GEOMETRIC DATA STRUCTURES

A Dissertation
submitted to the department of COMPUTER SCIENCE
of Tufts University
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Syed Muhammad Mashhood Ishaque
September 2010
Advisor: Diane L. Souvaine

Abstract

A data structure is a repository of information; the goal is to organize the data so that it needs less storage (space) and so that a request for information (query) can be processed quickly. A geometric data structure handles data which have locations attached (e.g. addresses of fire stations in the state of Massachusetts). Geometric data structures have become a pervasive and integral part of life, and can be queried to produce driving directions or the name of the nearest Italian restaurant. Since the space and query time of a data structure depend upon the type of queries it needs to support, it is important to study which tools and techniques are suitable for which data structures. The ongoing quest for better data structures sometimes results in improved methods and sometimes results in entirely new techniques. The goal is to determine optimal data structures with the best possible performance.

Given a set S of n points in $\mathbb{R}^d$, a data structure for geometric range searching may report: whether the query range contains any point (emptiness), the number of points in the range (counting), all points in the range (reporting), or the minimum/maximum point according to some criterion (optimization). A query range can be a circle, a halfspace, an axis-parallel rectangle, a simplex or some other geometric object. Geometric range searching is fundamental to computer science, and is a well-studied problem. There have been two themes to the study of geometric range searching: creating better data structures (upper bounds) for the many variants of range searching and proving lower bounds in different models of computation in order to understand which range searching problems are difficult e.g. halfspace counting has been shown to be much harder than halfspace emptiness.

In this thesis, we prove new lower bounds for simplex emptiness queries in the partition graph model. Our lower bounds are based on the lower bounds provided by Erickson [40]

for online hyperplane emptiness problem in the partition graph model, and are within a polylogarithmic factor of the optimal in the plane. Previously known lower bounds for simplex emptiness were based on the rather weak lower bounds due to Erickson [40] for halfspace emptiness in the partition graph model (only trivial lower bounds were known for $2 \leq d \leq 4$). Our lower bounds automatically imply lower bounds for simplex range reporting, where a data structure needs to report all r points contained in the query simplex. Chazelle and Rosenberg [34] had previously established lower bounds for simplex range reporting in the pointer machine model, but unfortunately their lower bounds only hold for the case when r is at least n^{δ} for some $\delta > 0$. Our lower bounds apply to host of other important problems such as point-inclusion in union of slabs, segment intersection searching, implicit point location, line-nearest neighbor and segment dragging queries etc.

Although these lower bounds make it impossible to create efficient data structures for various geometric problems, these data structures are not used in isolation but by specific algorithms. These algorithms may generate a sequence of queries with some special structure making it possible to beat the lower bounds by designing data structures specific to these algorithms. For example, Paterson and Yao's [64] classical randomized auto-partition algorithm can be implemented using a dynamic ray shooting data structure for disjoint polygonal obstacles, yielding a runtime of $O(n^{\frac{3}{2}} \lg n)$. Since the algorithm does not "really need" a general ray shooting data structure, we were able to improve the runtime of the algorithm to $O(n \lg^2 n)$ by developing a new data structure for ray shooting-and-insertion in the free space between disjoint polygonal obstacles. For a ray shooting-and-insertion query, the ray starts at the boundary of some obstacle, and the portion of the ray between the starting point and the first obstacle hit is inserted permanently as a new obstacle. The data structure uses $O(n \lg n)$ space and preprocessing time, and it supports m successive ray shooting-and-insertion queries (for sufficiently large m), in $O(n \lg^2 n + m \lg m)$ total time. For the auto-partitioning algorithm, we perform $m = O(n \lg n)$ shooting-and-insertion queries yielding a runtime of $O(n \lg^2 n)$

Finally, we present a useful tool for building data structures—convex partitions with 2-edge connected dual graphs. Convex partitions have proved to be very useful for creating efficient data structures especially in computer graphics. Given a set of convex polygonal obstacles and a bounding box, we may think of the bounding box as a simple polygon

and the obstacles as polygonal holes. Then the problem of creating a convex partition becomes that of decomposing the simple polygon with holes into convex parts. Convex polygonal decomposition has received considerable attention in the field of computational geometry. The focus has been to produce a decomposition with as few convex parts as possible. Lingas [59] showed that finding the *minimum convex decomposition* is NP-hard for polygons with holes. While minimum convex decomposition is desirable, it is not the only criterion for the *goodness* of a convex partition. Another criterion for the quality of a convex partition might be some property of its dual graph. In this thesis, we give a convex partitioning scheme such that the dual graph of the produced convex partition is 2-edge connected.

The take-away message for the thesis is that we need to create specialized data structures. The idea is not new, and has been pursued before in computer science, however, the existence of severe lower bounds for various geometric problems make the notion even more appealing.

Acknowledgements

I would like to thank my advisor Prof. Diane Souvaine for her guidance and support over the last few years. She allowed me the independence to purse problems of my choice, while providing feedback whenever I needed. I do not think I could have found a better environment to do research, and I will be forever indebted to her for this opportunity. I want to express my gratitude to my committee members: Prof. Cowen, Prof. Boghosian, Erik and Csaba for their time. I want to acknowledge Ben Hescott who has always been a wonderful source of advice. A major part of this thesis has been a joint work with Csaba. I learnt a lot from Csaba, and I cannot thank him enough for that. I also want to thank Erik, Marty and the rest of the MIT-Tufts research group for some of the most fun meetings of my life. I think I would really miss Marty's jokes. I want to thank Prof. Stafford for being so supportive of me, and I really appreciate the opportunity to be a guest lecturer in her classes. I want to thank Marwan and Sarah for allowing me to mentor them. I must mention the best staff one could ever wish for: Jeannine, Donna, Gail and George were always there to help. Especially George was always around whenever I needed to see a friendly face.

I owe the most heartfelt gratitude to Dr. Abbas Zaidi who inspired me to undertake graduate studies. He told me that graduate school was "the most rewarding experience" of his life, and it turned out to be best experience of my life as well.

I want to thank my friends and roommates Hank, Zinger, Guang Tao, Ali, Alireza, Bilal, Ehsan, Danyal, Marwan, Nathan, Nauman, Saad, Saeed, Samina and Shirwac for making my stay at Tufts extremely enjoyable. In the end I would like to thank my family for all they have done for me.

Dedication

إذا عبتها شبهتها البدر طالعا
وحسبك من عيب لها شبه البدر
لقد فضلت لبنى على الناس مثلما
على ألف شهر فضلت ليلة القدر

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Range searching data structures are some of the most fundamental data structures in computational geometry. These data structures have applications in graphics, geographic information systems, and spatial data bases. The goal of a range searching data structure is to preprocess a set of n points in $\mathbb{R}^d$ so that questions about a query range can be answered quickly. Although the kind of query range may depend upon the application, halfspaces, axis-parallel rectangles and simplicies are some of the more important query ranges in practice. We focus on the case where the range is a full dimensional simplex in Euclidean d-space $\mathbb{R}^d$, which is the basic building block for queries on many other polyhedral objects.

Simplex range searching is an umbrella term that includes *simplex emptiness* queries asking whether there is any point in the query simplex, *simplex reporting* queries asking for all points in the query simplex, and *simplex counting* asking for the number of points. In the weighted version of range counting, each point is assigned a weight from a semigroup (group), and a *range semigroup (group)* query asks for the semigroup (group) sum of the weights for points in the query range.

Simplex range searching is a well-studied problem in computational geometry and believed to be "almost completely solved" [1]. Chazelle [27] established quasi-optimal (within a polylogarithmic factor of the optimal) lower bounds for simplex counting queries in the semigroup arithmetic model. For range reporting, Chazelle and Rosenberg [34] established the quasi-optimal lower bounds for simplex reporting in the pointer machine model albeit for the case the number of points being reported is at least n^δ points for some

$\delta > 0$. However, for simplex range emptiness there is a huge gap between known upper and lower bounds. In this thesis we bridge this gap by improving the lower bounds on simplex emptiness.

Although the lower bounds presented for these geometric problems are very severe and thus there is no hope for efficient data structures. However, these geometric data structures do not exist in isolation but are used by applications (algorithms). These algorithm may have some special structure that we can exploit to get around these lower bounds. For example, Paterson and Yao's [64] classical randomized auto-partition algorithm can be implemented using a dynamic ray shooting data structure yielding a runtime of $O(n^{\frac{3}{2}} \lg n)$. Since the algorithm does not "really need" a general ray shooting data structure, therefore, we were able to improve the runtime of the algorithm to $O(n \lg^2 n)$ by developing a new data structure for ray shooting-and-insertion in the free space between disjoint polygonal obstacles.

Here we give an overview of the contributions of this thesis.

1.1 Lower Bounds for Simplex Emptiness

The best known upper bounds for simplex range searching are due to Matoušek [60] in the RAM model of computation. The data structure is based on the simplicial partitioning method and can support simplex counting in $O(\frac{n}{m^{\frac{1}{d}}} \lg^{d+1}(\frac{m}{n}))$ time while using m space. The preprocessing time has recently been improved by Chan [23]. The upper bounds for simplex counting immediately apply for simplex emptiness. While the data structures for simplex counting are optimal within a polylogarithmic factor in the semigroup arithmetic model [27], the lower bounds for simplex emptiness are far from optimal. Previously known lower bounds for simplex emptiness are rather weak; derived from halfspace emptiness queries through a lifting method for $d \geq 5$. Only trivial lower bounds were known for dimensions $2 \leq d \leq 4$.

We establish new lower bounds for the simplex emptiness queries in $\mathbb{R}^d$ in a variant of the partition graph model introduced by Erickson [40] for hyperplane emptiness queries. We reduce hyperplane emptiness to simplex emptiness. A data structure for simplex emptiness in the polyhedral partition graph model must spend $\Omega(\frac{n^{1-\frac{1}{d}}}{\text{polylog}\, n})$ time answering a query

when the preprocessing is restricted to be $O(n \operatorname{polylog} n)$. These lower bounds are quasi-optimal (within a polylogarithmic factor of the optimal), and hold in the worst case. We also prove weaker lower bounds in the general partition graph model, with no restriction on preprocessing, that hold in the average case. Our lower bounds imply lower bounds for simplex range reporting in the polyhedral partition graph model irrespective of the number of points being reported. Previous lower bounds for simplex range reporting hold only for queries containing at least n^{δ} points for some $\delta > 0$. We also reduce simplex emptiness to various other geometric problems such as segment intersection searching, implicit point location, , line-nearest neighbor, segment dragging, and point-inclusion in a union of slabs, thus establishing the lower bounds for these problems as well.

1.2 Data Structures for the Restricted Simplex Queries

Chazelle *et al.* [32] gave a linear-space data structure for halfplane range reporting that achieves $O(\lg n + r)$ query time, where r is the number of points being reported. The data structure maintains nested (peeling) convex layers for the given point set. Similarly for halfplane emptiness queries, a linear-space data structure that maintains the convex hull of the given point set allows the queries to be answered in $O(\lg n)$ time.

We consider the restricted version of simplex emptiness and reporting in the plane, where each query simplex contains the origin. The restricted version of simplex range searching behaves similar to halfplane range searching: for range emptiness and reporting we can create near-linear space ($O(n^{1+\varepsilon})$) data structures for any fixed $\varepsilon > 0$, with polylogarithmic query times. For the restricted simplex range counting queries the lower bounds for the halfplane range counting queries continue to hold: a near-linear space data structure cannot support counting queries in polylogarithmic time.

1.3 Data Structures for Permanent Ray Shooting

Ray shooting data structures are a classical core component of computational geometry. They store a set of preprocessed objects in space in such a way that one can efficiently find the first object hit by a query ray. Geometric algorithms often rely on a ray shooting data

structure, where the result of each query may affect the course of the algorithm and modify the data. Successive ray shooting queries are responsible for a bottleneck in the runtime of some geometric algorithms, which recursively partition the plane along rays (along the portion of rays between their starting points and the first obstacles hit, to be precise).

We present a data structure for ray shooting-and-insertion queries among disjoint polygonal obstacles lying in a bounding box B in the plane. Each query is a point p on the boundary of an obstacle and a direction d_p; we report the first point q where the the ray emanating from p in direction d_p hits an obstacle or the bounding box (ray shooting) *and* insert the segment pq (insertion) as a new obstacle edge . If the input polygons have a total of n vertices, our data structure uses $O(n \log n)$ preprocessing time, and it supports m shooting-and-insertion queries in $O((n+m) \log^2 n + m \log m)$ total time and $O((n+m) \log(n+m))$ space. We present two applications for our data structure: efficient implementation of *auto-partitioning* and *convex partitioning* algorithms. Our data structure improves the runtime of Paterson and Yao's [64] classical randomized auto-partition algorithm from $O(n^{\frac{3}{2}} \lg n)$ to $O(n \lg^2 n)$.

A simple polygon with n vertices can be preprocessed in $O(n)$ time to answer ray shooting queries in $O(\log n)$ time, using either a balanced geodesic triangulation [30] or a Steiner triangulation [48]. However, the free space between disjoint polygonal obstacles with a total of n vertices (e.g. $\frac{n}{2}$ disjoint line segments) cannot be handled as easily. The best ray shooting data structures can answer a query in $O(\frac{n}{\sqrt{m}})$ time (ignoring polylogarithmic factors) using $O(m)$ space and preprocessing, based on range searching data structures via parametric search. That is, a query takes $O(\sqrt{n})$ time on the average using $O(n)$ space. The biggest challenge in the design of our data structure was bridging the gap between the $O(\log n)$ query time for ray shooting in a simple polygon and the $O(\sqrt{n})$ query time among disjoint obstacles with n vertices. Our data structure is based on two tools: geometric partition trees in two dimensions and geodesic hulls. Here we briefly describe our data structure. In each convex cell of the geometric partition tree, we maintain the geodesic hull of all reflex vertices; a vertex is reflex if it forms a reflex angle in the free space. The geodesic hull separates the obstacles lying in the interior of the cell from all other obstacles. The geodesic hulls form a nested structure of depth $\lg n$ that consists of weakly simple polygons and creates a tiling of the free space. Each tile is a simple polygon

that can easily be processed for fast ray shooting queries. A ray shooting query can be answered by tracing the query ray along these polygons.

The use of geodesic hulls allows us to control the total complexity of m ray insertion queries. Basically, a query ray intersects the boundary of a geodesic hull only if it partitions the set of reflex vertices into two nonempty subsets. Since a set of k points can recursively be partitioned into nonempty subsets at most $k - 1$ times, we can charge the total number intersections between rays and geodesic hulls to the number of such partition steps.

1.4 Convex Partitions with 2-Edge Connected Dual Graphs

Convex partitioning decomposes a complex geometric object into convex parts. It is a useful tool in computer graphics, motion planning, and geometric modeling. Convex polygonal decomposition has received considerable attention in the field of computational geometry. The focus has often been to produce a decomposition with as few convex parts as possible. Lingas [59] showed that finding the *minimum convex decomposition* (decomposing the polygon into the fewest number of convex parts) is NP-hard for polygons with holes. For polygons without holes, however, minimum convex decompositions can be computed in polynomial time [29, 54]—see [53] for a survey on polygonal decomposition.

While minimum convex decomposition is desirable, it is not the only criterion for the *goodness* of a convex partition (decomposition). In fact, the measure of the quality of a convex partition can be specific to the application domain. In Lien's and Amato's work on approximate convex decomposition [58] with applications in skeleton extraction, the goal is to produce an approximate (not all cells are convex) convex partition that highlights salient features. In the *equitable* convex partitioning problem, all convex cells are required to have the same value of some measure e.g. the same number of red and blues points [52], or the same area [22] (with application to vehicle routing).

Another criterion for the quality of a convex partition might be some property of its dual graph (the definition of dual graph varies from application to application). A dual graph might represent a communication network whose desired characteristic is fault tolerance (no single point of failure). We consider the problem of creating convex partitions with 2-edge connected dual graphs.

For a finite set S of disjoint convex polygonal obstacles in the plane $\mathbb{R}^2$, a *convex partition* of the free space $\mathbb{R}^2 \setminus (\bigcup S)$ is a set C of open convex regions (called *cell*s) such that the cells are pairwise disjoint and their closures cover the entire free space. Since every vertex of an obstacle is a reflex vertex of the free space, it must be incident to at least two cells. A dual graph of the convex partition is obtained by creating a node for each cell in C, and adding an edge between two nodes of the dual graph if the two corresponding convex cell share an obstacle vertex.

It is straight forward to construct an arbitrary convex partition for a set of convex polygons as follows. Let V denote the set of vertices of the obstacles; each vertex of a convex obstacles is reflex. Let π be a permutation on V. Process the vertices in the order π. For a vertex $v \in V$, draw a directed line segment (called *extension*) that starts from the vertex along the angle bisector. For a line-segment obstacle, the extension is drawn along the supporting line. The extension ends when it hits another obstacle, a previous extension, or infinity (the bounding box). We call this a STRAIGHT-FORWARD convex partition. The order in which the extensions are drawn determines the convex partition we get.

Aichholzer *et al.* [7] conjectured for a set of disjoint line segments, there always exists a STRAIGHT-FORWARD convex partition such that dual graph of the partition is 2-edge connected. Since there are exponential number of STRAIGHT-FORWARD convex partitions for a given set of line segments ($(2n)!$ permutations for n segments), it is reasonable to conjecture that one of them may have the 2-edge connected dual graph. However, we construct a counterexample using 16 line segments where **no** permutation produces a STRAIGHT-FORWARD convex partition with 2-edge connected dual graph.

Although the conjecture due Aichholzer *et al.* is not true, it is still possible to create convex partitions with 2-edge connected dual graphs. We prove that for every finite set of disjoint convex polygons in the plane there is a convex partition (not necessarily STRAIGHT-FORWARD) with 2-edge connected dual graph. The dual graph has the same number of nodes as in the case of STRAIGHT-FORWARD convex partition.

1.5 Outline of the Thesis

The thesis is organized as follows: in Chapter 2, we describe our lower bounds for simplex range emptiness. In Chapter 3, we describe data structures for a restricted version of simplex emptiness that achieve better upper bounds than those possible for normal simplex emptiness. In Chapter 4, we present a data structure for ray shooting-and-insertion that improves the runtime of various auto-partitioning and convex partitioning algorithms. In Chapter 5, we give an algorithm to produce a convex partition of the free space between disjoint convex obstacles such that the dual graph of the partition is 2-edge connected. We conclude in Chapter 6.

Chapter 2

Lower Bounds for Simplex Emptiness

Range searching is one of the most fundamental problems in computational geometry. It has applications in graphics, geographic information systems, and spatial data bases. The goal is to preprocess a set of n points in Euclidean d-space $\mathbb{R}^d$ into a data structure so that questions about a query range can be answered quickly. We focus on the case where the range is a full dimensional simplex in $\mathbb{R}^d$, which is the basic building block for queries on arbitrary polyhedral objects. Simplex range searching is an umbrella term that includes *simplex emptiness* queries asking whether there is any point in the query simplex, *simplex reporting* queries asking for all points in the query simplex, and *simplex counting* asking for the number of points. In the weighted version of range counting, each point is assigned a weight from a semigroup (group), and a *range semigroup (group)* query asks for the semigroup (group) sum of the weights for points in the query range. See surveys on geometric range searching by Matoušek [61], and by Agarwal and Erickson [1].

Matoušek [60] showed that in the RAM model of computation, a data structure of size m can support simplex counting in $O(\frac{n}{m^{\frac{1}{d}}} \lg^{d+1}(\frac{m}{n}))$ time. The space and preprocessing time has recently been improved by Chan [23]; he gave a randomized method for preprocessing with an expected runtime of $O(n \operatorname{polylog} n)$. These upper bounds are fundamentally built on the partition tree technique [78] and the ε-net theory [46]. The upper bounds for simplex counting immediately apply for simplex emptiness, and for many optimization variants of simplex range searching (e.g. closest point to a line, etc.) discussed in Section 2.7. However, establishing lower bounds is much more challenging. Especially for

simplex emptiness there is a huge gap between known upper and lower bounds. The difficultly arises from the fact the models of computation used to establish lower bounds for simplex range searching are too powerful for the special case of simplex range emptiness. In this thesis, we establish quasi-optimal lower bounds (within a polylogarithmic factor of the optimal) on simplex emptiness in the modified partition graph model. Before we describe the new lower bounds, we discuss various model of computations in Section 2.1.

2.1 Models of Computation

A model of computation is a mathematical abstraction that specifies the allowed operations (computing steps), and the cost of these operations. A model can be stronger or weaker depending upon the kind of operations it allows. Once a model of computation has been specified, we can implement algorithms in it (upper bounds) and study their performance. We can also prove lower bounds on problems by showing any algorithm in the chosen model will take a certain number of operations to solve the particular problem. Ideally, we want to prove the upper bounds in weaker models, and the lower bounds in more powerful models of computation. One well-known example of a model of computation is the Turing machine model. The Turing machine model, with an infinite tape (memory) and sequential access, captures the notion of computability—which problems are solvable (decidable), however, it is not suitable for studying the performance of an algorithm. The RAM model, with random access memory, serves better for analyzing the runtime of an algorithm. The RAM model is very close to reality, hence an algorithm for the RAM model will run on a real computer (ignoring the issue of finite precision).

One might ask: if we already have a realistic model of computation—the RAM model, is there any need for a different model of computation? The answer is 'yes' because the purpose of different models is to capture different aspects of computation e.g. a model for external memory algorithms may focus on data transfers between disk and memory, and allow CPU operations for free. Moreover, showing lower bounds in the RAM model has the risk that if one day we are able to implement a more powerful model into a real computer (say Quantum computers), the lower bounds in the RAM model will become irrelevant.

In this thesis, our focus is on lower bounds for data structures, thus we need a model

that captures the interaction with the data structure and abstracts away other costs. Data structures for range searching are fundamental to computer science and have led to development of various models of computation of varying strength. For example, the semigroup arithmetic model was developed to show lower bounds for simplex range counting, but it is too powerful for range emptiness. Here we briefly discuss the three important models of computation for geometric range searching: the semigroup arithmetic model, the pointer machine model, and the partition graph model. Since we establish our lower bounds in the partition graph model, we give a detailed description of it in Section 2.2.

2.1.1 The Semigroup Arithmetic Model

The *semigroup arithmetic model* was introduced by Fredman [41] and modified by Yao [79]. The model was developed to prove lower bounds on range counting. In this model each point is assigned a weight from a semigroup, and a semigroup addition operator is defined. There is no subtraction operator (unlike for the group model). For example, we can attach a unit weight with each point and define the semigroup addition operator to be the arithmetic addition, then a semigroup range query gives the count of the points in the query range. Weights are not necessarily real numbers. For a range reporting query, we can attach with every point a singleton containing the label of the point and define the semigroup addition operator to be the set union.

