all grids together have size $O(n^2 \log n)$, by Lemma 5 and the fact that $\sum_{\delta \in T} n_\delta = O(n \log n)$. The ray shooting structure has size $O(n^2)$, by Lemma 3. The size of a tree $T_{\delta,i}$ is not of larger order than the summed sizes of the associated trees T_γ. Therefore, it is sufficient to prove a bound on the total size of all 1-dimensional trees.

To this end, we count for each induced point the number of 1-dimensional trees in which it occurs. This is easily seen to be $O(\log^2 n)$. Since the size of a 1-dimensional tree is linear in the number of induced points that occur in it, the bound follows. $\square$

This concludes the description of the 3-dimensional slab tree. The structure is extended to d dimensions in the following way. A d-dimensional slab tree is a balanced binary tree on the different x_d-coordinates. Every node δ has an associated structure, consisting of a $(d-1)$-dimensional grid G_δ of size $n \times \cdots \times n \times n'_\delta$, and for every one of the n'_δ rows, there is a $(d-1)$-dimensional slab tree. The following results are extensions of the lemmas above.

Lemma 9 *Any cell c in G_δ has only one non-white colour. Thus, the colouring of G_δ has size $O(n^{d-2} \cdot n_\delta)$. A 1-dimensional tree T_γ, corresponding to a subgrid G_γ with m_γ induced points, has complexity $O(m_\gamma)$. During a search with q, $O(\log^{d-1} n)$ colours of induced points are found, all these induced points are seen by q, and the best induced point seen by q is among these. A d-dimensional slab tree has size $O((n \log n)^{d-1})$.*

Proof: The proofs of the first four parts are straightforward generalizations of the 3-dimensional versions. For the last part, we prove that all grids together use $O(n^{d-1} \log n)$ space, and all 1-dimensional trees T_γ together use $O((n \log n)^{d-1})$ space.

Let $SC(d, n)$ be the space used by all grids in a d-dimensional slab tree with n leaves (or for a group of d-dimensional slab trees with n leaves together). Then we have

$$SC(d,n) \leq \sum_{\delta} O(n_\delta \cdot n^{d-2}) + n \cdot SC(d-1,n)$$

for $d \geq 4$, where we sum over the $O(n)$ nodes δ of the main tree. Furthermore, $SC(3, n) = O(n^2 \log n)$, by Lemma 8. With induction on d, $SC(d, n) = O(n^{d-1} \log n)$ easily follows.

Next we prove a bound on the total size of all 1-dimensional trees in a d-dimensional slab tree. To this end, we observe that each induced point occurs in $O(\log^{d-1} n)$ 1-dimensional trees. The complexity of a 1-dimensional tree is linear in the number of induced points that occur in it, and the space bound follows. $\square$

Lemma 10 *A query in a d-dimensional slab tree with a vertex of G_S and for a fixed simple step can be answered in $O(\log^{d-1} n)$ time.*

Proof: Ray shooting is performed $O(\log^{d-2} n)$ times, taking $O(\log^{d-1} n)$ time in total, by Lemma 3. Searching in grids is done $O(\log^{d-2} n)$ times, taking $O(\log n)$ time each, thus $O(\log^{d-1} n)$ time in total. Searching in the 1-dimensional trees is performed $O(\log^{d-1} n)$ times for one query. Applying fractional cascading (see [5]) to the 1-dimensional trees that are associated with one 2-dimensional slab tree, results in a total query time in 1-dimensional slab trees of $O(\log^{d-1} n)$. $\square$

We have presented the d-dimensional slab trees and the query algorithms for them. But at the beginning of this section, we assumed that queries are done with grid vertices, because for grid vertices we can prove that there is a shortest path that takes a simple step to an induced point. For an arbitrary point in d-space ($d \geq 3$), however, it can be the case that it takes many links before the shortest path to the target reaches an induced point. The following trick is used to overcome this problem. Instead of preprocessing the grid G_S, we preprocess a refinement G'_S of G_S, obtained by adding a hyperplane between any two consecutive parallel hyperplanes of G_S. This increases the complexity of the grid and the number of induced points with only a constant factor. So we get a shortest path tree and a slab tree with respect to the refined grid G'_S and the new set of induced points V'_S. Observe that in any cell (more generally: any i-face for $i > 0$) of the grid G_S, there is exactly one vertex in the refinement G'_S. For an arbitrary query point q in d-space, we compute this one vertex v_q that lies in the same cell of G_S as q. There is no need to store these $O(n^d)$ vertices; to compute v_q it suffices to store a 1-dimensional tree for each coordinate. A search in each of them gives the coordinates of v_q. Then the query in the slab tree is performed with point v_q. As q and v_q lie in the same cell in G_S, the shortest path from v_q to the target is equal to the shortest path from q to the target, up to some 'elementary' slidings of the path (slidings that do not cross new boundaries of cells). For each different simple step, we compute the distance from v_q to the target. From these distances, we get the distances from q to the target, and among these, the smallest one is selected. The corresponding path from v_q to the target is retrieved in additional $O(k)$ time (where k is the number of links of the path). This path can easily be transformed by slidings to a shortest path from q to the target in $O(k)$ time.