The data structure consists of a set of precomputed partial sums (semigroup sums of the subsets of points). The size of the data structure is the number of precomputed partial sums. A query is answered by performing semigroup additions to compute the sum of the weights in the query range, and the query time is the minimum number of additions needed.

Chazelle [27] established quasi-optimal (within a polylogarithmic factor of Matoušek's upper bounds in the RAM model) lower bounds for simplex counting queries in the semigroup arithmetic model. The model is too powerful for simplex emptiness; a simplex emptiness query takes $O(1)$ time in this model, since the query simplex is not empty if we perform even a single semigroup addition.

As we mentioned earlier, in the group model the weights associated with points come from a group i.e. subtraction operator is defined as well. Group model is very powerful,

hence it is very difficult to show lower bounds in this model. The only lower bounds known in the group model are for orthogonal range searching (see [28, 65]).

2.1.2 The Pointer Machine Model

The *pointer machine model* was developed by Tarjan [71, 72] by extending Knuth's *linking automaton* [56] to model list-processing problems. The model was later generalized by Chazelle [26]. The pointer machine model does not allow memory-address arithmetic, hence it is weaker than the RAM model. However, there exists numerous extensions to the pure pointer machine model. Here we discuss a powerful variant of the pointer machine model that allows nondeterminism.

The data structure is a directed graph. There is a fixed root node and each node has an outdegree 2. The size of the data structure is the number of nodes in the graph. A query is answered by traversing this graph starting from the root node. The computation to determine which node to visit next is free of cost (nondeterminism). The query processing algorithm is only charged for traversing a node. The nodes traversed ("working set") must contain the answer to the query, and the query time is the size of this set. For example, for a simplex emptiness query, if the given query simplex is not empty, then the query algorithm must visit a node associated with some arbitrary point (witness) inside the query simplex.

Chazelle and Rosenberg [34] established the quasi-optimal lower bounds for simplex reporting in the (nondeterministic) pointer machine model by showing that a data structure with $O(m)$ space must spend $\Omega(\frac{n^{1-\varepsilon}}{m^{\frac{1}{d}}} + r)$ time to answer a reporting query, where $\varepsilon > 0$ and r is the number of points being reported. However, the lower bounds only hold for $r = \Omega(\frac{n^{1-\varepsilon}}{m^{\frac{1}{d}}})$.

The reason for such restrictive lower bounds is that the (nondeterministic) pointer machine model is too powerful. One can easily implement a $O(n)$ space data structure that supports simplex reporting queries in $O(r \lg n)$ time by building a balanced binary tree on the given set of points, then any point in the query simplex can be reported by traversing at most $O(\lg n)$ nodes from the root. For the same reason a simplex emptiness query takes $O(\lg n)$ time in the pointer machine model.

2.1.3 The Partition Graph Model

The *partition graph model* was introduced by Erickson [40] for hyperplane emptiness queries (see also [39]). The data structure is built on a directed acyclic graph, called a *partition graph*. There is a fixed root node. Every node has a constant outdegree. Associated with every outgoing edge of a node is a partition (a connected region) of $\mathbb{R}^d$. The regions associated with the outgoing edges of a node, together cover the entire $\mathbb{R}^d$ space. The size of the data structure is the number of edges in the graph. The cost of building the partition graph is zero, and the cost of preprocessing is the total number of edges traversed as we store the points in this partition graph. A query is answered by traversing this graph starting at the root node (details in Section 2.2) and the query time is the number of edges traversed while answering a query. Any computation done outside the partition graph is free, however, the answer to any query should be verifiable by a valid query processing algorithm. Erickson defined two variations of the partition graph model: the *polyhedral* and *semialgebraic* partition graph models, in which the regions are constant-complexity polyhedra and semialgebraic sets, respectively.

2.2 The Modified Partition Graph Model

We use the partition graph model developed by Erickson [39, 40], and utilize prior bounds on hyperplane emptiness in this model to achieve new lower bounds for both simplex emptiness and simplex reporting. The partition graph model was initially developed for analyzing the data structures for hyperplane emptiness queries, and is very *ad hoc* in nature. So we adapt the original partition graph model to support data structures for simplex range queries. Our modification are very similar to what Erickson did to support halfspace emptiness data structures in the partition graph model, thus do not change the model significantly. For completeness sake, we describe the partition graph model as defined by Erickson, and then highlight the modifications necessary for simplex range queries.

Every data structure in the partition graph model is based on a *partition graph*. A partition graph is a directed acyclic (multi-)graph, with one source, called the root, and several sinks, called leaves. Associated with each non-leaf node v is a set R_v of regions

called the *query regions*, satisfying three conditions.

1. R_v contains at most Δ query regions, for some constant $\Delta \geq 2$.
2. Every query region is a connected subset of $\mathbb{R}^d$.
3. The union of the query regions in R_v is $\mathbb{R}^d$.

The node v has an outgoing edge for each query region in R_v. Thus, the outdegree of each node is at most Δ. The indegree can be arbitrarily large. In addition, every internal node v is labeled either a primal node or a dual node, depending on whether its query regions R_v are interpreted as a partition of primal or dual space. The query regions associated with primal (resp. dual) nodes are called the *primal* (resp. *dual*) query regions.

The model does not require the query regions to be disjoint. The query regions are not required to be convex or semialgebraic, however, a few of Erickson's results only hold for the partition graphs with query regions that are constant-complexity polyhedra or constant-complexity semialgebraic sets. If all the query regions in a partition graph are constant-complexity polyhedra, it is called a *polyhedral partition graph*. If all the query regions are constant-complexity semialgebraic sets (also called Tarski cells), it is called a *semialgebraic partition graph*.

In the partition graph model the construction of the partition graph is free of cost, thus the optimal partition graph can be constructed for any given point set. Once the optimal partition graph has been built, the set of point S is preprocessed into a data structure (Section 2.2.1). A query is answered by searching this partition graph (Section 2.2.2).

2.2.1 Preprocessing in the Partition Graph Model

For each point $p \in S$, we perform a depth-first search of the partition graph using the query regions to determine which edges to traverse, and we stop when we reach a leaf node. Whenever we reach a primal node v, we traverse the outgoing edges corresponding to the query regions in R_v that contain p. Whenever we reach a dual node v, we traverse the edges corresponding to the query regions in R_v that intersect the dual hyperplane p^*. For hyperplane emptiness, Erickson stored at each leaf node the subset of points that reached that leaf node. Here we store a subset at each node except root. For each node v with

nonzero indegree, we maintain a subset P_v containing the points that reach that node. See Algorithm 1 below. The cost of preprocessing is the total number of edges traversed for the point set.

Algorithm 1 PREPROCESS (p)

Let r be the root of the partition tree.
TRAVERSE (p, r)

Algorithm 2 TRAVERSE (p, v)

if v is a primal internal node **then**
  for each outgoing edge (v, w) **do**
    if the region corresponding to (v, w) contains p **then**
      Add p to the subset P_w
      TRAVERSE (p, w)
    end if
  end for
end if
if v is a dual internal node **then**
  for each outgoing edge (v, w) **do**
    if the region corresponding to (v, w) intersects with the hyperplane p^* **then**
      Add p to the subset P_w
      TRAVERSE (p, w)
    end if
  end for
end if
if v is a leaf **then**
  Add p to the subset P_v
end if

2.2.2 Query Processing in the Partition Graph Model

To answer a simplex query, we use almost the same algorithm we used for preprocessing. For a simplex q we perform a depth-first search of the partition graph, using the query regions to determine which edges to traverse. At each primal node we check for the intersection of the simplex q with the query region to decide which query regions to traverse. At a dual node, we traverse the query regions which intersect with the dual of the simplex

q. For hyperplane emptiness Erickson checked whether the query hyperplane (or dual of the hyperplane) intersected with a query region.

Algorithm 3 QUERY (q)

Let r be the root of the partition tree.
TRAVERSEQUERY (q, r)

Algorithm 4 TRAVERSEQUERY (q, v)

```
if v is a primal internal node then
  for each outgoing edge (v, w) do
    if the region corresponding to (v, w) is contained in q then
      Examine the subset P_w
    else if the region corresponding to (v, w) overlaps q then
      TRAVERSEQUERY (q, w)
    end if
  end for
end if
if v is a dual internal node then
  for each outgoing edge (v, w) do
    if the region corresponding to (v, w) contains the dual q* then
      Examine the subset P_w
    else if the region corresponding to (v, w) overlaps the dual q* then
      TRAVERSEQUERY (q, w)
    end if
  end for
end if
if v is a leaf then
  Examine the subset P_v
end if
```

A further modification is required for the case when the query region is completely contained in the query simplex q. In that case query algorithm examines the subset stored at the node, however, the node is not traversed any further. Whenever the query algorithm reaches a leaf, it examines the corresponding leaf subset. The output of the query algorithm is computed from the examined subsets (associated with both the leaves and the internal nodes). While processing a query, we are not allowed to examine a subset that contains some point not inside the query simplex. For example, a query simplex is empty iff all the

examined subsets are empty. The output of a counting query is the size of the union of the examined subsets. The output of a reporting query is the union of the examined subsets. See Algorithm 3.The runtime for the query is equal to the number of edges traversed. Since we can perform any computation on the points in the examined subsets for free—the only charge is for traversing an edge of the partition graph, the same query algorithm can be used to answer optimization variants of simplex range searching such as line-nearest neighbor.

2.2.3 Why is the Partition Graph Model Weaker?

There are two restrictions in the partition graph model that makes it weak enough so that non-trivial lower bounds on range emptiness can be established in this model:

1. An edge is traversed iff the query range overlaps with the geometric region associated with the edge.

2. The query processing algorithm is not allowed to examine subsets that contain some point not present in the query range.

2.2.4 Upper Bounds in the Partition Graph Model

The upper bounds for the simplex range searching and various other problems described in this chapter hold in the polyhedral partition graph model, since Matoušek's simplicial partitioning data structure can be implemented in this model (c.f. Section 4.5 of [39]). Very recently, Chan [23] showed that optimal simplicial partition trees can be in constructed in $O(n \operatorname{polylog} n)$ expected time—satisfying a restriction on preprocessing that Erickson utilized to prove quasi-optimal lower bounds on hyperplane emptiness.

Here we briefly describe how to convert a data structure DS_1 for simplex emptiness for n points S in $\mathbb{R}^d$ in the RAM model to a corresponding data structure DS_2 in the polyhedral partition graph model. Assume the data structure DS_1 is the optimal simplicial partition tree produced by Chan's randomized algorithm. Hence DS_1 is a tree with: a fixed root node, constant branching factor (outdegree), disjoint partitions (query regions), $O(n)$ space, $O(n \lg n)$ preprocessing, and $O(n^{1-\frac{1}{d}})$ query time with high probability. Each node in DS_1 is associated with a simplex, and stores all points contained in the simplex. For

a query simplex q, the emptiness query is answered by starting at the root of DS_1, and recursing into a child node iff q intersects with the simplex associated with the child node. In case the query simplex q contains the simplex associated with the child node, the point set stored at the child node is examined but there is no further recursion. The query simplex q is empty iff all the examined subsets are empty.

For the data structure DS_2 in the polyhedral partition graph model, use the same partitioning of the space as in DS_1. In the partition graph model computing this partitioning is free of charge. For each internal node of the partition graph DS_2, create a constant number of *dummy* outgoing edges whose corresponding regions (again simplicies) cover the space not covered by the original partition tree DS_1. The outdegree of DS_2 remains constant, thus satisfying condition 1 for a valid data structure in the partition graph model. Each dummy edge has a connected region (simplex) associated with it, and now the regions at each internal node cover the entire space $\mathbb{R}^d$, hence satisfying the conditions 2 and 3. Since the query regions associated with each outgoing edge in this partition graph is a a polyhedron, the data structure exists in the polyhedral partition graph model. The size of DS_2, measured as the number of edges, is $O(n)$ because we only added constant number of edges per each node of DS_1.

Now that we have a valid partition graph DS_2, preprocess each point p in S using the Algorithm 1. Each point gets stored at various internal nodes and a leaf node (DS_2 is a tree). Notice that during the preprocessing no point will ever end up at a dummy node. Thus storage of points in both DS_1 and DS_2 is exactly the same. The preprocessing for DS_2 is measured as the total number of edges traversed by all points in S. Since each point is stored at every node it visits (via a unique edge since DS_2 is a tree), the cost of preprocessing can be charged against the total number of points stored at various nodes. Since the space for DS_1 is $O(n)$, which includes the storage for points, the preprocessing for DS_2 in the partition graph model is only $O(n)$. To answer a simplex emptiness query for the query simplex q run the Algorithm QUERY. The simplex q is empty iff all the examined subsets are empty. The query time, measured as the number of edges traversed, is the same for both DS_1 and DS_2.

2.3 Previous Lower Bounds in the Partition Graph Model

2.3.1 Lower Bounds for Hyperplane Emptiness

Erickson [40] proved that any lower bound for halfspace range counting queries in the semigroup model implies the same lower bound for hyperplane emptiness in the partition graph model. Brönimann *et al.*[21] showed that a data structure of size m must spend $\Omega((\frac{n}{\lg n})^{\frac{(d^2+1)}{(d^2+d)}} \cdot \frac{1}{m^{\frac{1}{d}}})$ time to answer a halfspace counting query. Very recently, Arya *et al.* [12] improved this lower bound to $\Omega((\frac{n}{\lg n})^{\frac{d}{(d+1)}} \cdot \frac{1}{m^{\frac{1}{d}}})$. Both lower bound constructions [12, 21] use n independent uniformly distributed random points in a unit cube $[0,1]^d$. Hence, these lower bounds also hold with high probability for randomly generated point sets (i.e., the lower bounds hold on the *average*, not only in the worst case).

Erickson deduced lower bounds for hyperplane emptiness queries in the partition graph model from lower bounds on Hopcroft's problem. We can think of hyperplane emptiness as the online version of Hopcroft's problem. In Hopcroft's problem, for a given set of points and hyperplanes, we need to determine whether there is any incidence between a point and a hyperplane. The bounds on Hopcroft's problem are optimal up to polylogarithmic factors in the plane, and have been slightly improved by Braß and Knauer [20] for $d > 2$. Better lower bounds exist for the weaker versions of the partition graph model. In particular, if the preprocessing time is restricted to $O(n \operatorname{polylog} n)$, then a data structure must spend $\Omega(\frac{n^{1-\frac{1}{d}}}{\operatorname{polylog} n})$ time on answering a hyperplane emptiness query in the general partition graph model for $d = 2, 3$, and in the polyhedral partition graph model in dimensions $d \geq 4$. These bounds are optimal up to polylogarithmic factors. The lower bound constructions [20, 40] use a deterministic point set, a section of the integer lattice $\mathbb{Z}^d$, thus these bounds do not necessarily hold on the average.

2.3.2 Lower Bounds for Simplex Emptiness

The best currently known lower bounds for (non-degenerate) simplex emptiness result from a simple reduction [17] from halfspace emptiness in the partition graph model. The lower bounds on halfspace emptiness follow from reducing hyperplane emptiness in $\mathbb{R}^\delta$,

$d = \frac{\delta(\delta+3)}{2}$ to halfspace emptiness in $\mathbb{R}^d$ by a standard lifting argument, but only in the semi-algebraic partition graph model [40]. For dimension $2 \leq d \leq 4$, only trivial lower bounds are known: any data structure for halfspace emptiness using near-linear space ($O(n^{1+\varepsilon})$ space) must spend $\Omega(\lg n)$ time answering a query.

Table 2.3.2 provides a compendium of the relevant range searching lower bounds in three distinct models of computation: the semigroup arithmetic model, the pointer machine model, and the partition graph model.

Problem	**Model**	**Query Time**	**Source**
Counting	Semigroup arithmetic $(d = 2, m = n)$	$\Omega(\sqrt{n})$	[27]
Counting	Semigroup arithmetic $(d > 2)$	$\Omega(\frac{n}{m^{\frac{1}{d}} \lg n})$	[27]
Reporting	Pointer machine $(d = 2, m = n)$	$\Omega(n^{\frac{1}{2}-\varepsilon} + r)$	[34]
Reporting	Pointer machine	$\Omega(\frac{n^{1-\varepsilon}}{m^{\frac{1}{d}}} + r)$	[34]
Emptiness	Semialgebraic partition graph $(d = 5, m = n)$	$\Omega(n^{\frac{1}{3}})$ (average case)	[17, 40]
Emptiness	Semialgebraic partition graph $(d \geq \frac{\delta(\delta+3)}{2})$	$\Omega((\frac{n}{\lg n})^{\frac{\delta}{\delta+1}} \cdot \frac{1}{m^{\frac{1}{(\delta+1)}}})$ (average case)	[12, 17, 40]
Emptiness	Polyhedral partition graph *prep* = $O(n \operatorname{polylog} n)$ $(d \geq \frac{\delta(\delta+3)}{2})$	$\Omega(\frac{n^{1-\frac{1}{\delta}}}{\operatorname{polylog} n})$ (worst case only)	[17, 40]
Emptiness	Semialgebraic partition graph $(d = 2, m = n)$	$\Omega(n^{\frac{1}{3}})$ (average case)	new
Emptiness	Semialgebraic partition graph $(d > 2)$	$\Omega((\frac{n}{\lg n})^{\frac{d}{d+1}} \cdot \frac{1}{m^{\frac{1}{(d+1)}}})$ (average case)	new
Emptiness	Polyhedral partition graph *prep* = $O(n \operatorname{polylog} n)$	$\Omega(\frac{n^{1-\frac{1}{d}}}{\operatorname{polylog} n})$ (worst case only)	new

Table 2.1: Lower bounds for online simplex range searching using m space and *prep* preprocessing.

2.4 Results

We establish new lower bounds for the simplex emptiness queries in $\mathbb{R}^d$ in the partition graph model.

- We reduce hyperplane emptiness to simplex emptiness. A data structure for simplex emptiness in the polyhedral partition graph model must spend $\Omega(\frac{n^{1-\frac{1}{d}}}{\text{polylog}\, n})$ time answering a query when the preprocessing is restricted to be $O(n\ \text{polylog}\, n)$. These lower bounds are quasi-optimal, and hold in the worst case. We also prove weaker lower bounds in the general partition graph model, with no restriction on preprocessing, that hold in the average case. (Section 2.6)

  Previous lower bounds for simplex emptiness were rather weak, derived from halfspace emptiness queries through a lifting method for $d \geq 5$, and only trivial lower bounds were known for dimensions $2 \leq d \leq 4$.

- Our lower bounds imply lower bounds for the simplex range reporting in the polyhedral partition graph model irrespective of the number of points being reported. (Section 2.6)

  Previous lower bounds hold only for queries containing at least n^δ points for some $\delta > 0$.

- We reduce simplex emptiness to various other problems in computational geometry, such as segment intersection searching, implicit point location, line-nearest neighbor, segment dragging, halfplane proximity, halfplane convex hull queries, and point-inclusion in a union of slabs, see Fig. 2.1.

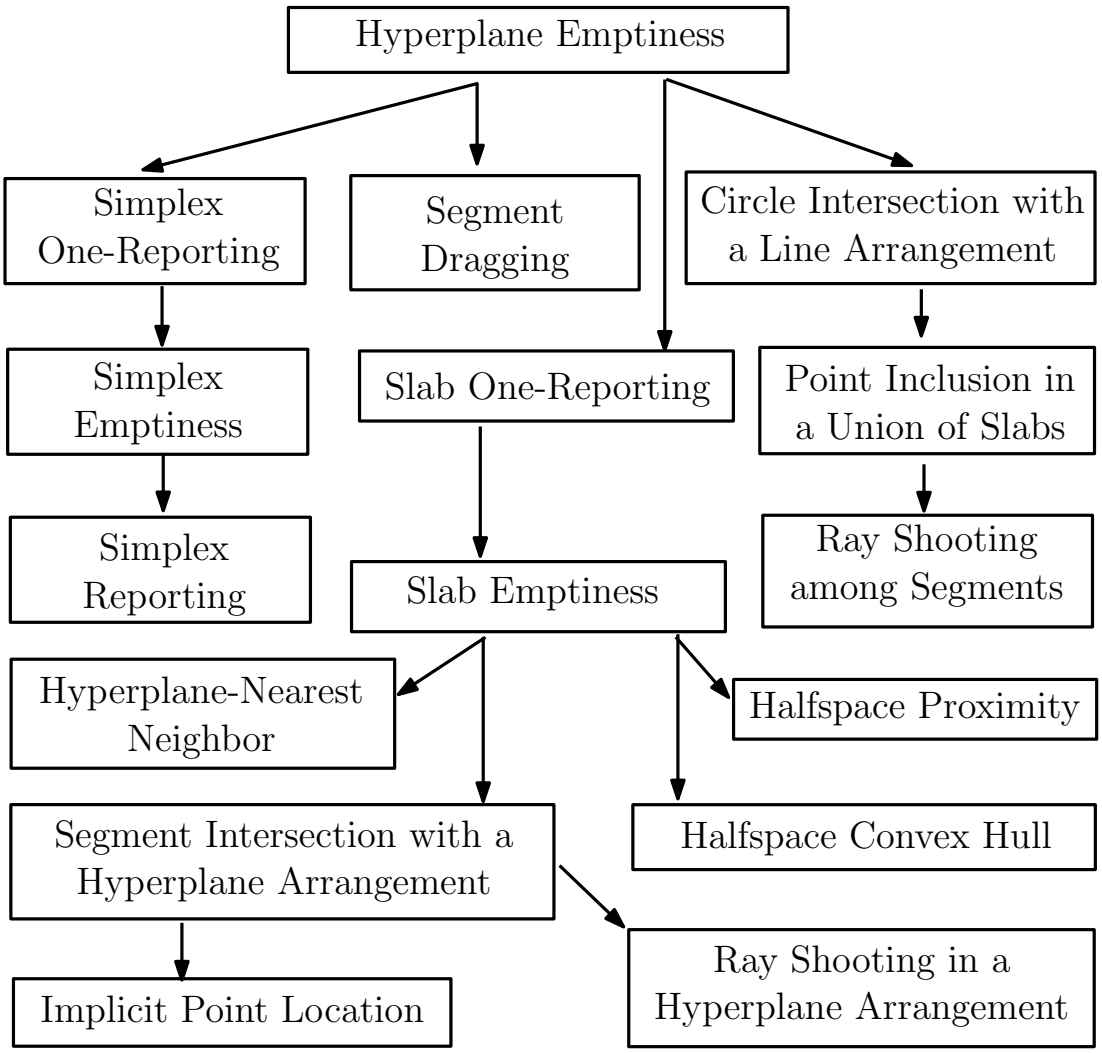


Figure 2.1: Reductions from hyperplane emptiness to simplex emptiness and other related problems. Arrows show the direction of reduction.

2.5 Preliminaries

In this section, we prove the key properties of the two point sets used for our lower bound constructions.

2.5.1 Deterministic Point Set on a Regular Lattice

Let $L_{d,n}$ be a set of n points in the integer lattice $\mathbb{Z}^d$ lying in the cube $[0, \lceil n^{\frac{1}{d}} \rceil]^d$ (Fig. 2.2). Erdős observed (c.f., [38]) that for every $m, n \in \mathbb{N}$ such that $\sqrt{n} \leq m \leq n^2$, there are m lines in the plane that determine $\Omega(n^{\frac{2}{3}} m^{\frac{2}{3}})$ point-line incidences with the point set $L_{2,n}$. This bound on incidences is best possible by the Szemerédi-Trotter theorem [69]. This construction led to lower bounds [39] for Hopcroft's problem in the plane—given n points and m lines, is there any incidence? Erickson [40] then proved lower bounds for the online version of Hopcroft's problem (hyperplane emptiness) in $\mathbb{R}^d$, using the point set $L_{d,n}$, by restricting the amount of preprocessing.

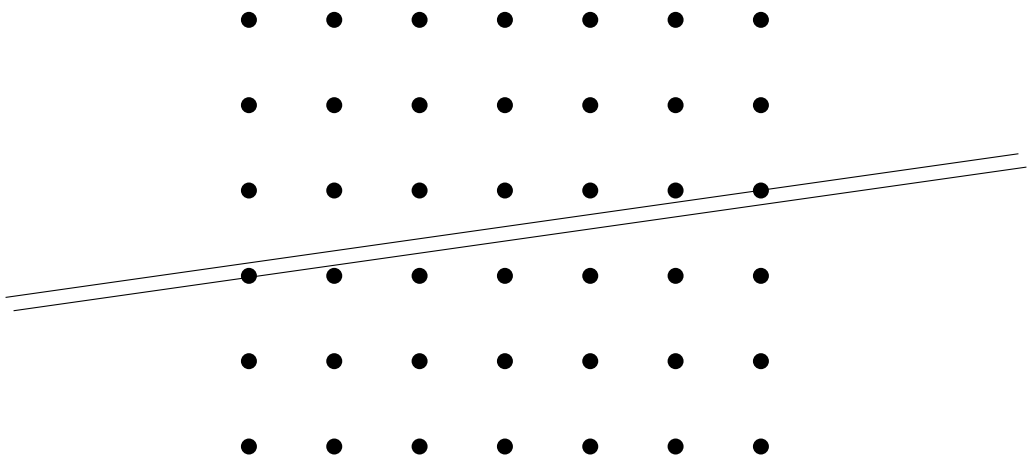


Figure 2.2: Point set on a regular $\sqrt{n} \times \sqrt{n}$ lattice—Erdős' construction.

Here we prove that points from a section of the integer lattice $\mathbb{Z}^d$ must lie on a hyperplane if they are contained in a sufficiently thin *slab*, which is a region between two distinct parallel hyperplanes in $\mathbb{R}^d$.

Lemma 2.1 *For every $d > 1$, there is a constant $c_d > 0$ such that all points in $L_{d,n}$ in a slab of width at most $c_d n^{\frac{(1-d)}{d}}$ lie on a hyperplane.*

Proof. The minimum positive volume of a full dimensional simplex in the integer lattice is $\frac{1}{d!}$. The volume of a simplex $\{p_0, p_1, \ldots, p_d\} \subset \mathbb{R}^d$ is $\frac{|\det A|}{d!}$, where A is an $d \times d$ matrix whose columns are the vectors $p_0 p_i$, $i = 1, 2, \ldots d$. If A is an integer matrix, then its determinant is an integer. Hence, every convex region that contains $d+1$ affine independent

points of $L_{d,n}$ must have volume at least $\frac{1}{d!}$. However, the volume of the intersection of a cube of side length $\lceil n^{\frac{1}{d}} \rceil$ and a slab of width $c_d n^{\frac{(1-d)}{d}}$ is at most $c_d \Delta^{d-1} n^{\frac{(1-d)}{d}}$, where $\Delta = \sqrt{d} \cdot \lceil n^{\frac{1}{d}} \rceil$ is the diameter of the cube. This is less than $\frac{1}{d!}$ if $c_d < \frac{1}{(2d^{\frac{(d-1)}{2}} d!)}$. □

2.5.2 Randomly Generated Set of Points in a Unit Cube

Let $P_{d,n}$ be a set of n points chosen independently and uniformly at random from the d-dimensional unit cube $[0,1]^d$. Chazelle [27] proved that the convex hull of any k points of $P_{d,n}$ has $\Omega(\frac{k}{n})$ volume with high probability if $k \gg \lg n$.

Theorem 2.1 ([27]) *For any $d \geq 2$, there exists a constant $c_d > 0$ such that the convex hull of any $k \gg \lg n$ points of $P_{d,n}$ has volume greater than $\frac{c_d k}{n}$ with probability greater than $1 - \frac{1}{n}$.*

Chazelle (along with his coauthors) used the point set $P_{d,n}$ to establish lower bounds for the simplex semigroup queries [27], simplex range reporting [34], and halfspace range counting [11, 21]. Erickson [40] showed (weaker) lower bounds for hyperplane emptiness via reduction from halfspace range counting using the same point set. Here we show that the lower bounds for hyperplane emptiness continue to hold even if the points have the following two additional properties: the first coordinates of any two points differ by $\Omega(n^{-3})$, and every point is at distance at least $\Omega(n^{-d(d+2)})$ from all hyperplanes spanned by d other points in $P_{d,n}$. Conveniently, the random point set $P_{d,n}$ has *all* these properties with high probability.

Lemma 2.2 *For every $d \geq 1$, the first coordinates of any two points in $P_{d,n}$ differ by at least $\frac{1}{n^3}$ with probability greater than $1 - \frac{1}{n}$.*

Proof. Let $\{p_1, p_2, \ldots, p_n\}$ denote the first coordinates of the points in $P_{d,n}$. These are mutually independent random variables with uniform distribution over the unit interval $[0,1]$. For $1 \leq 1 < j \leq n$, we have $|p_i - p_j| < \frac{1}{n^3}$ if p_j lies in the interval of length $\frac{2}{n^3}$ centered at p_i. Hence, the probability that $|p_i - p_j| < \frac{1}{n^3}$ is at most $\frac{2}{n^3}$. The probability that there are p_i and p_j, $i < j$, with $|p_i - p_j| < \frac{1}{n^3}$ is at most $\sum_{i<j} \frac{2}{n^3} = \binom{n}{2} \frac{2}{n^3} = \frac{n-1}{n^2} < \frac{1}{n}$. □

Lemma 2.3 *For every $d > 1$, there is a constant $c_d > 0$ such that* all *points of $P_{d,n}$ in a slab of width at most $c_d n^{-d(d+2)}$ must lie on a hyperplane, with probability $1 - \frac{1}{n}$.*

Proof. The diameter of the unit cube $[0,1]^d$ is $\sqrt{d}$. Hence, the intersection of a the unit cube and a slab of width w has volume less than $d^{\frac{(d-1)}{2}} w$. We show that the volume of every simplex determined by $d+1$ points in $P_{d,n}$ is at least $d^{\frac{(d-1)}{2}} c_d n^{-d(d+2)}$ with probability $1 - \frac{1}{n}$, and so it cannot be contained in a thin slab.

Note that any d points in $P_{d,n}$ are affinely independent and so span a hyperplane, with probability 1. Let $\alpha > 0$ be a small constant that we specify shortly. The probability that a random point is within distance $\alpha n^{-(d+2)}$ from a given hyperplane is less than $2\alpha d^{\frac{(d-1)}{2}} n^{-(d+2)}$. Hence, the probability that some point in $P_{d,n}$ lies in the $n^{-(d+2)}$-neighborhood of some hyperplane spanned by d other points in $P_{d,n}$ is less than $n\binom{n-1}{d} \cdot 2\alpha d^{\frac{(d-1)}{2}} n^{-d+2}$, which is less than $\frac{1}{n}$ if $\alpha > 0$ is a sufficiently small constant.

If every vertex of a d-dimensional simplex is at distance at least $\alpha n^{-(d+2)}$ from the hyperplane of the opposite face (*i.e.*, every *height* is at least $\alpha n^{-(d+2)}$), then its volume is at least $(\alpha n^{-(d+2)})^{\frac{d}{d!}}$. This is at least $c_d d^{\frac{(d-1)}{2}} n^{-d(d+2)}$ if $c_d = \frac{\alpha^d}{d!}$. □

Theorem 2.2 *For any $d > 1$, a random point set $P_{d,n}$ has, with probability greater than $1 - \frac{3}{n}$, the properties that*

(A) *the convex hull of any $k \gg \lg n$ points has volume greater than $\Omega(\frac{k}{n})$,*

(B) *first coordinates of any two points differ by at least $\frac{1}{n^3}$,*

(C) *there exists a width $w = \Omega(n^{-d(d+2)})$ such that all points in a slab of width at most w lie on a hyperplane.*

Proof. By Theorem 2.1, Lemma 2.2 and Lemma 2.3, a random point set $P_{d,n}$ has property (A), (B) and (C), respectively, with probability at least $1 - \frac{1}{n}$. Hence, it has all three properties with probability at least $1 - \frac{3}{n}$. □

2.6 Lower Bounds for Simplex Emptiness

In this section, we establish the lower bounds for both simplex and slab emptiness in the partition graph model. First we reduce hyperplane emptiness to simplex one-reporting.

That is, we are given a simplex one-reporting data structure (a black box), and we answer a hyperplane emptiness query using the information obtained from a carefully designed sequence of simplex one-reporting queries. Then we reduce simplex one-reporting to simplex emptiness. Our reduction arguments work for both the deterministic and the random point sets $L_{d,n}$ and $P_{d,n}$.

1. Given a data structure $DS_{1\mathrm{R}}$ for simplex one-reporting that answers "empty" if the simplex is empty and reports an arbitrary point (witness) inside the simplex otherwise, we show that a hyperplane emptiness query can be answered by making $O(d^2)$ queries to $DS_{1\mathrm{R}}$.

2. Given a data structure DS_{Φ} for simplex emptiness that answers "yes" whenever the simplex is empty and "no" otherwise, we show that a witness for a nonempty simplex can be reported by making $O(d \lg n)$ queries.

2.6.1 Reducing Hyperplane Emptiness to Simplex One-Reporting

Recall that $L_{d,n}$ is a section of the d-dimensional integer lattice. Let B be the bounding box of the point set. For the given query hyperplane h, create a "flat" simplex R that contains $h \cap B$ and lies in a sufficiently thin slab. Query the data structure $DS_{1\mathrm{R}}$. If the simplex R is empty, then the hyperplane is also empty. Otherwise, the data structure reports a point $p_1 \in R$. If $p_1 \in h$, which can be checked in constant time, then the hyperplane is not empty. Assume we have already found k affine independent points $\{p_1, p_2, \ldots, p_k\}$ in $R \setminus h$. If $k < d$, then let F_k be the $(k-1)$-dimensional flat spanned by them; cover $R \setminus F_k$ by a constant number of (possibly overlapping) simplicies, and query these simplicies. If all are empty, then the hyperplane is also empty. Otherwise, we find a point $p_{k+1} \in R$, which is affine independent from the first k points. If we find $k = d$ affine independent points in R, then every point in $R \cap L_{d,n}$ must lie on the hyperplane F_d by Lemma 2.1. Now the point set $L_{d,n} \cap F_d$ is an affine copy of points from the integer lattice $\mathbb{Z}^{d-1}$, and $h \cap F_d$ is a hyperplane within F_d. So we have reduced the problem to $(d-1)$ dimensions with $O(d)$ queries to $DS_{1\mathrm{R}}$. With $O(d^2)$ queries, we reduce the problem to zero dimension, where the location of a witness is limited to a single point. We can check in constant time whether this point is in $L_{d,n}$, and report the answer.

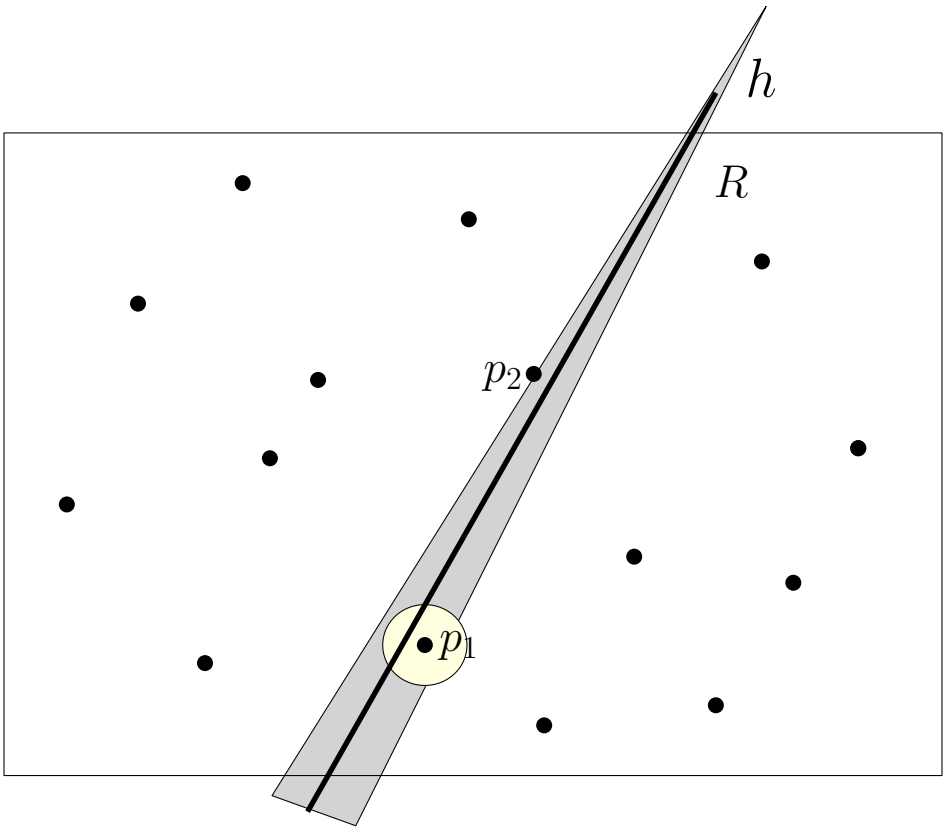


Figure 2.3: Reducing hyperplane emptiness to simplex one-reporting.

The reduction is somewhat simpler for the random point set $P_{d,n}$. If there are d affine independent points that lie in $R \setminus h$, then h must be empty (Lemma 2.3). Induction on the dimension is not necessary, and so $O(d)$ queries to $DS_{1\mathrm{R}}$ suffice.

2.6.2 Reducing Simplex One-Reporting to Simplex Emptiness

The hard part of answering a simplex one-reporting query lies in deciding whether or not the simplex is empty, rather than in reporting of a witness. We are given a data structure DS_Φ and a query simplex R. If the simplex is empty, then there is nothing to report. Otherwise, partition the point set in two equal halves along an axis-aligned hyperplane. Cover the parts of R in each half-space by a constant number of (possibly overlapping) smaller simplicies. Query all the smaller simplicies, and recurse on one of the nonempty

simplicies. After $O(\lg n)$ queries, the diameter of simplex drops below 1, and then we can report the unique lattice point that resides in the simplex in constant time.

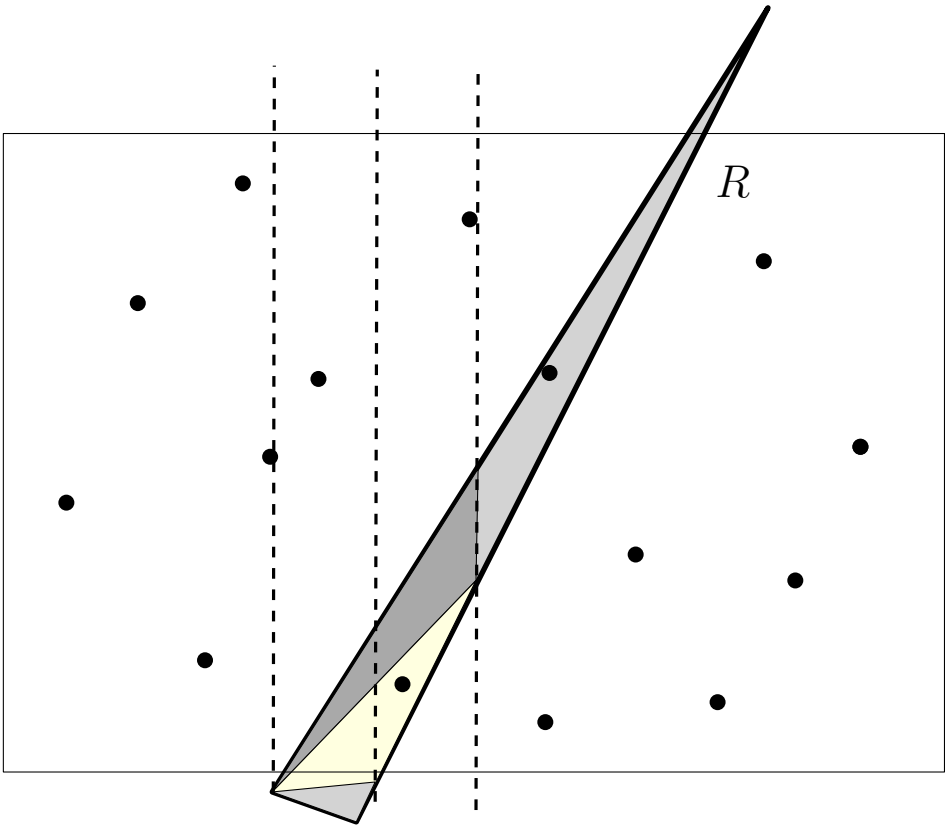


Figure 2.4: Reducing simplex one-reporting to simplex emptiness.

For a random point set, it is crucial that the underlying point set S has property (B), that is, the first coordinates of any two points differ by at least $\frac{1}{n^3}$. Repeat the above argument with the restriction that every halving hyperplane is orthogonal to the x_1-axis. (Note that a hyperplane contains $n^{\frac{(d-1)}{d}}$ point of $L_{d,n}$, so we used hyperplanes of different orientations for partitioning $L_{d,n}$.) Stop when the width between two parallel hyperplanes drops below $\frac{1}{2n^3}$. Since S satisfies property (B), there is a unique point that lies in a nonempty simplex, which we can find by simple binary search over the first coordinates of the points.

Remark. Since any data structure for simplex reporting can answer simplex emptiness queries, the lower bounds for simplex emptiness also hold for simplex reporting.

2.6.3 Reducing Hyperplane Emptiness to Slab Emptiness

We conclude this section by reducing hyperplane emptiness to slab one-reporting, which in turn can be reduced to slab emptiness. Note that a query about a slab carries less useful information than a simplex query, hence the reduction is more involved. We first present a reduction that uses slabs of constant width. This reduction works for the planar lattice $L_{2,n}$ and for random point sets $P_{d.n}$ for all $d \geq 2$. Then we present a refined reduction with varying slab widths that works for lattices $L_{d,n}$ in all dimensions.

We are given a set of n points in a bounding box $B \subset \mathbb{R}^d$, and want to answer an emptiness query for a hyperplane h. Construct a d-dimensional range tree [18] on the point set. Each node of the range tree corresponds to a subset of points clipped in an axis-aligned box. For each of these subsets, construct a slab one-reporting data structure. The range tree data structure costs an extra factor of $O(\lg^{d-1} n)$ space.

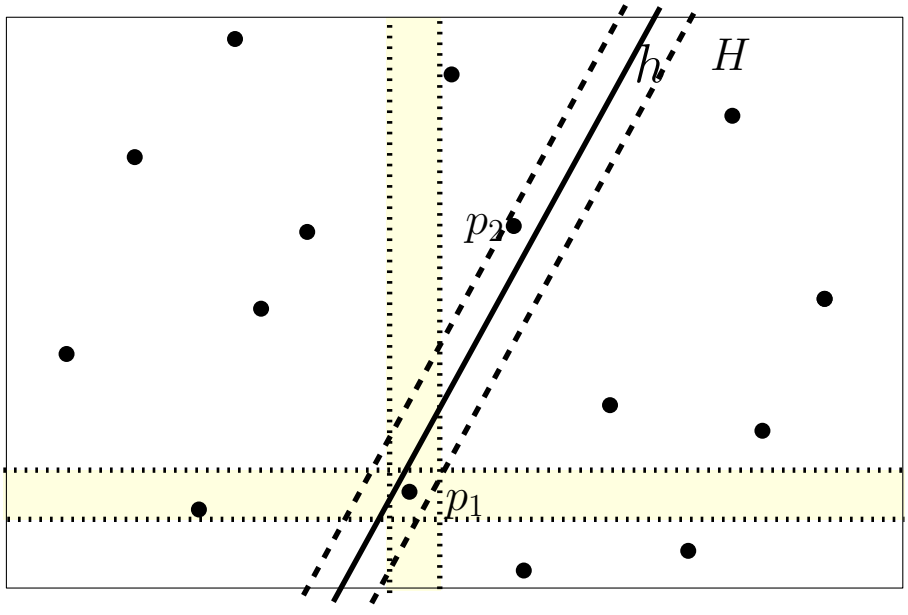


Figure 2.5: Reducing hyperplane emptiness to slab emptiness

For a query hyperplane h, create a sufficiently thin slab H containing $h \cap B$. Query the slab one-reporting data structure at the root of the range tree. If the slab H is empty, then there is nothing to report. Otherwise, the data structure reports a point $p \in H$. Assume we have already found k distinct points $\{p_1, p_2, \ldots, p_k\}$ in $H \setminus h$. If $k < d$, then we can

cover $H \setminus \{p_1, p_2, \ldots, p_k\}$ by $O(\lg n)$ axis-aligned boxes for which we maintain a slab one-reporting data structure. Query H for all $O(\lg n)$ data structures. If all return "empty", then the hyperplane is also empty. Otherwise, we find a point $p_{k+1} \in H$, which is distinct from the first k points. If we find $k = d$ distinct points in H, then every point in $H \cap P_{d,n}$ must lie on the hyperplane spanned by $\{p_1, \ldots, p_d\}$ (c.f., Lemma 2.3), hence h is empty.