The d-dimensional slab trees for all simple steps, the ray shooting structures, and the sp-tree together form the data structure that solves shortest path queries. In the following section we discuss how to build slab trees. Anticipating this, we give the main result of this paper.

Theorem 2 *Given a set of n d-boxes in d-space, a target t and a non-negative real C (the cost of a turn), a data structure exists for shortest path queries to the target t in the combined metric. The structure has size $O((n \log n)^{d-1})$, and the distance from any point in d-space to the target can be found in $O(\log^{d-1} n)$ time. A shortest path can be reported in additional $O(k)$ time, where k is the number of links of the path. Preprocessing takes $O(n^d \log n)$ time.*

4.4 Construction of the slab tree

Next we show how the slab tree can be built efficiently. In the plane, this will take $O(n \log n)$ time, and in higher dimensions $O((n \log n)^{d-1})$ time. Consequently, the preprocessing of the complete shortest path structure is dominated by the time to construct the the sp-tree of Section 3, which is $O(n^d \log n)$ time. The preprocessing of the slab tree is done bottom-up. We first build a skeleton of the structure, and then fill in the

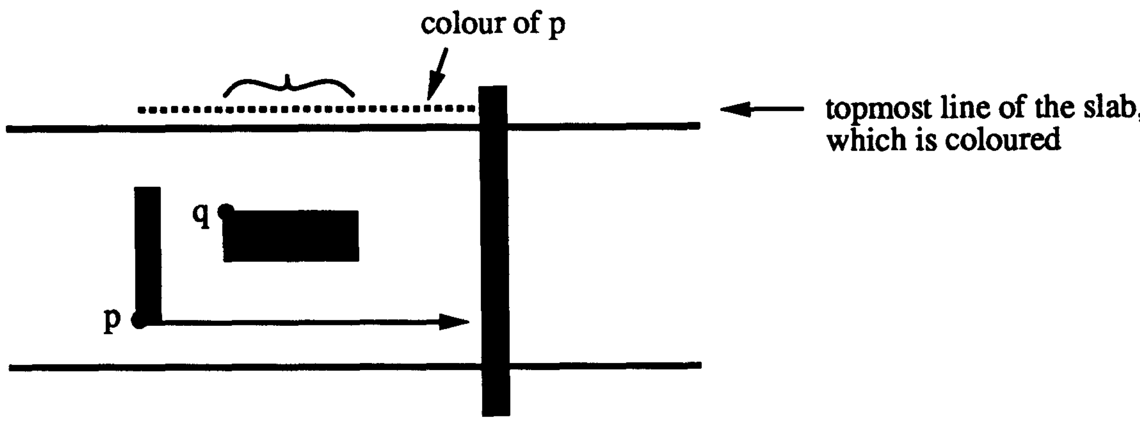


Figure 8: The incorrectness of the 2-dimensional slab tree at places where a query will not be performed. If p is the best induced point, then the colouring is as above. However, p is not always reachable, e.g. below the brace. But a query will not be performed there, because in that case the query interval intersects a box.

colours, starting at the leaves and at the lowest-dimensional trees. We repeatedly use the following observation. Assume that we are constructing a d-dimensional slab tree for the simple step $(-x_d, \ldots, -x_1)$. A grid G_δ at a node δ in a k-dimensional slab tree is only queried when, to reach this tree, the first $d-k$ links of the simple step are obstacle free. Therefore, the colouring need not be correct at places where a query is not performed anyway. This also holds for a 1-dimensional tree. In Figure 7, for example, a query is only performed at x_1-coordinates where there is no box. See also Figure 8.

We start with the planar slab tree. First, we build the skeleton. Then the associated lists of the nodes are computed in a bottom–up fashion. First we shoot in $+\vec{x_1}$-direction with each induced point p, using the structure of Lemma 3, to obtain an x_1-interval I_p. For all leaves, determine the induced points that lie in the corresponding elementary slabs. At one elementary slab, we just overlay all intervals and choose the best colour. It is easy to do this in $O(n \log n)$ time for all leaves. Observe that the colouring of the list may not be correct, because at an x_1-coordinate a colour of an induced point that cannot be reached may appear. But in that case, no query at that x_1-coordinate will be performed, because then the query interval would intersect a box in this elementary slab.