In case of the section of the integer lattice $L_{d,n}$, we cannot guarantee that the points $\{p_1, \ldots, p_k\}$ are affine independent when $d \geq 3$. Two distinct points are automatically affine independent, and so for $d = 2$ our argument works analogously to reduction for simplex one-reporting.

2.6.4 Reducing Hyperplane Emptiness to Slab Emptiness for $L_{d,n}$ $d \geq 3$

Recall that $L_{d,n}$ is a set of n points in the $[0, \lceil n^{\frac{1}{d}} \rceil]$ section of the integer lattice $\mathbb{Z}^d$. We reduce hyperplane emptiness to slab one-reporting, which in turn reduces to slab emptiness (using range trees as in Section 2.6.3). Recall that a data structure DS_{1RS} for slab one-reporting (for slabs of arbitrary widths) answers "empty" if the slab is empty and reports an arbitrary point (witness) inside the query slab otherwise.

For a query hyperplane h, create a thin slab H_0 containing h and of width $c_d n^{\frac{(1-d)}{d}}$, where c_d is the constant from Lemma 2.1. Query the data structure DS_{1RS}. If the slab H_0 is empty, then hyperplane h is also empty. Otherwise, the data structure reports a point $p_1 \in H$. If $p_1 \in h$, then the hyperplane is not empty.

Assume that we have already found k affine independent points $\{p_1, p_2, \ldots, p_k\}$ in $H_0 \setminus h$. If $k < d$, then let F_k be the $(k-1)$-dimensional flat spanned by them. Based on an orthogonal basis of F_k, compute a width w_k such that (i) any slab of width w_k parallel to h contains at most one (the closest) point form $L_{d,n}$; and (ii) w_k is smaller than the distance between h and $\{p_1, \ldots, p_k\}$. Query DS_{1RS} for a slab $H_k \subset H_0$ of width w_k containing h. If H_k is empty, then h is empty, otherwise, we obtain a new point $q \in H_k \subset H_0$. If $q \in h$, then the hyperplane is not empty. If $q \notin h$ but $q \in F_k$, then q is the closest point to h in $F_k \cap L_{d,n}$ (by the construction of H_k), and then we can replace p_k with q, and recompute w_k. Finally, if $q \notin h$ and $q \notin F_k$, then $p_{k+1} = q$ is a new affine independent point lying in $H_0 \setminus h$. If we find $k = d$ affine independent points in H, then every point in $H \cap L_{d,n}$

must lie on the hyperplane F_d by Lemma 2.1. From here, we can continue exactly as in our reduction to simplex one-reporting (Section 2.6.1).

2.7 Lower Bounds for Related Problems

We show that the above lower bounds for simplex emptiness queries also apply to several other problems in computational geometry. We present the problems for $d = 2$ (that is, in the plane) only, but the lower bounds extend to their d-dimensional variants. For the planar variants of these problems, there are data structures with $n \leq m \leq n^2$ space and $O(\frac{n}{\sqrt{m}} \operatorname{polylog}(n))$ query time. The data structures are based on the simplicial partition trees, which can be implemented in the partition graph model (see Section 2.2.4). The lower bounds are all optimal up to polylogarithmic factors, and they are shown by reduction from slab emptiness.

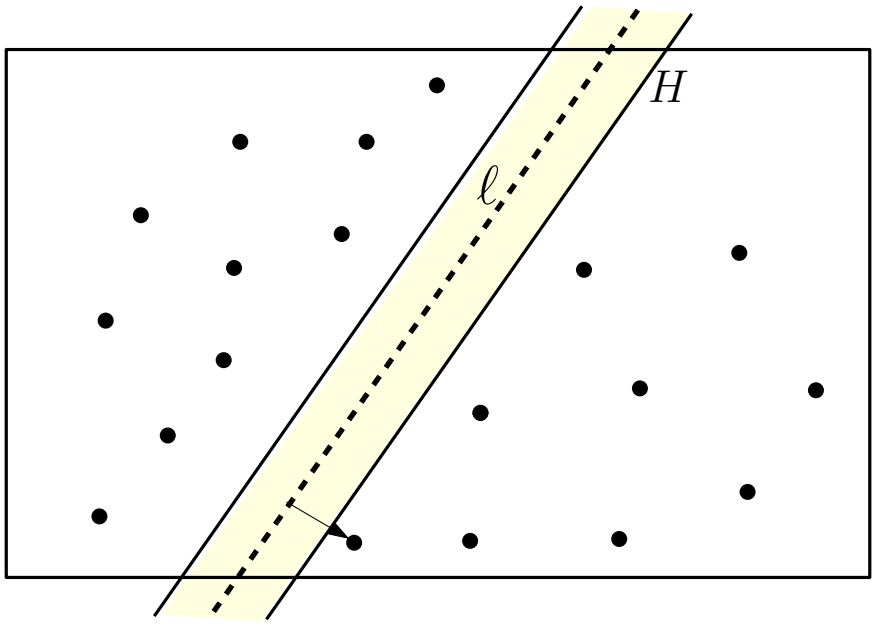


Figure 2.6: Reduction from slab emptiness to line-nearest neighbor queries.

2.7.1 Line-Nearest Neighbor

Given a set of n points in the plane, a *line-nearest neighbor query* asks for the closest point to a query line ℓ. We show that slab emptiness reduces to line-nearest neighbor queries. Assume we have a data structure that supports line-nearest neighbor queries, and we want to tell whether a slab H is empty or not. It is enough to query the line ℓ, which is is parallel to H and partitions H into two congruent slabs. The slab H is empty iff the nearest point to ℓ lies outside of H, which can be verified in constant time.

2.7.2 Segment Intersection with an Arrangements of Lines

The standard point-line duality is a bijection between a point $p(a, b)$ and a nonvertical line ℓ in the plane via $p(a, b) \leftrightarrow \ell : y = ax + b$. The dual of a set of points is an arrangement of nonvertical lines. A nonvertical slab is the union of parallel lines, whose duals are points along a vertical line segment. A nonvertical slab is empty iff the dual segment does not intersect any line. It follows that our lower bounds directly apply to the *vertical segment intersection query*, which asks whether a vertical query segment intersects any line in a given set of n lines in the plane. Clearly, our lower bounds also apply to the more general segment intersection query, in which the query segment does not have to be vertical.

Segment intersection, in turn, reduces to *point location queries* in an arrangement of lines, since a segment intersects a line iff its two endpoints are in distinct faces (recall that all faces are convex).

Similarly, it also reduces to *ray shooting queries* in an arrangement of lines: a segment intersects a line iff the ray shot from one endpoint along the segment hits a line before reaching the other endpoint. Notice that for a seemingly similar problem of ray intersection in an arrangement of lines, there is a data structure with $O(\text{polylog}\, n)$ query time using near-linear space [17].

2.7.3 Segment Dragging

Given a set of polygonal objects in the plane, a *segment dragging query* [2] asks for the first object hit if a query segment e moves (is dragged) in a query direction ρ. A segment

dragging query reduces to a *parallelogram emptiness* query. Hyperplane emptiness reduces to parallelogram emptiness exactly as it reduced to slab emptiness (c.f., Section 2.6.3). The obstacles are points in the plane. A query parallelogram R is empty iff when dragging one side of R towards the opposite side, the first point hit lies outside of R. The best known data structure supports segment dragging queries in $O(\frac{n}{\sqrt{m}}\ \mathrm{polylog}\, n)$ time using $O(m)$ space [2] in the RAM model. Chazelle [25] gave an $O(n)$-size data structure in the RAM model with $O(\lg n)$ query time for the special case of horizontal segments and vertical directions. Segment dragging has applications in computer graphics, motion planning and manufacturing [2, 25].

2.7.4 Implicit Point location

Agarwal and Kreveld [6] considered the problem of implicit point location for a set of n (possibly intersecting) line segments in the plane. A data structure implicitly stores the arrangement of segments such that for a query segment e, it can detect whether the two endpoints of e belong to the same face of the arrangement. They gave a data structure for implicit point location with $O(n \lg^2 n)$ space and $O(\sqrt{n} \lg^2 n)$ query time in the RAM model.

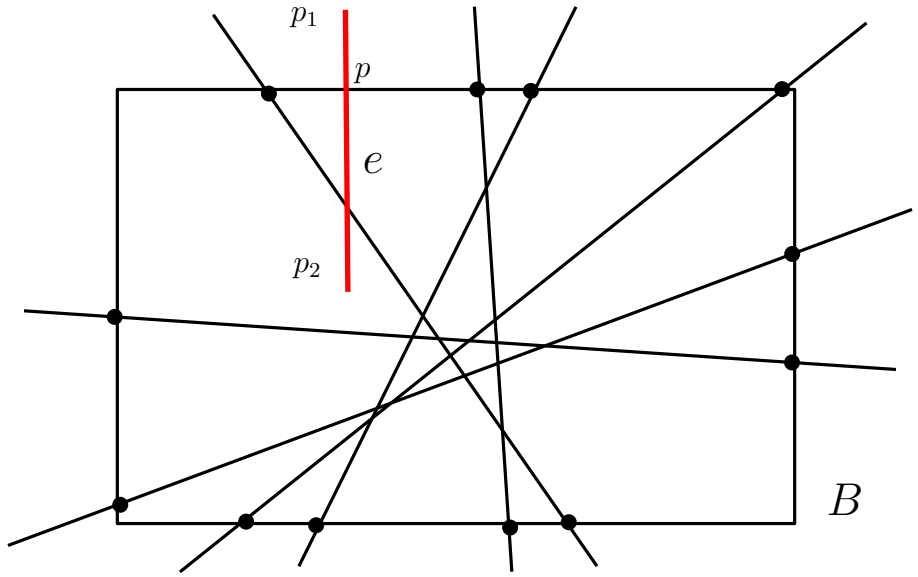


Figure 2.7: Reduction from segment intersection to implicit point location.

We establish lower bounds by reducing vertical segment intersection with an arrangement of lines to implicit point location. We preprocess an arrangement of n lines as follows. Assume that no line is vertical (by a rotation if necessary) and not all are parallel. Let B be the bounding box of all intersection points among the lines. Build an implicit point location data structure DS_{implicit} on the parts of the lines clipped in B, and the four edges of B. Build a linear-size explicit point location data structure DS_{explicit} with $O(\lg n)$ query time for the portion of the arrangement outside B (e.g. sort the rays by slope). Finally, build a binary search tree on the $2n$ intersection points between the box B and the lines (sorted counterclockwise along B). We can then answer a segment intersection query as described in Algorithm SEGMENTINTERSECTIONWITHARRANGEMENT. A query segment $e = p_1p_2$ inside B intersects with the arrangement of lines iff the endpoints p_1 and p_2 lie in the different faces of the arrangement, since the faces of the arrangement of lines as well as the interior of B are convex.

Algorithm 5 SEGMENTINTERSECTIONWITHARRANGEMENT (e)

```
Let p1 and p2 be the two endpoints of e
if e does not intersect B then
  if both p1 and p2 lie outside B then
    if DS_explicit reports that p1 and p2 are not in the same face then
      return "intersection"
    end if
  else if DS_implicit reports that p1 and p2 are not in the same face then
    return "intersection"
  end if
else if e intersects B at point p then
  if p is one of the 2n intersection points of B with the arrangement then
    return "intersection"
  else
    Break e into two half-open segments [p1, p) and [p2, p)
    SEGMENTINTERSECTIONWITHARRANGEMENT [p2, p)
    SEGMENTINTERSECTIONWITHARRANGEMENT [p1, p)
  end if
end if
```

2.7.5 Halfplane Convex Hull Queries

For a set S of n points in the plane and a halfplane h^+, the *halfplane convex hull* is the convex hull $\mathrm{ch}(S \cap h^+)$ of the subset of points lying in h^+. If a data structure can report the halfplane convex hull explicitly for a query halfplane h^+, then it can also answer a line nearest neighbor query in an additional $O(\lg n)$ time. Hence, our lower bounds for slab emptiness apply to such a data structure as well. Reporting $\mathrm{ch}(S \cap h^+)$ may take up to $\Omega(n)$, however. A family of *halfplane convex hull queries* asks only $O(1)$ information about the convex hull $\mathrm{ch}(S \cap h^+)$, where h^+ is part of the query. These queries include *extremal-point* query, *tangent* query, *line-stabbing* query, and *containment* query. For example, a *halfplane line-stabbing* query (h^+, ℓ) asks whether the query line ℓ intersects the convex hull $\mathrm{ch}(S \cap H^+)$.

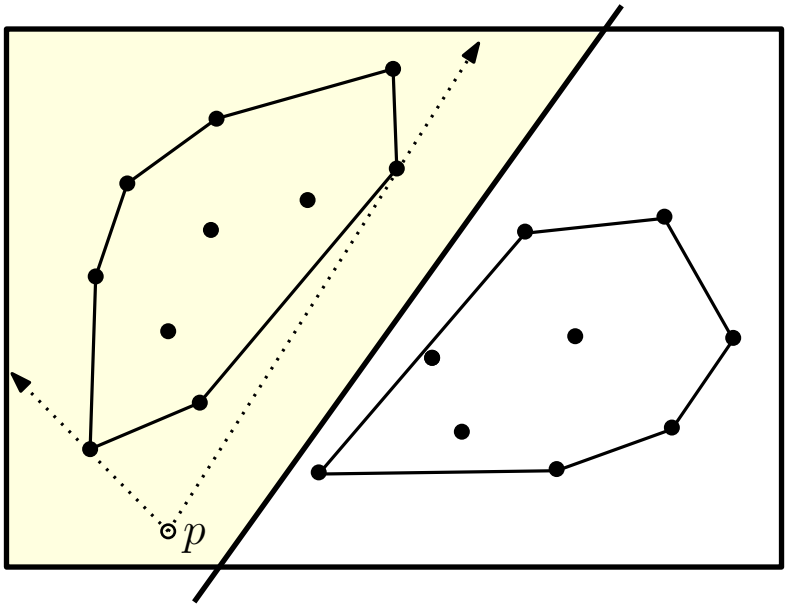


Figure 2.8: Halfplane convex hull queries.

A slab emptiness query reduces to a halfplane extremal-point query and halfplane line-stabbing query. For a slab H defined by the two parallel lines ℓ_l and ℓ_r, a slab emptiness query can be answered by finding the extremal point in the direction perpendicular to ℓ_l in the closed halfplane to the left of ℓ_r. Similarly a line-stabbing query for the line ℓ_l in the

closed halfplane to the left of ℓ_r can answer a slab emptiness query. A halfplane extremal-point query in turn reduces to halfplane tangent query. Hence, the slab emptiness lower bounds apply to halfplane extremal point, tangent and line-stabbing queries. However, finding a reduction to halfplane containment queries remains an open problem.

2.7.6 Halfplane Proximity Queries

A halfplane proximity query (h^+, p) is similar to halfplane convex hull query. It asks for the nearest/farthest neighbor of the query point p among the points in $S \cap h^+$. Halfplane proximity queries have applications in polygonal line simplification, and have been studied by Aronov *et al.* [10] and Daescu *et al.* [36].

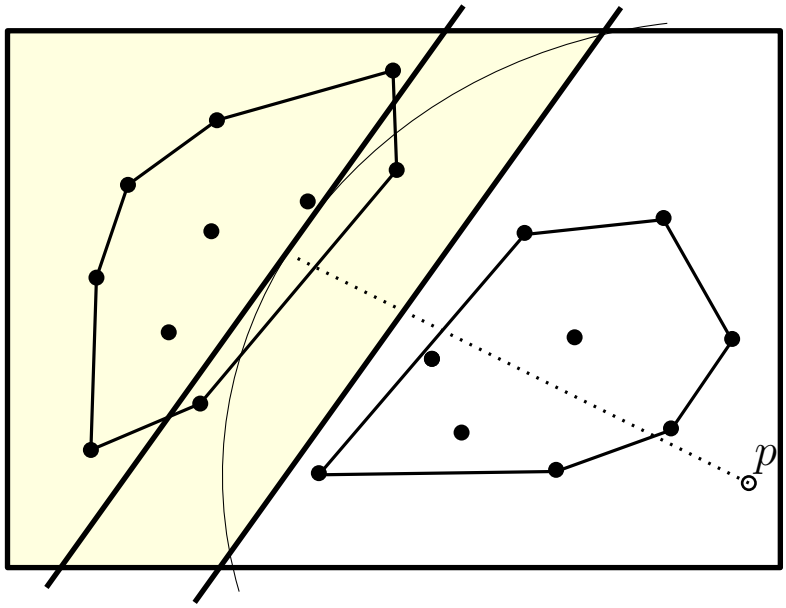


Figure 2.9: Halfplane queries proximity queries.

The lower bounds for halfplane proximity queries can be established by a reduction from slab emptiness. For a slab H defined by the two parallel lines ℓ_l and ℓ_r, a slab emptiness query can be answered by finding the nearest/farthest neighbor of a point p in the closed halfplane to the left of ℓ_r. The point p is placed sufficiently far in the direction perpendicular to ℓ_l. A point p is sufficiently far if a circle centered at p is "indistinguishable" from a line within the bounding box of the point set S. That is, within the bounding box B

of S, a line segment and a tangent circle centered at p are both contained in a slab of width $O(n^{-\frac{1}{2}})$ (*c.f.*, Lemmas 2.1 and 2.3). For nearest neighbor, the point p is to the left of ℓ_r, and to the right of ℓ_l for the farthest neighbor.

2.7.7 Point-Inclusion in a Union of Slabs

The *circle intersection* with an arrangement of lines asks whether a query circle intersects any line in the arrangement. Point-line duality maps the points of a circle to lines tangent to a hyperbola (i.e., lines whose union fills the interior of the hyperbola). Therefore, circle intersection with an arrangement of lines is equivalent to *hyperbola emptiness* for a set of points in the plane. Hyperplane emptiness reduces to hyperbola emptiness exactly as it reduced to slab emptiness (c.f., Section 2.6.3). The same argument works with hyperbolas instead of slabs, since every hyperbola contains a line (a symmetry axis), and every thin slab contains a "thin" hyperbola clipped inside the bounding box B of the point set.

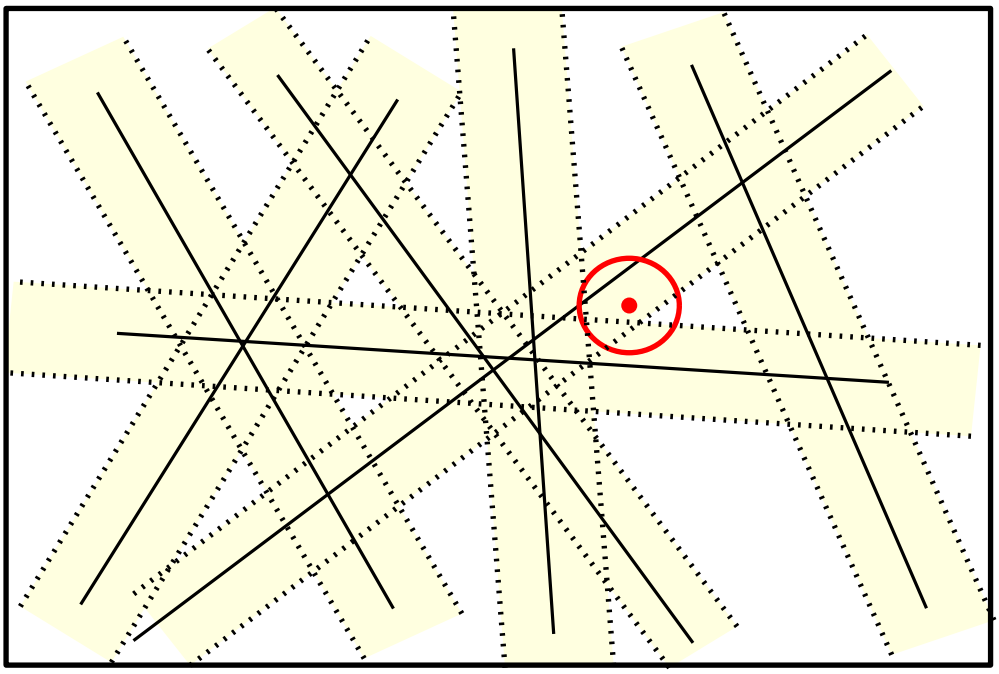


Figure 2.10: Reducing circle intersection to point-inclusion in a union of slabs.

We can also assume that *all* query circles have a fixed (but very small) radius, which means that the corresponding hyperbolas (clipped in a bounding box) all fit in an identical

thin slab. Unit circle intersection with an arrangement of lines reduces to *point inclusion in a union of slabs*. Replace each line in the arrangement by a slab of width 2. Then a unit circle intersects a line iff the center of the circle is inside some slab.

Using the point-line duality again, point-inclusion in a union of slabs is equivalent to detecting whether a query line intersects any segment in an arrangement of vertical line segments, which trivially reduces to ray shooting among vertical segments. The previous lower bounds for ray shooting were shown using point obstacles [40].

Note. The reduction from slab emptiness to circle intersection works only in the plane.

2.8 Conclusion

In this chapter we presented quasi-optimal lower bounds for the simplex emptiness and reporting queries in the partition graph model. The lower bounds on simplex emptiness (slab emptiness) also hold for numerous other geometric problems. One interesting open problem is to understand the complexity of halfplane containment queries.

Chapter 3

Data Structures for Restricted Simplex Queries

In Chapter 2, we established lower bounds on simplex emptiness in the partition graph model: for a set S of n points in Euclidean d-space $\mathbb{R}^d$ a data structure with $O(n)$ space must spend $\Omega(\frac{n^{1-\frac{1}{d}}}{\text{polylog}\, n})$ time answering a query. The same lower bounds hold for simplex reporting. Hence, we cannot achieve linear-space and polylogarithmic query time simultaneously for simplex range searching. In this chapter, we consider a restricted version of simplex emptiness queries called *restricted planar simplex emptiness* where the points are in the plane and each query simplex contains a fixed point (origin) in its interior. In the plane a simplex is simply a triangle, so we will use the two terms interchangeably. Since we can triangulate any triangle containing the origin into three triangles such that each triangle has one vertex incident on the origin, from now on we assume that each query triangle has one vertex at the origin. The same idea works for any convex polygon containing the origin, however, the number of query triangles is equal to the number of sides in the polygon.

For this restricted version, the lower bounds of general simplex emptiness do not hold; we present various linear-space data structures for n points in the plane, that achieve

Data Structures for Restricted Triangular Range Searching, Nadia M. Benbernou, Mashhood Ishaque and Diane L. Souvaine. Appeared in the proceedings of 20th Canadian Conference on Computational Geometry, 2008.

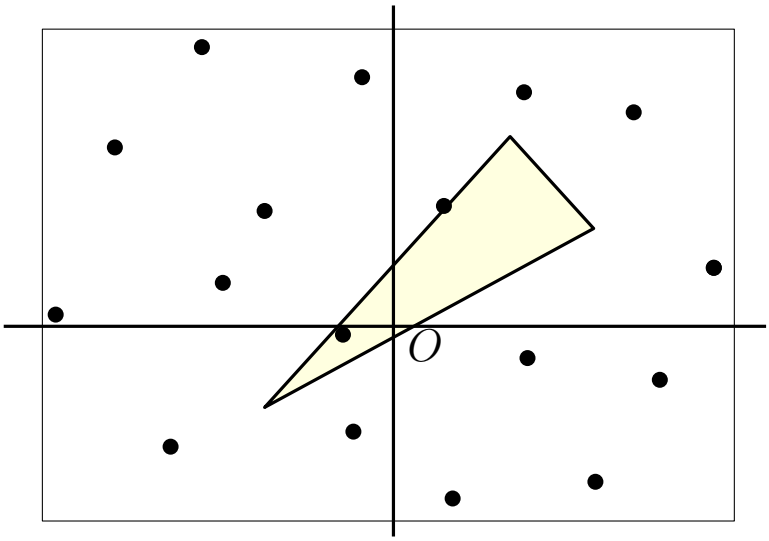


Figure 3.1: Restricted triangular range queries

$O(\text{polylog}\, n)$ query time in the RAM model. In the plane, the restricted version of simplex range searching behaves analogous to halfplane range searching i.e. there exist linear-space data structures for emptiness and reporting that support queries in polylogarithmic times. However, a data structure for restricted simplex counting with $O(n)$ space must spend $\Omega(\sqrt{n})$ time answering a counting query just like the data structures for halfplane range counting. The lower bounds on halfplane range counting were proved by Chazelle [27] in the semigroup arithmetic model. The same lower bounds can be established on restricted simplex range counting via a trivial reduction from halfplane range counting to simplex range counting (See Figure 3.2).

3.1 Previously known upper bounds

The best known data structure for simplex range searching in the plane uses $O(n)$ space and supports queries in $O(\sqrt{n})$ time in the RAM model [23]; additional $O(r)$ time is taken for reporting r points. The data structure is based on simplicial partitioning method. For super-linear space, Chazelle *et al.* [35] gave a data structure in the RAM model with $O(n^{2+\varepsilon})$ space that supports simplex reporting queries in $O(\lg n + r)$ time for any fixed $\varepsilon > 0$, where the constant of proportionality depends on ε. Goswami *et al.* [45] presented a $O(n^2)$

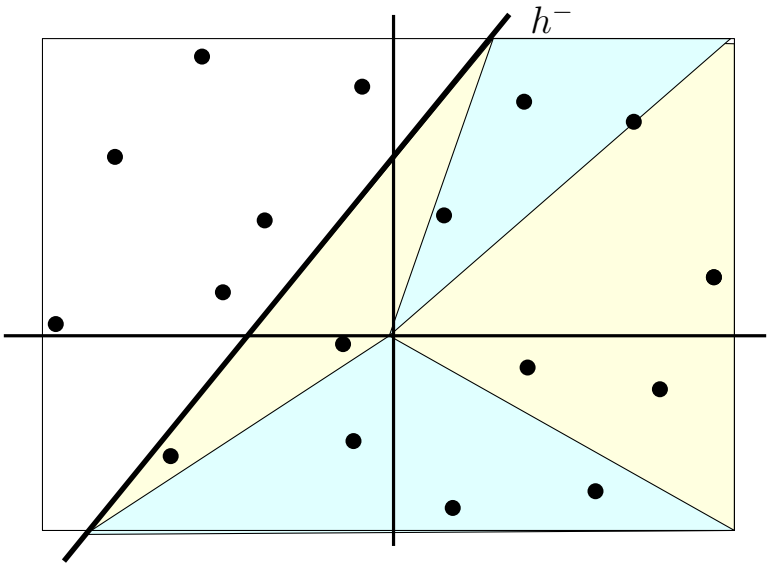


Figure 3.2: Reduction from halfplane range counting to simplex range counting. The halfplane h^- inside the bounding box is a convex polygon with constant number of sides, hence it can be triangulated into a constant number of restricted simplicies. The number of points in the halfplane h^+ is $(n - count_{h^-})$.

space data structure in the RAM model that can support triangular reporting queries in $O(\lg^2 n + r)$ and triangular counting (and hence emptiness) queries in $O(\lg n)$ time.

For halfplane emptiness, a data structure with $O(n)$ space that maintains the convex hull of the given point set, can support emptiness queries in $O(\lg n)$ time in the RAM model. For halfplane range reporting, Chazelle *et al.* [32] gave a linear-space data structure in the RAM model that achieves $O(\lg n + r)$ query time. The data structure maintains nested (peeling) convex layers for the given point set.

3.2 Results

Since it is not possible to achieve the linear-space and the polylogarithmic query time simultaneously for simplex range searching in light of the lower bounds from Chapter 2, we consider a restricted version and give linear-space data structures with polylogarithmic query times in the RAM model. The summary of our data structures for restricted simplex emptiness and reporting is presented in Table 3.2.

Problem	Space	Query	Preprocessing	Section
Emptiness	$O(n \lg n)$	$O(\lg^2 n)$	$O(n \lg n)$	3.4.1
	$O(n)$	$O(\lg^2 n)$	$O(n \lg n)$	3.4.2
	$O(n \lg n)$	$O(\lg n)$	$O(n \lg n)$	3.4.4
	$O(n^{1+\varepsilon})$	$O(2^{\frac{1}{\varepsilon}} \lg n)$	$O(n^{1+\varepsilon})$	3.4.3
Reporting	$O(n \lg n)$	$O(\lg^2 n + r)$	$O(n \lg^2 n)$	3.4.1
	$O(n^{1+\varepsilon})$	$O(2^{\frac{1}{\varepsilon}} \lg n + r)$	$O(n^{1+\varepsilon})$	3.4.3

Table 3.1: Data structures for restricted planar simplex emptiness and reporting.

In Section 3.4.5 we create data structures for ray intersection detection and reporting among an arrangement of n lines, and in Section 3.4.6 we present data structures for non-orthogonal square emptiness and reporting. Note that all reporting queries incur an additional cost of $O(r)$ where r is the number of objects to be reported.

3.3 Tools and Techniques

In order to build our data structures we employ the following data structural techniques: range trees, and space-reducing transformation of Aronov *et al.* [10]. We briefly describe each of these techniques below. We also use standard point-line duality.

3.3.1 Range Trees

For a set of n points sorted by x-coordinate (w.l.o.g), a 1-d range tree is a balanced binary search tree that supports accessing points in any query interval along x-axis. A node in the range tree can be augmented with auxiliary information such as the convex hull of the points in the node's subtree. A 1-d range tree augmented with convex hulls can support any convex hull query constrained by a query interval (vertical strip). To answer a query, the leaves in the range tree corresponding to query interval are located. Then by climbing up the tree from the two leaves until the least common ancestor is reached, up to $O(\lg n)$ auxiliary structures are collected. Together these structures cover all the points in the interval.

For the 1-d range tree on n points there is a $O(\lg n)$-factor overhead for both the space and the query time. For example, a 1-d range tree augmented with convex hulls needs

$O(n \lg n)$ space and supports queries in $O(\lg^2 n)$ time.

Similarly we can create a 2-d range tree, where each node in the tree contains a (possibly augmented) 1-d range tree. A 2-d range tree supports accessing points in an axis-parallel rectangle, and carries an additional overhead of $O(\lg n)$ for both the space and the query time. A 2-d range tree augmented with convex hulls needs $O(n \lg^2 n)$ space and supports queries in $O(\lg^3 n)$ time.

Sometimes it is possible to shave off the extra $O(\lg n)$ factor from the query time of the range tree by using the fractional cascading technique [31].

3.3.2 Space-Reducing Transformation

We are given a data structure DS on n points, with $O(n)$ space supporting a query q in $O(\lg n)$ time. The points $\{p_1, p2_2, \ldots, p_n\}$ are in some sorted order. Using a 1-d range tree augmented with DS, we can answer the query q on any interval $[i, j]$ of points. The query time is $O(\lg^2 n)$. However, we can get rid this extra $O(\lg n)$ factor by the using the space-reducing transformation of Aronov *et al.* [10].

Here is how the transformation works. First a build a naïve data structure on n points that stores one data structure $DS_{i,j}$ for each possible interval $[i, j]$. Since there are n^2 possible intervals, the space for this naïve data structure is n^3. However, the query time is clearly $O(\lg n)$. Now select every mth point in the sorted order and call it a *breakpoint*. For each breakpoint m_i, compute the data structures $DS_{i,i+2}, DS_{i,i+4}, \ldots$ i.e. for each sequence of points starting at m_i whose length is a power of two. Similarly compute $DS_{i,i-2}, DS_{i,i-4}, \ldots$. This constitutes linear space for each breakpoint. Since there are $O(\frac{n}{m})$ breakpoints, the space for these data structures is $O(\frac{n}{m} \cdot n)$. Now for each half-open interval $[m_i, m_{i+1})$ formed by the breakpoints m_i and m_{i+1}, compute the naïve data structure with the $O(m^3)$ space and the $O(\lg n)$ query time. Thus the total space for the data structure after one space-reducing transformation is $O((\frac{n}{m} + 1)m^3 + \frac{n^2}{m})$.

We can apply this space-reducing transformation recursively. Let $M(n)$ be the size of this recursive data structure on n points. The recurrence relation (same as in [10]) for the space of the data structure is:

$$M(n) = (\tfrac{n}{m} + 1)M(m) + O(\tfrac{n^2}{m}).$$

Applying $(\lceil \frac{1}{2} + \frac{1}{\varepsilon} \rceil)$ transformations for any given $\varepsilon > 0$, as in [10], yields the desired space and preprocessing of $O(n^{1+\varepsilon})$.

To answer a query $q_{i,j}$, identify the leftmost and rightmost breakpoints m_i and m_j inside the interval. The open intervals (i, m_i) and (m_j, j) do not contain any breakpoint, thus we must have a recursive data structure for them. Let $Q(m)$ be the time to query these data structures. For the interval $[m_i, m_j]$, query the data structures for associated with the breakpoints m_i and m_j. Each query takes $O(\lg n)$ time and two such queries will cover all the points in the interval. The recurrence relation for query time is given below, with a solution of $O(2^{\frac{1}{\varepsilon}} \lg n)$ (as in [10]):

$$Q(n) = 2Q(m) + O(\lg n).$$

3.4 The Data Structures

3.4.1 $O(\lg^2 n)$ Query Time with $O(n \lg n)$ Space

Given a set of n points in the plane, sort the points rotationally in counterclockwise order around the origin. Assign each point as ID its order in the sorted list of points. Build a 1-d range tree on the point set, augmented with convex hulls (data structure for halfplane emptiness). Now consider any single wedge formed by two rays emanating from the origin. Let i be the first and j be the last point inside the wedge that will be hit if we were to sweep rotationally around origin using a ray going counterclockwise. Observe that all the points inside the wedge are consecutive (with wrap-around) in the sorted order, see Figure 3.3. For any given wedge, we can find the points i and j in $O(\lg n)$ time. For any triangular simplex emptiness query Δ_{abc} with a vertex b incident on the origin, we can extend the two sides as rays $\overrightarrow{ba}$ and $\overrightarrow{bc}$ away from the origin to form a wedge. Now we answer a simplex emptiness query by finding the extreme point in the direction perpendicular to supporting line containing $\overleftrightarrow{ac}$—a halfplane emptiness query. The space for the data structure is $O(n \lg n)$ and the query time is $O(\lg^2 n)$. The preprocessing is $O(n \lg n)$ because we can compute the convex hull of sorted points in $O(n)$ time.

For simplex reporting, we augment the 1-d range tree with Chazelle's nested-convex-layers data structure (for halfplane reporting). The data structure needs $O(n \lg n)$ space,

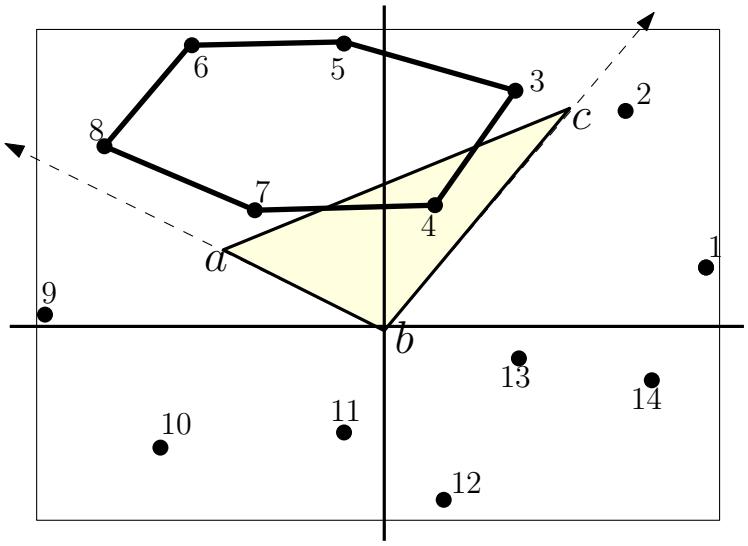


Figure 3.3: Points sorted around the origin

$O(n \lg^2 n)$ preprocessing time and supports queries in $O(\lg^2 n + r)$ time. Daescu *et al.* [37] used a similar idea to build a data structure for halfplane farthest-point queries.

3.4.2 $O(\lg^2 n)$ Query Time with $O(n)$ Space for Emptiness

For emptiness queries, we can improve the space to be $O(n)$ by using the dynamic convex hull structure of Overmars and van Leeuwen [63]. The data structure needs $O(n)$ space, and $O(n \lg n)$ preprocessing. The query time remains $O(\lg^2 n)$.

3.4.3 $O(2^{\frac{1}{\varepsilon}} \lg n)$ Query Time with $O(n^{1+\varepsilon})$ Space

Given a set of n points in the plane sorted rotationally around the origin, a naïve data structure for restricted simplex emptiness queries store a convex hull for each pair of indicies (i, j). Thus the data structure support emptiness queries in $O(\lg n)$ time, but the space requirement is $O(n^3)$. The preprocessing time is $O(n^3)$ because we can compute the convex hull in linear time for points in sorted order. For triangular reporting we store the nested convex layers instead of the convex hulls. The query time is $O(\lg n + r)$, the space is $O(n^3)$, and the preprocessing is $O(n^3 \lg n)$.

The space and processing time for these naïve data structures can be reduced to $O(n^{1+\varepsilon})$ by recursively applying the space-reducing transformation from Aronov *et al.* [10](see Section 3.3.2). The space-reducing transformations preserve the $O(\lg n)$ query time, but the constant is exponential in $\frac{1}{\varepsilon}$.

To answer a triangular emptiness (reporting) query, identify in $O(\lg n)$ time the extreme points i and j inside the wedge. Let m_i and m_j be the breakpoints inside the wedge that are the closest to points i and j respectively. The open intervals (i, m_i) and (m_j, j) do not contain any breakpoint, thus we must have a recursive data structure for them. For the interval $[m_i, m_j]$, two convex hull (convex layers) queries will cover all the points in the interval. For a reporting query we may report some points twice.

3.4.4 $O(\lg n)$ Query Time with $O(n \lg n)$ Space for Emptiness

We apply the fractional cascading technique [31] to the augmented 1-d range tree data structure for restricted simplex emptiness given in Section 3.4.1, and reduce the query time from $O(\lg^2 n)$ to $O(\lg n)$. However, the space requirement remains $O(n \lg n)$.

The basic idea is to augment a 1-d range tree by associating with every node a data structure for extremal point queries for the points in node's subtree, and then to use the fractional cascading technique. At each node we store the set of extremal points and associate with each extreme point a half-open interval of slopes—the point is extremal for slopes in that particular interval. We can store a sorted array of these intervals and for a given slope find the extreme point in $O(\lg n)$ time using binary search. The sorted array at each node also stores pointers to arrays in child nodes such that we need to perform binary search only once. As a result, we perform $O(\lg n)$ extremal point queries, but only the first extremal point query takes $O(\lg n)$ time, and the queries after that can be answered in constant time.

We cannot achieve $O(\lg n + r)$ time for reporting queries using the same method, since Chazelle and Liu [33] showed that the fractional cascading technique does not generalize to planar maps.

3.4.5 Ray Intersection Queries in a Line Arrangement

In the data structures for restricted triangular range queries if the points are sorted by their x-coordinate instead of radially sorted around origin, the data structures can support wedge range searching queries where one of the lines forming the wedge is vertical. Using two such wedges, we can answer the double-wedge range searching queries where one of the lines forming the double-wedge is vertical (see Figure 3.4).

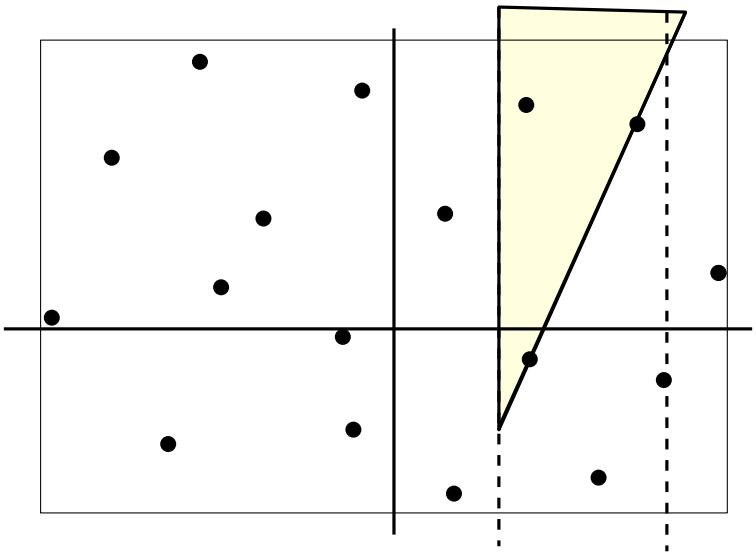


Figure 3.4: A wedge with one vertical line.

Since such a double-wedge corresponds to a ray under the standard point-line duality transform, the data structures for planar restricted simplex range searching can be used to answer ray intersection detection and reporting queries among an arrangement of n lines.

3.4.6 Non-Orthogonal Square Range Searching Queries

A $2d$-range tree augmented with a convex-hull (nested-convex-layers) data structure can support axis-parallel right-triangular emptiness (reporting) queries, with a $O(\lg n)$ factor overhead. Since a non-orthogonal square can be partitioned into at most eight axis-parallel right triangles, the data structures for axis-parallel right triangles also support non-orthogonal square (or rectangles with constant aspect ratio i.e. fat rectangles) emptiness and reporting queries.

This is how we partition an arbitrarily oriented square: from the highest vertex of the given square draw a vertical line segment down to one of the non-adjacent sides. Similarly from the lowest vertex draw a vertical line segment upwards. Let x be the side-length then each vertical segment has a length in the interval $[x, \sqrt{2}x]$. The two diagonals of the square intersect at a distance $\frac{1}{\sqrt{2}}x$ from each vertex. Therefore, the downward vertical segment goes below and the upward vertical segments goes above this point of intersection. Thus we can draw horizontal segment from the lower (upper) endpoint of the downward (upward) vertical segment to the other vertical segment.

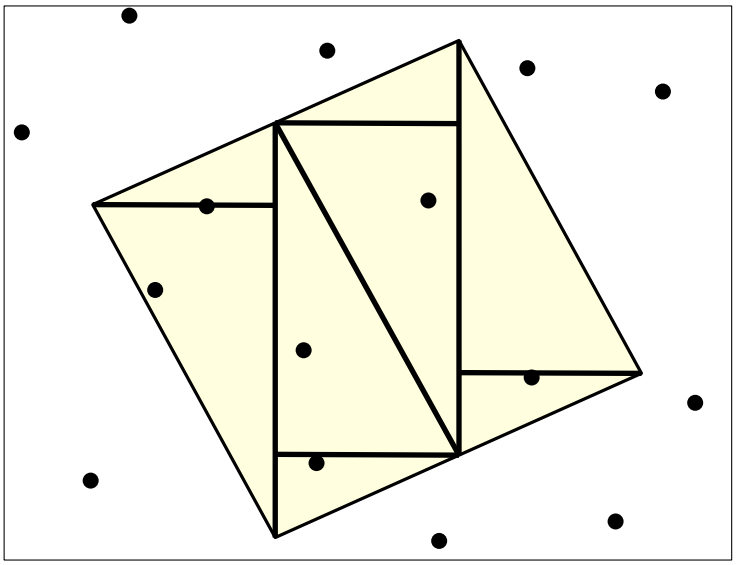


Figure 3.5: Non-orthogonal square range queries

3.5 Conclusion

In this chapter we presented various data structures for the restricted version of the simplex emptiness and reporting queries, where the query simplex contains the origin. We showed that the upper bounds for restricted simplex range searching are similar to those for halfplane range searching. An interesting open problem is how to create the optimal data structures for restricted simplex emptiness and reporting—$O(n)$ space and $O(\lg n)$ query time.

Chapter 4

Data Structures for Permanent Ray Shooting

Ray shooting data structures are a classical core component of computational geometry. They store a set of preprocessed of objects in space such that one can efficiently find the first object hit by a query ray. A simple polygon with n vertices can be preprocessed in $O(n)$ time to answer ray shooting queries in $O(\lg n)$ time, using either a balanced geodesic triangulation [30] or a Steiner triangulation [48]. However, the free space between disjoint polygonal obstacles with a total of n vertices (e.g. $\frac{n}{2}$ disjoint line segments) cannot be handled as easily. The best ray shooting data structures can answer a query in $O(\frac{n}{\sqrt{m}})$ time (ignoring polylogarithmic factors) using $O(m)$ space and preprocessing, based on range searching data structures via parametric search. That is, an average query takes $O(\sqrt{n})$ time using $O(n)$ space. Refer to [3, 66] for higher dimensional variants and special cases.

Geometric algorithms often rely on ray shooting data structures where the result of each query may affect the course of the algorithm and modify the data. The current best dynamic data structure, due to Goodrich and R. Tamassia [44], uses $O(n \lg n)$ space and preprocessing time and supports ray shooting queries, segment insertions and deletions in $O(\lg^2 n)$ time, however, it requires that the free space between the obstacles consists of

Shooting Permanent Rays Among Disjoint Polygons in the Plane, Mashhood Ishaque, Bettina Speckmann and Csaba D. Tóth. A preliminary version appeared in the proceedings of 25th Annual ACM Symposium on Computational Geometry, 2009.

simply connected faces. For the free space between disjoint obstacles (equivalently, for a polygon with holes), the current best data structure uses dynamized range searching data structures via parametric search, which can answer a query in $O(\frac{n}{\sqrt{m}})$ time using $O(m)$ space and preprocessing.