Now consider two nodes δ_{left} and δ_{right} that have the same parent δ. Because we are working bottom up, we can assume that we have the associated lists of δ_{left} and δ_{right}. From this information we have to compute the associated list of δ. This colouring is a straightforward merge of the lists of δ_{left} and δ_{right}. We simply choose the best of the two colours. Again, this colouring is correct at all x_1-values at which a query may be performed (see Figure 8).

This building algorithm takes time linear in the total size of all associated lists. We already noted that the size of an associated list of a node is linear in the number of induced points in the corresponding slab. Hence, the algorithm takes $O(n)$ time at every level of the tree and $O(n \log n)$ time in total. The application of fractional cascading does not influence the bound on the preprocessing.

The construction of the higher-dimensional slab tree is a generalization of the above procedure, but it is somewhat more involved. Again, we begin by constructing a skeleton of the whole tree. The construction of the leaves of 1-dimensional trees proceeds as before: we shoot with each induced point p to obtain an x_1-interval I_p, and we determine in which 1-dimensional tree p may appear. Then we colour a 1-dimensional tree; as in the 2-dimensional case, we just overlay the intervals and choose the best colour. Again, the colouring is correct at all x_1-coordinates that can be query values. For all 1-dimensional trees of a 2-dimensional tree with m induced points, this takes $O(m \log m)$ time.

Next we compute the colouring of the grid G_δ at a node δ in a k-dimensional slab tree. We have already computed the grids $G_{\text{left}(\delta)}$ and $G_{\text{right}(\delta)}$ of the left child and right child of δ. Furthermore, we have computed all associated structures (the $(k-1)$-dimensional slab trees) at all nodes in the subtree rooted at δ. With this information, we colour each cell of the grid G_δ. Consider such a cell c. Because every box that is short at a child of δ is also short at δ itself, the grid G_δ is a refinement of the grids of its children. Consequently, $c \subseteq c_{\text{left}}$ for some cell $c_{\text{left}} \in G_{\text{left}(\delta)}$, and $c \subseteq c_{\text{right}}$ for some cell $c_{\text{right}} \in G_{\text{right}(\delta)}$. Clearly, the best colour for cell c can be found either in the left subtree or in the right subtree of δ. Let x be the minimal x_{k-1}-coordinate of cell c, and define x_{left} and x_{right} similarly for c_{left} and c_{right}. Then we have $x \geq x_{\text{left}}$ and $x \geq x_{\text{right}}$, because G_δ is a refinement of $G_{\text{left}(\delta)}$ and $G_{\text{right}(\delta)}$. For the colour of c, we need the best colour of an induced point p in G_δ with $(k-1)$-coordinate less than x (because p should lie in another row). Consequently, the colours of c_{left} and of c_{right} are candidates for the colour of c. Furthermore, a candidate colour may lie in the x_{k-1}-interval $[x_{\text{left}} : x]$ of the left subtree, and in $[x_{\text{right}} : x]$ of the right subtree. We find these by searching with the given intervals in the associated $(k-1)$-dimensional slab trees corresponding to the rows containing c_{left} and c_{right}, respectively. In fact, we either have $x = x_{\text{left}}$ or $x = x_{\text{right}}$, so we do not have to search in both subtrees. Of the three candidates we find (the colour of c_{left}, the colour of c_{right}, and a colour from the associated search), we choose the best one for the colour of c. With this procedure, it takes $O(\log^{k-2} n)$ time to determine this colour. By Lemma 8, all grids together contain $O(n^{d-1} \log n)$ cells, which leads to a total construction time of $O((n \log n)^{d-1})$, since $k \leq d$.

This leads to

Lemma 11 *The d-dimensional slab tree can be constructed in $O((n \log n)^{d-1})$ time.*

5 Conclusions and directions for further research

A shortest path problem has been studied, which introduced two new aspects. First, a metric has been used that generalizes both the L_1- and link metric. The cost of a path was defined to be its L_1-length, plus some constant times the number of turns the path makes. The second important feature is that a shortest path problem is studied in arbitrary (fixed) dimension. As far as we know, there are no other results on shortest paths in dimensions greater than three, and for link distance there are no results in dimensions greater than two.

This paper also shows that the general *rectilinear* shortest path problem can be solved in polynomial time in arbitrary dimension (in fact, in $O(n^d \log n)$ time), whereas the general *Euclidean* problem is already NP-hard in 3-dimensional space[2].

Many areas for future research lie open. First of all, it may be possible to improve our results. Furthermore, it is interesting to find other restricted cases of higher dimensional shortest path problems that are solvable in polynomial time. Finally, we think that the generalization of our metric to non-rectilinear paths is worth studying.

References

[1] Canny, J. F., *The Complexity of Robot Motion Planning.* ACM Doctoral Dissertation Award, MIT Press, 1987.