4.1 Related work

For a set of points in the plane, the fully dynamic *convex hull* data structure of Overmars and van Leeuwen [63], as well as the semi-dynamic data structures of Chazelle [24] and Hershberger-Suri [47] rely on a binary hierarchy of nested convex hulls. This idea was recently extended to a semi-dynamic data structure for *geodesic hulls* that supports point deletion and obstacle insertion in a companion paper [51], but it only works in the special case that every face in the free space is *simply connected*, and the runtime uses additional polylogarithmic factors.

A tiling of the free space between disjoint polygons in the plane can serve as a certificate that the polygons do not intersect. If the tiling is easily maintainable as the polygons move, then it can be the basis for kinetic algorithms for collision detection. Kirkpatrick and Speckmann [55, 68], and independently Agarwal *et al.* [4, 14] developed kinetic data structures that are based on tilings with *pseudo-triangles*—simple polygons with exactly three convex angles.

The theoretical study of geodesics in the interior of a simple polygon was pioneered by Toussaint [76, 77]. He showed that the geodesic hull $\mathrm{gh}_D(S)$ of a set S of n points in a simple n-gon D can be computed in $O(n \lg n)$ time, and any line segment in the interior of D crosses at most two edges of $\mathrm{gh}_D(S)$. Mitchell [62] and Ghosh [43] survey results on geometric shortest paths in the plane.

Dynamic ray shooting in simple polygons. Chazelle *et al.* [30] showed that a balanced geodesic triangulation of a polygon with n vertices can be used to answer ray shooting queries in the polygon in $O(\lg n)$ time. Goodrich and Tamassia [44] generalized this data structure to dynamic subdivisions defined by noncrossing line segments where each face is a *simple polygon*. They maintain a balanced geodesic triangulation of each face. For m segments, the data structure has $O(m)$ size. Each segment insertion and deletion, point

location, and ray shooting query takes $O(\lg^2 m)$ time.

4.2 Results

We present a data structure for ray shooting-and-insertion queries among disjoint polygonal obstacles lying in a bounding box B in the plane. Each query is a point p on the boundary of an obstacle and a direction d_p; we report the first point q where the the ray emanating from p in direction d_p hits an obstacle or the bounding box (ray shooting) *and* insert the segment pq (insertion) as a new obstacle edge (Fig. 4.1). If the input polygons have a total of n vertices, our data structure uses $O(n \lg n)$ preprocessing time, and it supports m ray shooting-and-insertion queries in $O((n+m)\lg^2 n + m \lg m)$ total time and $O((n+m)\lg(n+m))$ space. The worst case time bound for a single ray shooting-and-insertion query, however, is $O(n^{\frac{1}{2}+\varepsilon} + \lg m)$ for any fixed $\varepsilon > 0$, where the constant of proportionality depends on ε.

We present two applications for our data structure: efficient implementation of *auto-partitioning* and *convex partitioning* algorithms. The condition that every query point p is on the boundary of an obstacle is satisfied for our motivating applications. This condition cannot easily be relaxed using our current techniques, since our data structure is built on the geodesic hull of all reflex vertices of the free space between the obstacles. If we insert a line segment along a query ray emanating from a point v in the interior of the free space, then v becomes a new reflex vertex of the free space. Introducing m new reflex vertices can generate $\Omega(nm)$ combinatorial changes in the geodesic hull, as was recently shown in [51].

4.3 Applications

Successive ray shooting queries are responsible for a bottleneck in the runtime of some geometric algorithms, which recursively partition the plane along rays (along the portion of rays between their starting points and the first obstacles hit, to be precise). We present two specific applications.

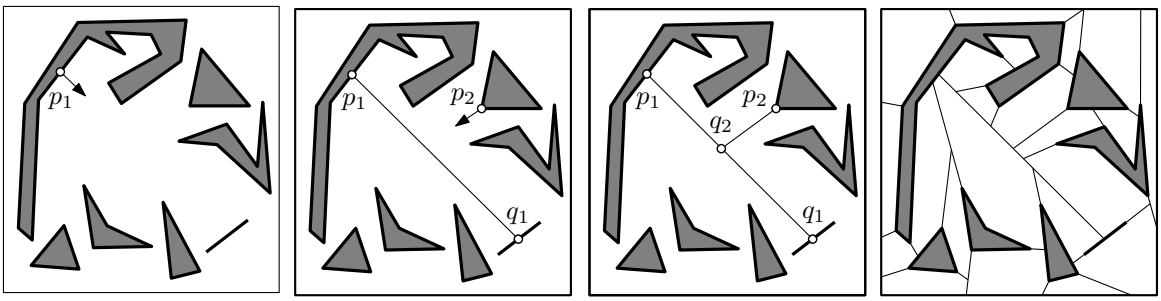


Figure 4.1: (a) Disjoint polygonal objects in the plane; (b) first ray shooting-and-insertion query at p_1; (c) second ray shooting-and-insertion query at p_2; (d) a convex partition of the free space.

4.3.1 Computing a Binary Space Partition

An *binary space partition* (for short, *BSP*) for a set L of disjoint line segments in the plane is a recursive decomposition of the plane into convex cells. Each step partitions the plane along a line and recurses on the segments clipped in each open halfplane. An *auto-partition* is a BSP where each partition is made along the supporting line of an input segment [18]. Patersen and Yao [64] proved that a simple randomized auto-partition that recursively partitions along the supporting lines of randomly selected segments, fragments the input into an expected $O(n \lg n)$ fragments. A dynamic ray shooting data structure for Patersen and Yao's algorithm leads to an expected runtime of $O(n^{\frac{3}{2}-\frac{\varepsilon}{2}})$ time using $O(n^{1+\varepsilon})$ space; and an implementation with a "somewhat disappointing" runtime of $O(n^2 \lg n)$ was presented in [18]. Recently, Tóth [75] presented a deterministic auto-partition that fragments the n input segments into $O(\frac{n \lg n}{\lg \lg n})$ pieces, which is best possible [74]. Another deterministic auto-partition [73] fragments n input segments with k distinct slopes into $O(kn)$ pieces. These auto-partition algorithms are recursive, the partition lines depend on the previous steps of the algorithm, and dynamic ray shooting data structures support $O(n^{\frac{3}{2}} \lg n)$ runtime.

We can implement BSPs and auto-partitions by inserting the partition lines as new barriers. To partition along the supporting line of segment $\ell \in L$, shoot rays from the endpoints of ℓ, and whenever a ray hits another segment ℓ', shoot a new ray from the opposite side of ℓ' in the same direction. An auto-partition that fragments the input segments

into m pieces requires $O(m)$ ray shooting-and-insertion queries. Our data structure supports these queries in $O(n \lg^2 n + m \lg m)$, slightly faster than m arbitrary queries, since in this case many consecutive queries are collinear. In particular, with our structure the classical randomized auto-partition algorithm by Patersen and Yao [64] can be implemented in $O(n \lg^2 n)$ expected time. Similarly, Tóth's deterministic auto-partitions can be implemented in $O(n \lg^2 n)$ time.

4.3.2 Computing a Convex Partition

Assume that we are given a set of disjoint polygons in the plane with a total of n vertices, a permutation of the reflex vertices of the free space, and a half-line at each reflex vertex that partitions the reflex angle into two convex angles. The *convex partitioning* algorithm processes the reflex vertices in the specified order. For each reflex vertex, it draws a segment emanating from the vertex along the given ray until it hits another obstacle, a previously drawn segment, or infinity. Since every reflex angle is split into convex angles, the free space is decomposed into convex faces.

If we are allowed to choose the permutation π, then it is easy to compute *a* convex partition in $O(n \lg n)$ time: first process simultaneously all rays pointing to the right in a left-to-right line sweep, if two segments along the rays meet, give priority to one over the other arbitrarily; then process similarly all the rays pointing to the right in a right-to-left sweep. However, in many applications [7, 8, 49], the order of the reflex vertices and the rays is given online. If π is given (either in advance or online), then previously known best data structures requires $O(n^{\frac{3}{2}-\frac{\varepsilon}{2}})$ time using $O(n^{1+\varepsilon})$ space. Our data structure improves the runtime to $O(n \lg^2 n)$ and uses $O(n \lg n)$ space.

4.4 Techniques

The biggest challenge in the design of our data structure was bridging the gap between the $O(\lg n)$ query time for ray shooting in a simple polygon and the $O(\sqrt{n})$ query time among disjoint obstacles with n vertices. Our data structure is based on two tools which we describe in detail in the beginning of Section 4.5: geometric partition trees in two

dimensions and geodesic hulls. In the remainder of Section 4.5 we introduce the types of polygons we use in our tiling of the free space between the obstacles. In a nutshell, our data structure—which we discuss in Section 4.6—works as follows. In each convex cell of the geometric partition tree, we maintain the geodesic hull of all reflex vertices, which separates the obstacles lying in the interior of the cell from all other obstacles. The geodesic hulls form a nested structure of depth $\lg n$ that consists of weakly simple polygons and creates a tiling of the free space. Each tile is a simple polygon that can easily be processed for fast ray shooting queries. A ray shooting query can be answered by tracing the query ray along these polygons.

The use of geodesic hulls allows us to control the total complexity of m ray insertion queries. Basically, a query ray intersects the boundary of a geodesic hull only if it partitions the set of reflex vertices into two nonempty subsets. Since a set of k points can recursively be partitioned into nonempty subsets at most $k-1$ times, we can charge the total number intersections between rays and geodesic hulls to the number of such partition steps. We detail this analysis in Section 4.7.

4.5 Preliminaries

4.5.1 Geometric Partition Trees

Geometric partition trees are at the core of many hierarchical data structures. A *geometric partition tree* for n points in a bounding box B in Euclidean d-space $\mathbb{R}^d$ is a rooted tree T of bounded degree where (1) every node u corresponds to a convex cell C_u in $\mathbb{R}^d$; (2) the root, at level 0, corresponds to B; (3) the children of every node u correspond to a subdivision of C_u into convex cells; and (4) every cell C_u, $u \in T$, at level k of T contains at most $\frac{n}{2^{ck}}$ points for a constant $c > 0$. The convex cells C_u, for all leaf nodes $u \in T$ form a subdivision of the bounding box B. In particular, in a *binary* geometric partition tree, for every non-leaf node $u \in T$, the cell C_u is partitioned into two convex cells along a hyperplane.

The simplest geometric partition tree for a set S of n points in the plane is a binary partition of the point set by vertical lines. All cells C_u are vertical slabs; if $|S \cap C_u| \geq 2$,

then C_u is subdivided into two slabs by a vertical median of $S \cap C_u$. This *vertical slab* partition tree can be constructed in $O(n \lg n)$ time, and it was used for dynamic convex hull computation [24] and many other early dynamic data structures [63].

Chazelle *et al.* [35] constructed geometric partition trees with *low stabbing numbers* in $O(n \lg n)$ time. In the plane, in particular, they construct for any fixed $\varepsilon > 0$ a geometric partition tree such that the total complexity of all cells is $O(n)$ and any line stabs at most $O(n^{\frac{1}{2}+\varepsilon})$ cells, where the constants of proportionality depend on ε. They essentially alternate two steps $O(\lg n)$ times: (1) decompose a triangular cell along a constant number of lines, and (2) triangulate a cell. It follows that the tree can be represented as a *binary* geometric partition tree where every cell is a convex polygon with at most $O(1)$ vertices. Agarwal and Sharir [5] noted that this structure can support point insertions and deletions in $O(\lg^2 n)$ amortized time using standard dynamization techniques.

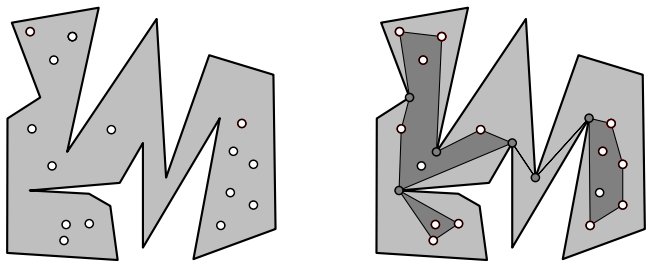


Figure 4.2: (a) A point set S in a simply connected polygonal domain D; (b) the geodesic hull $\mathrm{gh}_D(S)$.

4.5.2 Geodesic Hulls

The *geodesic hull*, which is also known as *relative convex hull*, was introduced in the digital imaging community [67] and later rediscovered in computational geometry (c.f. [76]). It is a key concept for the best available data structures for motion planning, collision detection [4, 14], and robotics [42]. The geodesic hull is a generalization of the convex hull, restricted to some simply connected domain. For a set S of points in the plane, the *convex hull* is the minimum set that contains S and is convex (i.e., contains the line segment between any two of its points). For a set S of points and a simply connected domain D, the

geodesic hull is the minimum set that contains $S \cap D$, lies in D, and contains the shortest path with respect to D between any two of its points.

The boundary of the convex hull is a convex polygon, denoted by $\mathrm{ch}(S)$. Toussaint [77] showed that the boundary of a geodesic hull is a *weakly simple polygon*, denoted by $\mathrm{gh}_D(S)$ (albeit, not necessarily a simple polygon, see Fig. 4.2(b)). A weakly simple polygon on k vertices is a closed polygonal chain $(p_1, p_2, \ldots, p_k)$ such that for any $\varepsilon > 0$ the points p_i can be moved to a location p_i' in the ε-neighborhood of p_i so that $(p_i', p_2', \ldots, p_k')$ is a simple polygon. Toussaint [77] also showed that $\mathrm{gh}_D(S)$ has the property that any line segment lying in D intersects $\mathrm{gh}_D(S)$ in at most two points.

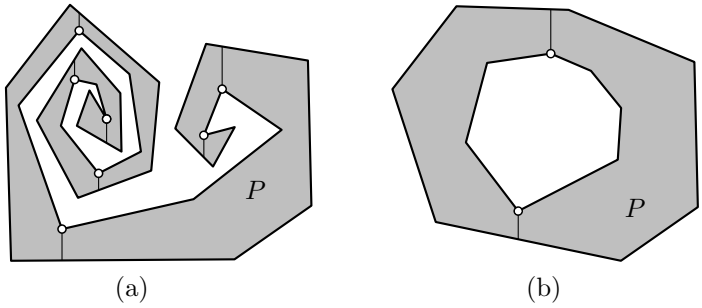


Figure 4.3: (a) A simple crescent polygon; (b) a non-simple crescent polygon.

4.5.3 Crescent Polygons

Data structures for ray-shooting and collision detection in the plane typically use tessellations of the free space between obstacles by *pseudo-triangles*—simple polygons with exactly three convex angles. We construct a tessellation by *crescent polygons*, which are a superset of *spiral polygons*. Crescent polygons are bounded by one convex polygonal chain and at most one reflex polygonal chain (see Fig. 4.3). A polygonal chain on the boundary of a polygon P is *convex* (*reflex*) if the interior angle of P is less (more) than π at every internal vertex of the chain. As opposed to spiral polygons, crescent polygons do not have to be simple—the two polygonal chains can be disjoint. That is, the family of crescent polygons does not only include all spiral polygons (which, in particular, include convex polygons), but also convex polygons with a convex hole (the boundary of the hole is the reflex chain of the polygon).

We build a *monotone decomposition* to store a crescent polygon P. A reflex vertex v of P is y-extremal if both incident vertices are in a closed halfplane bounded by a horizontal line passing through v. For each y-extremal reflex vertex v, shoot a vertical ray in the interior of P, the ray necessarily hits the convex chain of P. (If v is both y-extremal and x-extremal then we consider the two half-planes bounded by a horizontal line through v and shoot the ray in that half-plane which does not contain the two polygon edges adjacent to v.) The rays partition P into y-monotone subpolygons (see Fig. 4.3). Each subpolygon is bounded by a monotone reflex chain, up to two vertical segments, and a convex chain. We store the edges of the convex and the reflex chain of each subpolygon in a self-balancing search tree; we also store the adjacency relations between the subpolygons. This monotone decomposition has $O(n)$ size for a crescent polygon with n edges and can easily be built in $O(n \lg n)$ time. The following propositions formulate some important properties of crescent polygons.

Proposition 4.1 *A line segment inside a crescent polygon P intersects at most two subpolygons of P.*

Proof. If the line segment s is vertical, then it is disjoint from the separators between subpolygons and can hence intersect only one subpolygon. Assume that s is not vertical and crosses a vertical separator between two subpolygons P' and P''. We may also assume without loss of generality that the separator between P' and P'' is incident to a local *minimum* of the reflex chain of P. Then in both P' and P'', the portion of the reflex chain lies *above* the line through s. Any other separator on the boundary of P' or P'' is incident to a local maximum of the reflex chain, which must also be above the line through s. Hence s cannot cross any other separator on the boundary of P' and P''. □

Proposition 4.2 *Let P be a crescent polygon with a reflex chain of n_1 edges, a convex chain of n_2 edges, and a monotone decomposition. A ray shooting query is a triple $(e, p, \overrightarrow{d})$, where e is an edge of P, p is a point on e, and $\overrightarrow{d}$ is a direction. A ray shooting query in the interior of P can be answered in $O(\lg n_1)$ time if the ray hits the reflex chain, and in $O(\lg n_1 + \lg n_2)$ time if it hits the convex chain.*

Proof. By Proposition 4.1, the ray intersects at most two subpolygons of P. The edge e is adjacent to a unique subpolygon, hence we know which subpolygon the ray starts from.

We trace the ray through the subpolygons of P. In each subpolygon P', first detect any intersection with the reflex chain of P'. This takes $O(\lg n_1)$ time, since the reflex chain of P' is y-monotone and its edges are sorted by slope. If the ray intersects the reflex chain, then it ends there and we can report the first point hit. Otherwise, detect any intersection with the vertical separating segments on the boundary of P' in $O(1)$ time. If the ray crosses a separator, then it leaves to an adjacent subpolygon. Otherwise, the ray must hit the convex chain of P'. We can compute the first edge hit in $O(\lg n_2)$ time. □

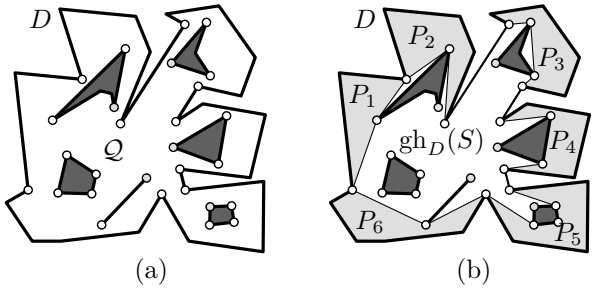


Figure 4.4: (a) A simply connected polygonal domain D containing a set $\mathcal{Q}$ of polygonal obstacles in the interior, empty circles mark the set S. (b) The exterior polygons of $\mathrm{gh}_D(S)$ are $P_1, \ldots, P_6$.

4.5.4 Exterior, Bridge, and Double-Crescent Polygons

Let D be a simple polygon, and let $\mathcal{Q}$ be a set of polygonal obstacles in the interior of D. Denote by S the set of reflex vertices of the free space $\mathrm{int}(D) \setminus (\bigcup \mathcal{Q})$ in the interior of D between the obstacles. D contains the geodesic hull bounded by $\mathrm{gh}_D(S)$. The space in the interior of D and the exterior of $\mathrm{gh}_D(S)$ decomposes naturally into connected components which we call the *exterior polygons* of $\mathrm{gh}_D(S)$, see Fig. 4.4. We show that every exterior polygon is a crescent polygon.

Proposition 4.3 *Every exterior polygon of* $\mathrm{gh}_D(S)$ *is a crescent polygon, where the convex chain is a maximal convex chain of* D *and the reflex chain is a subchain of* $\mathrm{gh}_D(S)$.

Proof. If D is convex, then $\mathrm{gh}_D(S) = \mathrm{ch}(S)$ lies in the interior of D. The boundary of the free space between D and $\mathrm{gh}_D(S)$ has two connected components: an outer boundary D,

and a hole $\mathrm{gh}_D(S) = \mathrm{ch}(S)$. Assume that D has $k \geq 1$ reflex vertices. All reflex vertices are in S. The boundary of the geodesic hull $\mathrm{gh}_D(S)$ passes through all k reflex vertices of D, but none of the convex vertices. The two endpoints of any maximal convex chain of D are connected by two polygonal chains: the maximal convex chain of D and a polygonal chain along $\mathrm{gh}_D(S)$. The chain along $\mathrm{gh}_D(S)$ is reflex, since if it had a convex angle at an internal vertex $p \in S$, then the geodesic hull of S would not contain the shortest path between two neighbors of p. If the length of a maximal convex chain of D is at least two, then the two polygonal chains are different, and they bound a simple polygon. Otherwise, both chains are reduced to the same line segment, and they do not enclose a simple polygon. □

In our data structure, we have to deal with pairs of adjacent crescent polygons, separated by a line. Let a *double-crescent* Q be a polygon bounded by at most two convex chains and at most two reflex chains such that the two reflex chains are separated by a line ℓ and ℓ decomposes Q into two crescent polygons Q_1 and Q_2 (see Fig. 4.5). Let α_1 and α_2 be the convex chains of Q. We define two crescent polygons $P(\alpha_1)$ and $P(\alpha_2)$. $P(\alpha_1)$ is bounded by α_1 and the shortest path between the endpoints of α_1 inside Q; $P(\alpha_2)$ is defined analogously. The crescent polygons $P(\alpha_1)$ and $P(\alpha_2)$ are interior disjoint and are adjacent to α_1 and α_2. If they do not fill Q entirely, then the space in between them is called the *bridge polygon* of Q.

Proposition 4.4 *A double-crescent Q has at most one bridge polygon, which is a simple polygon bounded by two reflex chains on opposite sides of ℓ and two common tangents of the reflex chains of Q.*

Proof. Every internal vertex of the reflex chains of $P(\alpha_1)$ and $P(\alpha_2)$ is a reflex vertex of Q. Since the reflex chains of Q are separated by a line, the overlap of the reflex chain of $P(\alpha_i)$, $i = 1, 2$, and a reflex chain of Q is a continuous polygonal chain. The reflex chain of $P(\alpha_i)$, $i = 1, 2$, consists of subchains of the two reflex chains of Q connected by a their common external tangents, see Fig. 4.5 (c). If these common tangents are not identical, then $P(\alpha_1)$ and $P(\alpha_2)$ do not fill Q entirely, and the remaining space is bounded by a subchain of each reflex chain of Q (a subchain may also be a single edge or a vertex), and the two common external tangents of the reflex chains. □

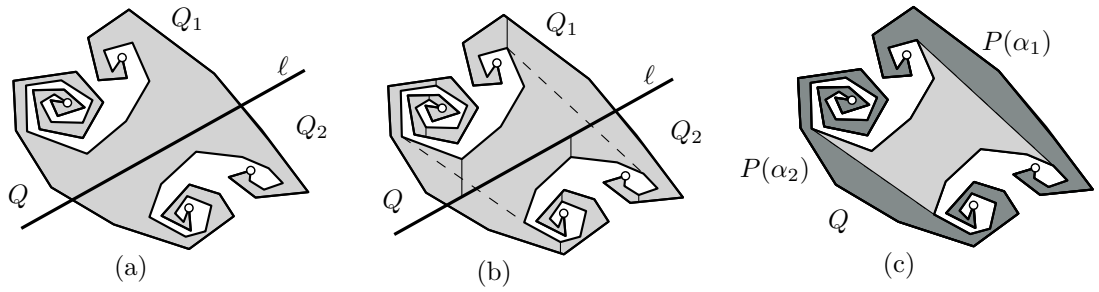


Figure 4.5: (a) Double-crescent Q decomposed into two crescent polygons; (b) the common external tangents of the two reflex chains in Q; (c) the crescent polygons $P(\alpha_1)$ and $P(\alpha_1)$ and a bridge polygon.

Proposition 4.5 *The common external tangents of the two reflex chains of a double-crescent Q (and hence the decomposition of Q into $P(\alpha_1)$, $P(\alpha_2)$, and a possible bridge polygon) can be computed in $O(\lg n)$ time, where n is the total number of edges in the two reflex chains.*

Proof. By Proposition 4.1, line ℓ, which forms the common boundary of Q_1 and Q_2, intersects at most two subpolygons in each of Q_1 and Q_2. All common tangents of the reflex chains of Q_1 and Q_2 cross ℓ, and so they may intersect the subpolygons along ℓ and adjacent subpolygons in the decomposition of Q_1 and Q_2. The common external tangents of the reflex chains of Q are common external tangents of two reflex chains in some pair of these subpolygons. We have $O(1)$ pairs of subpolygons of Q_1 and Q_2 to consider. In each pair, the common tangents can be computed in $O(\lg n)$ time [63], since the reflex chain in each subpolygon is y-monotone. Out of $O(1)$ common tangents of reflex subchains, we can select the common external tangents of the two reflex subchains of Q in $O(1)$ time. □

Our data structure in Section 4.6 tiles the exterior of a geodesic hull with (exterior) crescent polygons. The interior of a geodesic hull is tiled with obstacles, crescent, and double crescent polygons. Hence the free space between the obstacles can be tiled with crescent and bridge polygons, where the crescent polygons are either interior or exterior to a geodesic hull. The details can be found in the next section.

4.6 Data structure

In this section, we present our data structure and establish bounds on its space complexity and preprocessing time. Our structure directly gives a tessellation of the free space between the obstacles into crescent polygons and bridge polygons. We can *answer* a ray shooting query by simply tracing the ray through the tessellation. The update of our data structure after *inserting* the query ray as a new obstacle is discussed in Section 4.7.

4.6.1 Description

We are given a set $\mathcal{P}$ of pairwise disjoint polygons with a total of n vertices lying in the interior of a bounding box B. Let $F = \mathrm{int}(B) \setminus (\bigcup \mathcal{P})$ denote the free space between the obstacles. Denote by S the set of reflex vertices of the free space (that is, convex vertices of the obstacles). The backbone of our data structure is a binary geometric partition tree T for the point set S, where the root corresponds to the bounding box B. The choice of the partition tree has no effect on our results for the total processing time of m queries; it affects only the time required for a single query. If we use a suitable geometric partition tree of *low stabbing number* as proposed by Chazelle *et al.* [35], then each query is processed in $O(n^{\frac{1}{2}+\varepsilon} \lg m)$ time for any fixed $\varepsilon > 0$, where the constant of proportionality depends on ε. We augment the geometric partition tree T, and store several structures related to the portion of the obstacles clipped in C_u at each node $u \in T$. Recall that

- C_u is the convex cell associated with node $u \in T$; and
- ℓ_u is the line splitting cell C_u into two subcells at a non-leaf node $u \in T$.
- Let $S_u = S \cap C_u$ be the set of reflex vertices of the free space in cell C_u;
- let $\mathcal{P}_u$ denote the set of obstacles that intersect the boundary of C_u;
- let $\mathcal{Q}_u$ denote the obstacles lying in the interior of C_u (see Fig. 4.6 (a));
- let $\mathcal{Q}_u^* \subseteq \mathcal{Q}_u$ denote the obstacles $\mathcal{Q}_u$ that intersect the splitting line ℓ_u (in particular, obstacles in $\mathcal{Q}_u^*$ are in the interior of cell C_u but not in the interior of any subcell of C_u).

At the root $u_0 \in T$, we have $C_{u_0} = B$, $\mathcal{P}_{u_0} = \emptyset$, and $\mathcal{Q}_{u_0} = \mathcal{P}$. In general, the set $C_u \setminus (\bigcup \mathcal{P}_u)$ is the union of the free space clipped in C_u and all obstacles in $\mathcal{Q}_u$. Each connected component of $C_u \setminus (\bigcup \mathcal{P}_u)$ is a *simply connected* polygonal domain (see Fig. 4.6(b)). At node $u \in T$, we store all components of $C_u \setminus (\bigcup \mathcal{P}_u)$ that are incident to some vertex in S. Each of them is stored in a doubly linked edge list (components of $C_u \setminus (\bigcup \mathcal{P}_u)$ that are not incident to any reflex vertices of obstacles are *not* stored).

- Let $\mathcal{D}_u$ denote the set of the components of $C_u \setminus (\bigcup \mathcal{P}_u)$ stored at u.

For each polygonal domain $D \in \mathcal{D}_u$, we store $\mathrm{gh}_D(S)$ and every exterior polygon of $\mathrm{gh}_D(S)$ (see Fig. 4.6 (c)). Further, for every non-leaf node, we store pointers between

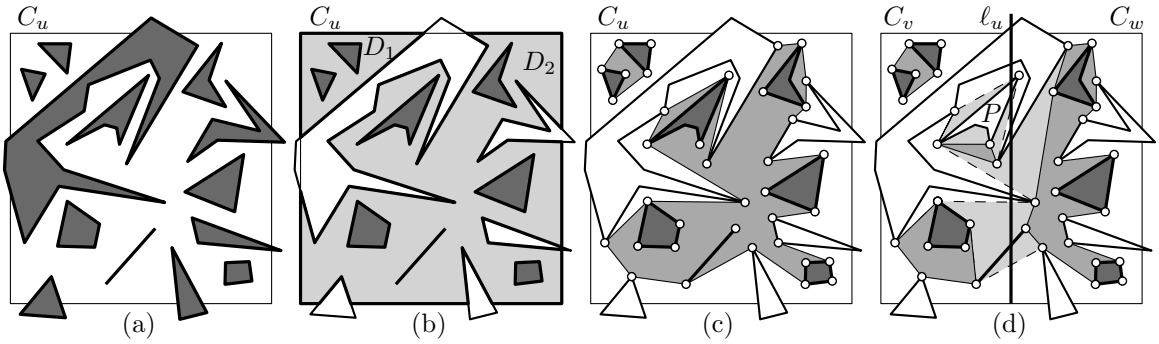


Figure 4.6: (a) A set $\mathcal{P}$ of polygonal obstacles that intersect in a cell C_u; (b) $C_u \setminus (\bigcup \mathcal{P}_u)$ consists of two simply connected polygonal domains D_1 and D_2; (c) the geodesic hulls $\mathrm{gh}_{D_i}(S)$ for $i = 1, 2$; (d) $\mathrm{gh}_{D_2}(S)$ is decomposed into the geodesic hulls $\mathrm{gh}_{D_2^-}(S)$ and $\mathrm{gh}_{D_2^+}(S)$, the obstacle P lying in the interior of C_u but intersecting line ℓ_u, and some bridge polygons.

some adjacent domains and exterior polygons. If $u \in T$ is a non-leaf node, then cell C_u is split along line ℓ_u into two subcells, say $C_v = C_u \cap \ell_u^-$ and $C_w = C_u \cap \ell^+$, where $v, w \in T$ are the two children of u. Consider a domain $D \in \mathcal{D}_u$ that intersects the splitting line ℓ_u and contains some vertices in S on both sides of ℓ_u. Note that the sets $D^- = D \cap \ell_u^- \in \mathcal{D}_v$ and $D^+ = D \cap \ell_u^+ \in \mathcal{D}_w$ may each be disconnected (see Fig. 4.6(d)). If a connected component $\hat{D}$ of D^- or D^+ contains a vertex of S, then it contains a unique domain stored at $\mathcal{D}_v$ or $\mathcal{D}_w$, respectively, which is obtained as $E_v = \hat{D} \setminus (\bigcup \mathcal{P}_v)$ or $E_w = \hat{D} \setminus (\bigcup \mathcal{P}_w)$, after removing the obstacles of $\mathcal{Q}_u$ that intersect ℓ_u. For each domain $D \in \mathcal{D}_v$ (resp., $\mathcal{D}_w$), we maintain a pointer to the domain of $\mathcal{D}_u$ that contains it. For each exterior polygon of each

$\mathrm{gh}_{E_v}(S)$, $E_v \in \mathcal{D}_v$, along line ℓ_u, we maintain a pointer to the adjacent exterior polygon on the opposite side of ℓ_u, which is an exterior polygon of $\mathrm{gh}_{E_w}(S)$ for some $E_w \in \mathcal{D}_w$.

Finally, we store a tessellation of the bounding box B. By construction, any two domains in our data structure, D_1 and D_2, are either interior disjoint or nested (that is, one contains the other). The geodesic hulls $\mathrm{gh}_{D_1}(S)$ and $\mathrm{gh}_{D_2}(S)$ are interior disjoint if the domains D_1 and D_2 are interior disjoint; and $\mathrm{gh}_{D_1}(S)$ lies inside $\mathrm{gh}_{D_2}(S)$ if $D_1 \subseteq D_2$. Therefore, no two edges of two geodesic hulls stored in our data structure cross. We obtain a planar graph, denoted by G, as a union of the edges of B, the edges of all obstacles, and the edges of the geodesic hulls $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$, $u \in T$. The graph G is a tessellation of the bounding box B, where some of the faces are crescent polygons (c.f., Lemma 4.2 below). We store G, and we store each crescent polygon with a monotone decomposition as described in Section 4.5. This completes the description of our data structure.

4.6.2 Structural Properties

We next present a few lemmas on the structure of the tessellation in the exterior and the interior of a geodesic hull $\mathrm{gh}_D(S)$ in a domain $D \in \mathcal{D}_u$ for a node $u \in T$. Let $u \in T$ be a non-leaf node in our data structure, with children v and w. Let $D \in \mathcal{D}_u$ be a domain, which is incident to at least one vertex in S. Assume that the splitting line ℓ_u intersects D, and $D \cap S$ contains vertices on both sides of ℓ_u.

Lemma 4.1 *Consider an edge e of the geodesic hull $\mathrm{gh}_D(S)$ which is not an edge of $\mathrm{gh}_E(S)$ for any domain $E \in \mathcal{D}_v \cup \mathcal{D}_w$.*

1. *e lies in the union of a domain $E_v \in \mathcal{D}_v$ and a domain $E_w \in \mathcal{D}_w$;*
2. *e lies in the union of an exterior polygon P_v of $\mathrm{gh}_{E_v}(S)$ and an exterior polygon P_w of $\mathrm{gh}_{E_w}(S)$;*
3. *both P_v and P_w are adjacent to the splitting line ℓ_u and to the boundary ∂D of D;*
4. *e is the common tangent of the reflex chain of the double-crescent polygon $P_v \cup P_w$.*

Proof. The splitting line ℓ_u decomposes domain D into subdomains. Every edge of $\mathrm{gh}_D(S)$ that lies entirely in a subdomain $E \in \mathcal{D}_v \cup \mathcal{D}_w$ is an edge of $\mathrm{gh}_E(S)$. So we may assume

that edge e is not contained in any domain $E \in \mathcal{D}_v \cup \mathcal{D}_w$, and so it crosses the splitting line ℓ_u.

1. Since e crosses ℓ_u at most once, it can intersect at most two subdomains of D, at most one on each side of ℓ_u. Denote these domains by $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$.

2. Since the geodesic hulls $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ are nested in $\mathrm{gh}_D(S)$, the edge e cannot cross any edge of $\mathrm{gh}_{E_v}(S)$ or $\mathrm{gh}_{E_w}(S)$. It must lie in the exterior of both $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Since E_v and E_w are separated by the line ℓ_u, the edge e can intersect at most one exterior polygon of each of $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Denote these exterior polygons by $P_v \subset E_v$ and $P_w \subset E_w$.

3. If P_v is not adjacent to ℓ_u, then no shortest path between $S \cap D$ can enter the interior of P_v. If P_v is adjacent to ℓ_u but not adjacent to the boundary of D, then it is adjacent to some obstacles in $\mathcal{Q}_u$, which lie in the interior of D. Since every obstacle in $\mathcal{Q}_u$ is contained in $\mathrm{gh}_D(S)$, the polygon P_v is also contained in $\mathrm{gh}_D(S)$. Therefore, if e lies in P_v, then P_v is adjacent to ℓ_u and to ∂D. The same argument holds for P_w.

4. On the boundaries of the exterior polygons P_v and P_w, all vertices of S are in two reflex chains, which we denote by $\beta_v \subset \mathrm{gh}_{E_v}(S)$ and $\beta_w \subset \mathrm{gh}_{E_w}(S)$, respectively. The two chains are separated by the line ℓ_u. Since the geodesic hull of $S \cap D$ with respect to D contains the shortest path between any two points of S, it contains all line segments between the reflex chains on the boundaries of P_v and P_w. The edge e is on the convex hull $\mathrm{ch}(\beta_v \cup \beta_w)$, and so e is a common tangent of the arcs β_v and β_w. □ Next, we study the tiling of the interior of $\mathrm{gh}_D(S)$.

Lemma 4.2 *The interior of* $\mathrm{gh}_D(S)$ *is tiled by the following interior disjoint polygons.*

(i) *geodesic hulls* $\mathrm{gh}_E(S)$*, for some* $E \in \mathcal{D}_v \cup \mathcal{D}_w$*,* $E \subset D$*;*

(ii) *obstacles in* $\mathcal{Q}_u^*$*, which are in the interior of* D*;*

(iii) *crescent polygons along the boundary of obstacles in* $\mathcal{Q}_u^*$*, which are contained in* D*;*

(iv) *bridge polygons in the union of any two adjacent exterior polygons of* $\mathrm{gh}_{E_v}(S)$ *and* $\mathrm{gh}_{E_w}(S)$*, for any* $E_v \cup E_w \subseteq D$*,* $E_v \in \mathcal{D}_v$ *and* $E_w \in \mathcal{D}_w$ *on opposite sides of* ℓ_u*.*

Proof. It is clear that $\mathrm{gh}_D(S)$ contains all geodesic hulls $\mathrm{gh}_E(S)$, for all $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$. Every obstacle in the interior of C_v or C_w is contained in the geodesic hull $\mathrm{gh}_E(S)$

for some $E \in \mathcal{D}_v \cup \mathcal{D}_w$. All obstacles in $\mathcal{Q}_u$, which lie in the interior of D, are contained in $\mathrm{gh}_D(S)$. However, only the obstacles in $\mathcal{Q}_u^*$ are not contained in smaller nested geodesic hulls. If an obstacle is contained in $\mathrm{gh}_D(S)$, then all adjacent crescent polygons are also contained in $\mathrm{gh}_D(S)$, since a geodesic hull of reflex vertices cannot separate an obstacle from an adjacent crescent polygon. It follows that the polygons of type (i), (ii), and (iii) are all contained in $\mathrm{gh}_D(S)$. It is clear that they are interior disjoint. It remains to show that any remaining tile is a bridge polygon.

Delete all polygons of type (i), (ii), and (iii) from the interior of $\mathrm{gh}_D(S)$, and let P be a connected component of the remaining space. Since P is outside of the geodesic hulls $\mathrm{gh}_E(S)$ for any $E \in \mathcal{D}_v \cup \mathcal{D}_w$, it must be contained in exterior polygons. An exterior polygon of $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$, is a crescent polygon, bounded by a convex chain β and a subchain of $\mathrm{gh}_E(S)$. If β is disjoint from the boundary of C_v, then β is a maximal convex chain along an obstacle in $\mathcal{Q}_u^*$, hence the exterior polygon is a crescent polygon adjacent to an obstacle in $\mathcal{Q}_u^*$. If β touches the boundary of C_u but is disjoint from ℓ_u, then the exterior polygon is exterior to $\mathrm{gh}_D(S)$, too, and it is disjoint from P. So P is contained in the exterior polygons of some $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$ that lie in the interior of C_u and are adjacent to ℓ_u.

Consider two adjacent exterior polygons Q_1 and Q_2 on opposite sides of ℓ_u such that P intersects their union $Q_1 \cup Q_2$ (see Fig. 4.5). Let $s \subset \ell_u$ be a common edge of Q_1 and Q_2, and let α_1 and α_2 be the two maximal convex chains along obstacles that contain the endpoints of s. By Proposition 4.4, the union $Q_1 \cup Q_2$ of the two exterior polygons is covered by the crescent polygons of α_1 and α_2, and a possible bridge polygon R. Since P is disjoint from crescent polygons, we have $P \subseteq R$. The bridge polygon R lies in the free space and it is bounded by two geodesics of $\mathrm{gh}_{E_v}(S)$, $E_v \in \mathcal{D}_v$, and $\mathrm{gh}_{E_w}(S)$, $E_w \in \mathcal{D}_w$; and by two of their common external tangents. Therefore, R is interior disjoint from any polygon of type (i), (ii), and (iii); and so $P = R$. □

Corollary 4.1 *The free space between two nested layers of geodesic hulls in our data structure is tiled with crescent polygons and bridge polygons.*

We next study the position of a line segment in the free space with respect to the domains stored in our data structure and the tiling G.

Lemma 4.3 *Let pq be a line segment lying in the free space.*

1. *For every $u \in T$, segment pq intersects at most one domain $D \in \mathcal{D}_u$.*
2. *Segment pq intersects at most two crescent polygons in the tessellation G of B.*
3. *For every $u \in T$, segment pq intersects at most one bridge polygon within $\mathrm{gh}_D(S)$, $D \in \mathcal{D}_u$.*

Proof. 1. If pq contains points $a \in D_a$ and $b \in D_b$, with $D_a, D_b \in \mathcal{D}_u$, then the line segment $ab \subseteq pq$ lies in the convex cell C_u. Since $D_a, D_b \subset C_u$ are separated by obstacles in $\mathcal{P}$, segment ab must traverse an obstacle, contradicting the assumption that pq lies in the free space.

2. Each crescent polygon is bounded by a convex chain along an obstacle P and a reflex chain. If a ray $\overrightarrow{pq}$ enters a crescent polygon P, it enters through its reflex chain and hits the convex chain along P. Since the rays $\overrightarrow{pq}$ and $\overrightarrow{qp}$ can hit at most two convex chains along obstacles, pq intersects at most two crescent polygons.

3. If pq intersects two distinct bridge polygons in the tiling of $\mathrm{gh}_D(S)$, denote by $R_1, R_2 \subset D$ two consecutive bridge polygons along pq. Let $a, b \in pq$ be the closest points on the boundaries of R_1 an R_2, respectively. Since bridge polygons are not adjacent, we have $a \neq b$, and segment $ab \subset pq$ does not pass through any other bridge in $\mathrm{gh}_D(S)$. The segment ab cannot cross ℓ_u, since all points of ℓ_u in the interior of $\mathrm{gh}_D(S)$ are covered by obstacles, crescent polygons, and bridges. Hence, ab lies on one side of the line ℓ_u, and so ab lies in some domain $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$. That is, ab connects two points along $\mathrm{gh}_E(S)$, while the relative interior of ab is in the exterior of $\mathrm{gh}_E(S)$. This contradicts the fact that the shortest path between any two points along $\mathrm{gh}_E(S)$ passes in the geodesic hull of $S \cap E$ with respect to E. □

4.6.3 Space

A binary geometric partition tree T for n points in the plane has $O(n)$ nodes and $O(\lg n)$ height. A node $u \in T$ stores a convex cell C_u, of constant complexity, and a splitting line ℓ_u. Every vertex of a domain $D \in \mathcal{D}_u$ is either a vertex of an obstacle or the intersection of an edge of an obstacle and the boundary of the cell C_u. Since C_u has $O(1)$ edges, a domain

$D \in \mathcal{D}_u$ has at most $O(1)$ consecutive vertices that are not vertices of any obstacle. Recall that a domain $D \in \mathcal{D}_u$ is stored only if it contains at least one reflex vertex in S. Every vertex in S lies in at most $O(\lg n)$ cells C_u (at most one on each level of T), hence the domains $D \in \mathcal{D}_u$ for all $u \in T$ have a total of $O(n \lg n)$ vertices. All vertices of $\mathrm{gh}_D(S)$ are in S, and each vertex $p \in S$ occurs in $O(\lg n)$ geodesic hulls (one at each level of T), hence the total complexity of $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$ and $u \in T$ is $O(n \lg n)$. Finally, the planar straight line graph G has n vertices and $O(n)$ edges. So the total size of all crescent and bridge polygons of G is $O(n)$.

4.6.4 Preprocessing

A binary geometric partition tree T for n points can be computed in $O(n \lg n)$ time, for both the vertical slab tree [63] or the one with low stabbing number [35]. In a top-down traversal of the tree T, we can compute all domains $D \in \mathcal{D}_u$, the sets S_u of reflex vertices lying in C_u, and the sets $\mathcal{Q}_u$ of obstacles in the interior of C_u. At the root u_0, we have $\mathcal{D}_{u_0} = \{B\}$. If a domain $D \in \mathcal{D}_u$ intersects the splitting line ℓ_u, it is decomposed into subdomains as follows: test every edge along the boundary of D and every edge of obstacles lying in the interior of D, whether they intersect ℓ_u. The intersection points of ℓ_u with the edges of ∂D can be sorted along ℓ_u by traversing the boundary of the simple polygon D. We can insert into this sorted list each intersection with obstacles in $\mathcal{Q}_u$ in $O(\lg n)$ time. We can trace out the boundary of each subdomain of D by traversing the edges of D, the portions of ℓ_u between consecutive intersection points, and the boundaries of obstacles $\mathcal{Q}_u$ crossed by line ℓ_u. We discard any subdomain of D that is not incident to any reflex vertex in S. We spend $O(\lg n)$ time per edge for sorting edges of the obstacles in $\mathcal{Q}_u$ stabbed by ℓ_u, which were in the interior of domain D but will be on the boundary of subdomains of D; and we spend $O(1)$ time for each edge of each domain $D \in \mathcal{D}_u$. Over each of the $O(\lg n)$ levels of T, we spend $O(n)$ time traversing the edges of the domains and the obstacles. For each of the $O(n)$ obstacles, we spend $O(\lg n)$ time when it is first crossed by a splitting line ℓ_u, and we insert it into the sorted order the intersections of ℓ_u with the boundary of a domain. For all $u \in T$, we compute all domains $D \in \mathcal{D}_u$, sets S_u, and sets $\mathcal{Q}_u$ in $O(n \lg n)$ total time.