[2] Canny, J., and J. Reif, New Lower Bound Techniques for Robot Motion Planning Problems. *Proc. 29th IEEE Symp. on Foundations of Computer Science* (1987), pp. 49–60.

[3] Chazelle, B., A Theorem on Polygon Cutting with Applications. *Proc. 23rd IEEE Symp. on Foundations of Computer Science* (1982), pp. 339–349.

[4] Chazelle, B., Triangulating a Simple Polygon in Linear Time. *Proc. 31st IEEE Symp. on Foundations of Computer Science* (1990), pp. 220–230.

[5] Chazelle, B., and L. J. Guibas, Fractional Cascading: I. A Data Structuring Technique. *Algorithmica* 1 (1986), pp. 133–162.

[6] Chen, J., and Y. Han, Shortest Paths on a Polyhedron. *Proc. 6th ACM Symp. on Comp. Geom.* (1990), pp. 360–369.

[7] Clarkson, K., S. Kapoor, and P. Vaidya, Rectilinear Shortest Paths through Polygonal Obstacles in $O(n \log^2 n)$ Time. *Proc. 3rd ACM Symp. on Comp. Geom.* (1987), pp. 251–257.

[8] de Berg, M., On Rectilinear Link Distance. To appear in: *Comp. Geom.: Theory and Applications.*

[9] de Berg, M., M. van Kreveld, B. J. Nilsson, and M. Overmars, Finding Shortest Paths in the Presence of Orthogonal Obstacles using a Combined L_1 and Link Metric. *Proc. SWAT 1990*, Lect. Notes in Comp. Science 447 (1990), pp. 213–224.

[10] de Rezende, P. J., D. T. Lee, and Y. F. Wu, Rectilinear Shortest Paths with Rectangular Barriers. *Discr. & Comp. Geom.* 4 (1989), pp. 41–53.

[11] Dijkstra, E. W., A Note on Two Problems in Connection with Graphs. *Numer. Math.* **1** (1959), pp. 269–271.

[12] Edelsbrunner, H., L. J. Guibas, and J. Stolfi, Optimal Point Location in a Monotone Subdivision. *SIAM J. Comput.* **15** (1986), pp. 317–340.

[13] Ke, Y., An Efficient Algorithm for Link Distance Problems. *Proc. 5th ACM Symp. on Comp. Geom.* (1989), pp. 69–78.

[14] Kirkpatrick, D. G., Optimal Search in Planar Subdivisions. *SIAM J. Comput.* **12** (1983), pp. 28–35.

[15] Larson, R. C., and V. O. Li, Finding Minimum Rectilinear Distance Paths in the Presence of Barriers. *Networks* **11** (1981), pp. 285–304.

[16] Lee, D. T., *Proximity and Reachability in the Plane.* Ph.D. Thesis, University of Illinois, 1978.

[17] Lee, D. T., and F. P. Preparata, Euclidean Shortest Paths in the Presence of Rectilinear Barriers. *Networks* **14** (1984), pp. 393–410.

[18] Lenhart, W., R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides, and C. Yap, Computing the Link Center of a Simple Polygon. *Proc. 3rd ACM Symp. on Comp. Geom.* (1987), pp. 1–10.

[19] Mehlhorn, K., *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry.* Springer-Verlag, Berlin, 1984.

[20] Mitchell, J. S. B., An Optimal Algorithm for Shortest Rectilinear Paths Among Obstacles in the Plane. *Abstracts of the 1st Canad. Conf. on Comp. Geom.* (1989), p. 22.

[21] Mitchell, J. S. B., D. M. Mount, and C. H. Papadimitriou, The Discrete Geodesic Problem. *SIAM J. Comput.* **16** (1987), pp. 647–668.

[22] Mitchell, J. S. B., G. Rote and G. Wöginger, Computing the Minimum Link Path Among a Set of Obstacles in the Plane. *Proc. 6th ACM Symp. on Comp. Geom.* (1990), pp. 63–72.

[23] O'Rourke, J., and C. Schevon, Computing the Geodesic Diameter of a 3-Polytope. *Proc. 5th ACM Symp. on Comp. Geom.* (1989), pp. 370–379.

[24] Reif, J., and J. A. Storer, *Shortest Paths in Euclidian Space with Polyhedral Obstacles.* Techn. Rep. CS–85–121, Comp. Science Dept., Brandeis University, Waltham, MA, 1985.

[25] Shamos, M. I., *Computational Geometry.* Ph.D. Thesis, Yale University, New Haven, CN, 1978.

[26] Sharir, M., and A. Schorr, On Shortest Paths in Polyhedral Spaces. *SIAM J. Comput.* **15** (1986), pp. 193–215.

[27] Suri, S., *Minimum Link Paths in Polygons and Related Problems.* Ph.D. Thesis, Johns Hopkins University, 1986.