We compute the crescent polygons and the geodesic hulls $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$,

$u \in T$, in a bottom-up traversal of T. Note that for any polygonal domain $D \in \mathcal{D}_u$, $u \in T$, all reflex vertices of D are in S_u. If $u \in T$ is a leaf, then cell C_u contains exactly one vertex of S, and the geodesic hull $\mathrm{gh}_D(S)$ is a single point. Assume now that $u \in T$ is a non-leaf node with children v and w, the geodesic hull $\mathrm{gh}_E(S)$ has been computed for every subdomain $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$, and we want to compute $\mathrm{gh}_D(S)$. By Lemma 4.1, we obtain all new edges of $\mathrm{gh}_D(S)$ and any crescent polygon contained in $\mathrm{gh}_D(S)$ by computing the common external tangents of two reflex polygonal arcs along adjacent exterior polygons of $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$. By Proposition 4.5, each common external tangent can be computed in $O(\lg n)$ time. Since G is a planar graph with n vertices and $O(n)$ edges, a total of $O(n)$ common tangents are computed. Hence we can compute all geodesic hulls $\mathrm{gh}_D(S)$ and crescent polygons in $O(n \lg n)$ total time. The total size of all crescent polygons in the tiling G is $O(n)$, and so we can compute a monotone decomposition for each crescent polygon in the tessellation in $O(n \lg n)$ total time.

4.7 Ray Tracing and Update Operations

4.7.1 A Single Ray Shooting-and-Insertion Query

Given a point p on the boundary of an obstacle and a direction d_p, we answer the associated ray shooting query by tracing the ray through the bridge and crescent polygons of the tessellation G until it hits the boundary of an obstacle or leaves the convex hull $\mathrm{ch}(S)$. By Proposition 4.2, we can trace a ray through these polygons in logarithmic time in terms of the number of vertices. Recall that after inserting $O(m)$ rays, the number of convex vertices is $O(n+m)$ but the number of reflex vertices remains $O(n)$. Therefore, ray shooting in a crescent or bridge polygon takes $O(\lg n)$ time if the ray hits a reflex chain or line ℓ_u; and it takes $O(\lg(m+n))$ time if it hits a convex chain. Since convex chains lie on the boundary of obstacles, each ray hits at most one convex chain. So if the segment pq traverses k faces of the tessellation G, then the ray shooting query takes $O(k \lg n + \lg(m+n)) = O(k \lg n + \lg m)$ time.

After answering the ray *shooting* query, we also *insert* the portion of the ray between the starting point p and the first point q where it hits an obstacle as a new obstacle into our

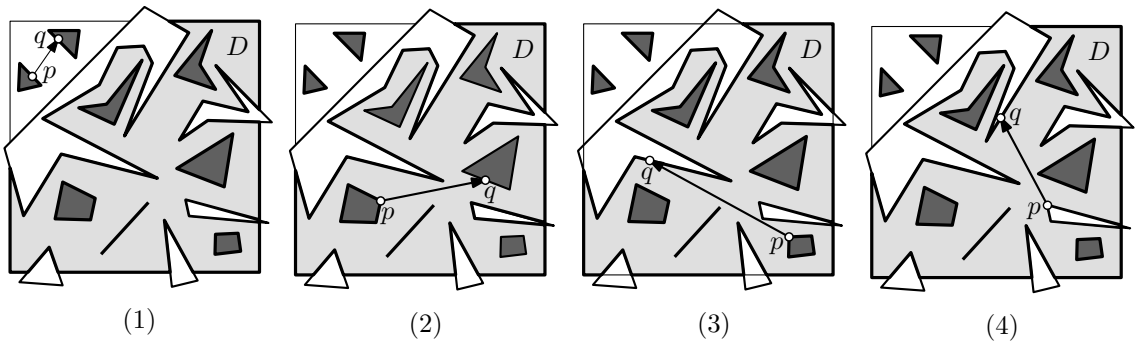


Figure 4.7: Cases for updating a domain $D \in \mathcal{D}_u$ depending on the position of the segment pq w.r.t. D.

data structure. The tree T (and the corresponding hierarchical cell decomposition) remains the same.

First, we update the set of domains $\mathcal{D}_u$, $u \in T$. As before, let v and w denote the children of a non-leaf node $u \in T$. In a top-down traversal of the tree T, detect all domains $D \in \mathcal{D}_u$ that intersect pq. By Lemma 4.3, pq intersects at most one domain $D \in \mathcal{D}_u$ for each $u \in T$. We distinguish four cases depending on the position of pq within $D \in \mathcal{D}_u$ (refer to Fig. 4.7). (1) If pq is disjoint from D, then no update is necessary, and there is no need to descend to the polygons of $\mathcal{D}_v$ and $\mathcal{D}_w$ contained in D. (2) If pq lies in the interior of D (that is, its endpoints lie on obstacles in the interior of D), then D is not updated, but we descend to the domains of $\mathcal{D}_v$ and $\mathcal{D}_w$ contained in D. (3) If pq has exactly one endpoint in the interior of D, say p, which is incident to an obstacle P in the interior of D, then we update D by appending the edges $pq \cap D$ and all edges of P to the boundary of D; and descend to its subdomains in $\mathcal{D}_v$ and $\mathcal{D}_w$. (4) Finally, if pq intersects the boundary of D twice, then we split D into two domains; and descend to its subdomains in $\mathcal{D}_v$ and $\mathcal{D}_w$.

We have identified all domains D that intersect the segment pq. In each domain $D \in \mathcal{D}_u$ along pq, we also update the geodesic hulls $\mathrm{gh}_D(S)$. This is done in a bottom up traversal of the tree T. When processing the geodesic hull with respect to $D \in \mathcal{D}_u$, $u \in T$, assume that the geodesic hulls $\mathrm{gh}_E(S)$ have been updated for all $E \in \mathcal{D}_v \cup \mathcal{D}_w$ at the children $v, w \in T$ of u. Note that $\mathrm{gh}_D(S)$ changes only if the domain D changes, that is, in cases (3) and (4) above.

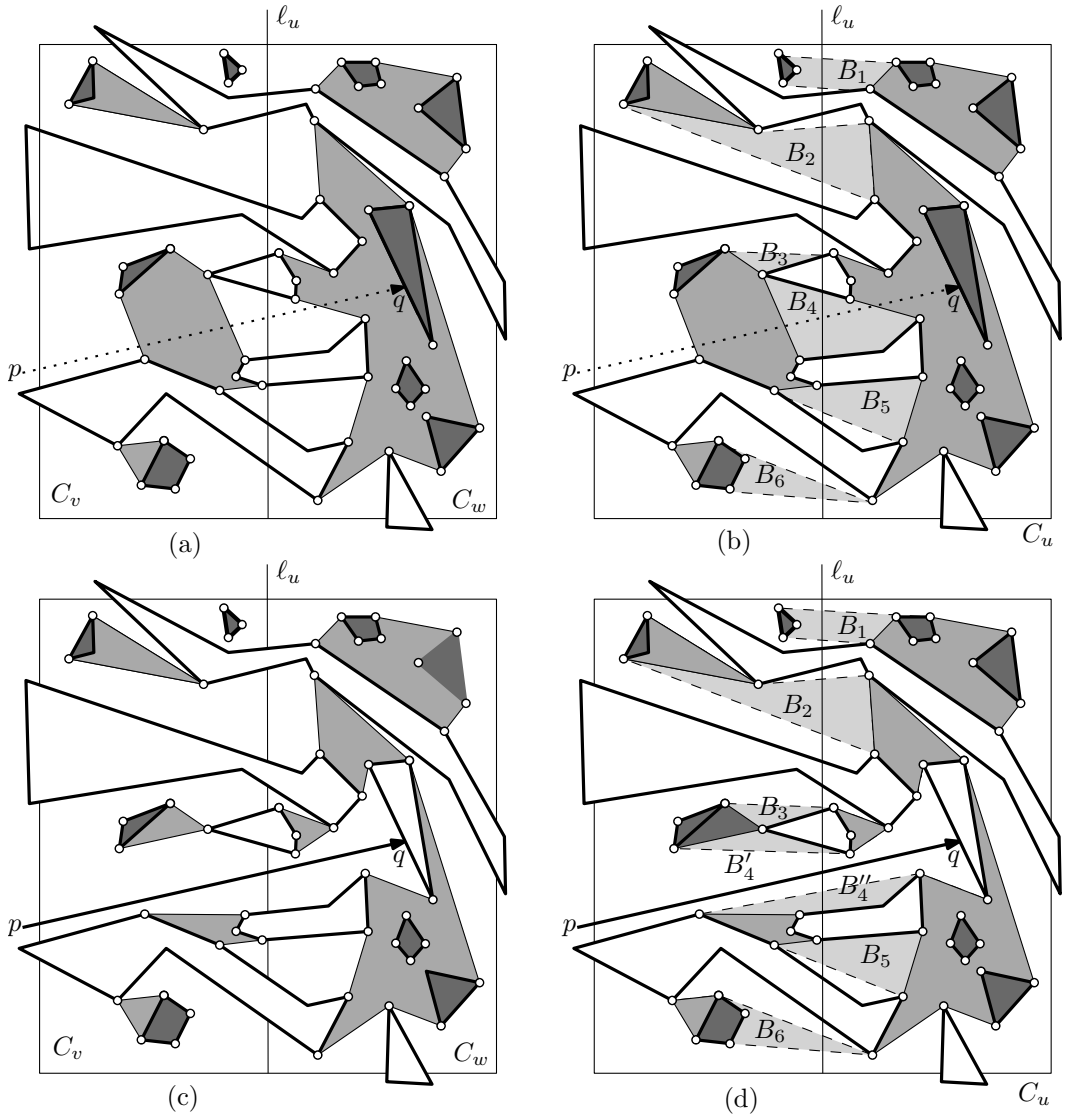


Figure 4.8: The geodesic hulls $\mathrm{gh}_D(S)$ for $D \in \mathcal{D}_v \cup D_w$ before (a) and after (b) inserting segment pq. The geodesic hulls $\mathrm{gh}_D(S)$ for $D \in \mathcal{D}_u$ before (c) and after (d) inserting segment pq.

By Lemma 4.3, segment pq crosses at most two edges of $\mathrm{gh}_D(S)$, these edges have to be removed. By Lemma 4.1, every edge of a crescent polygon or $\mathrm{gh}_D(S)$ that is not an edge of any $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$, is a common external tangent of two reflex arcs in two exterior polygons of some $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ for some $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$

that are adjacent to line ℓ_u. It follows that an update is necessary only if the two exterior polygons are adjacent to line ℓ_u and the segment pq. There are two pairs of such extremal polygons, one on each side of pq. It is enough to compute the common external tangents between $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ in these two pairs of exterior polygons. Hence, we can update the geodesic $\mathrm{gh}_D(S)$ and the crescent polygons of $\mathcal{Q}_u^*$ by deleting two edges crossed by pq and inserting two new common external tangents between two reflex arcs of $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Since any two reflex arcs have a total of $O(n)$ vertices, we can update the geodesic hull in $\mathrm{gh}_D(S)$ in $O(\lg n)$ time by Proposition 4.5.

4.7.2 Time Complexity of m Successive Queries

For a segment pq in the free space, denote by $\mathrm{tr}(pq)$ the number of geodesic hulls $\mathrm{gh}_D(S)$ on all levels of T whose edges cross pq. We need to update all $\mathrm{tr}(pq)$ geodesic hulls along pq (some of which may split into two parts). We have seen that the ith ray shooting query takes $O(\mathrm{tr}(p_iq_i)\lg n + \lg i)$ time. The *insertion* of the ith segment p_iq_i as a new obstacle takes $O(\mathrm{tr}(p_iq_i)\lg n))$ time. The total runtime of processing m successive rays shooting and insertion queries is

$$O\left(\left(\sum_{i=1}^{m}\mathrm{tr}(p_iq_i)\right)\lg n + m\lg m\right).$$

Proposition 4.6 *For m successive ray shooting-and-insertion queries, we have*

$$\sum_{i=1}^{m}\mathrm{tr}(p_iq_i) = O((n+m)\lg n).$$

Proof. The tree T has $O(\lg n)$ levels. The domains $D \in \mathcal{D}_u$, for all nodes u on one level of T, are pairwise interior-disjoint, and they partition S into subsets $S \cap D$. Hence, the geodesic hulls $\mathrm{gh}_D(S)$ are pairwise disjoint on each level of T. We deduce an upper bound on the number of crossings between query segments p_iq_i and the edges of the geodesic hulls on one level of T. By Lemma 4.3, a segment p_iq_i crosses at most two edges of each geodesic hull $\mathrm{gh}_D(S)$.

We distinguish two cases. (1) If p_iq_i crosses one edge of $\mathrm{gh}_D(S)$, then one endpoint of p_iq_i lies in the interior of $\mathrm{gh}_D(S)$. Since the geodesic hulls are disjoint a level of T, each point p_i or q_i lies in the interior of at most one geodesic hull $\mathrm{gh}_D(S)$. The number of this type of crossings is at most $2m$. (2) If p_iq_i crosses two edges of $\mathrm{gh}_D(S)$, then it partitions D into two domains, each containing some vertices of S. Since a set of cardinality k can recursively be partitioned into nonempty subsets at most $k-1$ times, the sets $D \cap S$ on each level are partitioned at most $O(n)$ times. Hence the number of this type of crossings is at most $O(n)$. Summing both types of crossings over all $O(\lg n)$ levels of T, we obtain $\sum_{i=1}^{m} \mathrm{tr}(p_iq_i) = O((n+m)\lg n)$. □ The total runtime of m successive ray shooting-and-insertion queries is $O((n+m)\lg^2 n + m \lg m)$. If the geometric partition tree T is arbitrary, then $\mathrm{tr}(pq) = \Theta(n \lg n)$ is possible and the ith ray shooting-and-insertion query takes $O(n \lg^2 n + \lg i)$ time. However, if we use a geometric partition tree T of low stabbing number, then every line intersect at most $O(n^{\frac{1}{2}+\delta})$ cells C_u, $u \in T$. By Lemma 4.3(1), a line segment in the free space intersects at most one domain $D \in \mathcal{D}_u$ and at most one geodesic hull $\mathrm{gh}_D(S)$ in cell C_u, and so $\mathrm{tr}(pq) = O(n^{\frac{1}{2}+\delta} \lg n)$. Hence the ith ray shooting-and-insertion query takes $O(n^{\frac{1}{2}+\delta} \lg^2 n + \lg i) \leq O(n^{\frac{1}{2}+2\delta} + \lg i)$ for any fixed $\delta > 0$.

4.7.3 Collinear Ray Shooting-and-Insertion Queries

In our application in auto-partitions, many consecutive ray shooting-and-insertion queries are collinear. Recall that for a partition step along the supporting line of an obstacle segment $\ell \in L$, we shoot rays from the endpoints of ℓ, and whenever a ray hits another obstacle segment ℓ', we shoot a new ray from the opposite side of ℓ' in the same direction. An auto-partition that fragments n disjoint input segments into m pieces can be implemented with $2n + m$ ray shooting-and-insertion queries. Since there are 2n groups of consecutive collinear queries, we can slightly improve the general runtime of $O((n+m)\lg^2 n + m \lg m)$ to $O(n \lg^2 n + m \lg m)$. In particular, if $m = O(n \lg n)$, then the total runtime becomes $O(n \lg^2 n)$.

Proposition 4.7 *Suppose that m successive ray shooting-and-insertion queries are arranged into m_0 groups of consecutive collinear queries such that if i and $i+1$ are in*

the same group, then points q_i and p_{i+1} are on the boundary of the same obstacle. Then we have $\sum_{i=1}^{m} \mathrm{tr}(p_i q_i) = O((n + m_0) \lg n)$.

Proof. For $j = 1, \ldots, m_0$, denote by $a_j b_j$ the convex hull of the j-th group of collinear segments $p_i q_i$. The tree T has $O(\lg n)$ levels. The domains $D \in \mathcal{D}_u$, for all nodes u on each level of T, are pairwise interior-disjoint, and they partition S into subsets $S \cap D$. Hence, the geodesic hulls $\mathrm{gh}_D(S)$ are pairwise disjoint on each level of T.

We deduce an upper bound on the number of crossings between query segments $p_i q_i$ and the edges of the geodesic hulls on one level of T. Recall that each geodesic hull $\mathrm{gh}_D(S)$ is simply connected. If the consecutive and collinear query segments along $a_j b_j$ cross r edges of a geodesic hull $\mathrm{gh}_D(S)$ and $a_j b_j$ has 0 (resp., 1 or 2) endpoints in the interior of $\mathrm{gh}_D(S)$, then $a_j b_j$ decomposes $\mathrm{gh}_D(S)$ into $\frac{r}{2}$ (resp., $\frac{r-1}{2}$ or $\frac{r-2}{2}$) pieces; and it also partitions the set $D \cap S$ into the same number of subsets. If a_j (b_j) lies in the interior of $\mathrm{gh}_D(S)$, then we charge the nearest crossing to point a_j (b_j). We charge all other crossings to the partitioning of set $D \cap S$. Since the geodesic hulls $\mathrm{gh}_D(S)$ are pairwise disjoint on one level of T, we charge at most $2m_0$ crossings to the points a_j, b_j, for $j = 1, 2 \ldots, m_0$. Since a set of cardinality k can recursively be partitioned into nonempty subsets at most $k - 1$ times, we charge at most $O(n)$ to the partitioning of the sets $D \cap S$. Summing the crossings over $O(\lg n)$ levels, we obtain $\sum_{i=1}^{m} \mathrm{tr}(p_i q_i) = O((n + m_0) \lg n)$. □

4.8 Conclusion

We proposed a data structure for disjoint polygonal obstacles in the plane with n vertices that supports m ray shooting-and-insertion queries in $O((n + m) \lg^2 n + m \lg m)$ total time and $O((n + m) \lg n)$ space. Our data structure applies, with minimal adjustments, to polygons with holes having a total of n vertices. With our data structure, we improve the expected runtime of Patersen and Yao's classical randomized auto-partition algorithm for n disjoint line segments in the plane to $O(n \lg^2 n)$. Also the convex partition of the free space between n disjoint line segments in the plane (with $m = n$), where the segments are extended beyond their endpoints in a specified order, can now be computed in $O(n \lg^2 n)$ time and $O(n \lg n)$ space. It remains a challenge for future research to find an $O(n \lg n)$

time algorithm.

It was essential for our techniques that the obstacles are *polygonal*. We believe that our techniques can be extended to handle curvilinear polygons, bounded by arcs of algebraic curves of bounded degree. Typically, n such arcs can be decomposed into a finite number of locally convex or concave arcs, and be approximated well enough by polygonal arcs with a total of $O(n)$ vertices. The extension of our data structure to disjoint obstacles of bounded description complexity is left for future work.

Chapter 5

Convex Partitions with 2-Edge-Connected Dual Graphs

Convex partitioning decomposes a complex geometric object into convex parts. It is a very useful tool in computer graphics, motion planning, and geometric modeling. Given a set of convex polygonal obstacles and a bounding box, we may think of the bounding box as a simple polygon and the obstacles as polygonal holes. Then the problem of creating a convex partition becomes that of decomposing the simple polygon with holes into convex parts. Convex polygonal decomposition has received considerable attention in the field of computational geometry. The focus has been to produce a decomposition with as few convex parts as possible. Lingas [59] showed that finding the *minimum convex decomposition* (decomposing the polygon into the fewest number of convex parts) is NP-hard for polygons with holes. For polygons without holes, however, minimum convex decompositions can be computed in polynomial time [29, 54]—see [53] for a survey on polygonal decomposition.

While minimum convex decomposition is desirable, it is not the only criterion for the *goodness* of a convex partition (decomposition). In fact, the measure of the quality of a convex partition can be specific to the application domain. In Lien's and Amato's work

Convex Partitions with 2-Edge Connected Dual Graphs, Marwan Al-Jubeh, Michael Hoffmann, Mashhood Ishaque, Diane L. Souvaine, and Csaba D. Tóth. A version of this work is to appear in the Special issue of Journal of Combinatorial Optimization, 2010. Previously published in the proceedings of 15th International Computing and Combinatorics Conference, 2009. Abstract appeared in 18th Fall Workshop on Computational Geometry, 2008.

on approximate convex decomposition [58] with applications in skeleton extraction, the goal is to produce an approximate (not all cells are convex) convex partition that highlights salient features. In the *equitable* convex partitioning problem, all convex cells are required to have the same value of some measure e.g. the same number of red and blues points [52], or the same area [22] (with application to vehicle routing).

Another criterion for the quality of a convex partition might be some property of its dual graph (the definition of dual graph varies from application to application). A dual graph might represent a communication network whose desired characteristic is fault tolerance (no single point of failure). We consider the problem of creating convex partitions with 2-edge connected dual graphs.

> **Problem Definition:** For any finite set of disjoint convex polygonal obstacles in the plane, with a total of n vertices, partition the free space between the obstacles into into open convex cells such that the dual graph of the convex partition is 2-edge connected.

5.1 Convex Partitions and Dual Graphs

For a finite set S of disjoint convex polygonal obstacles in the plane $\mathbb{R}^2$, a *convex partition* of the free space $\mathbb{R}^2 \setminus (\bigcup S)$ is a set C of open convex regions (called *cell*s) such that the cells are pairwise disjoint and their closures cover the entire free space. Since every vertex of an obstacle is a reflex vertex of the free space, it must be incident to at least two cells. Let σ be an assignment of every vertex to two adjacent convex cells in C. A convex partition C and an assignment σ define a *dual graph* $D(C, \sigma)$: the cells in C correspond to the nodes of the dual graph, and each vertex v of an obstacle corresponds to an edge between the two cells assigned to v (see Fig. 5.1). Double edges are possible, corresponding to two endpoints of a line-segment obstacle on the common boundary of two cells.

It is straight forward to construct an arbitrary convex partition for a set of convex polygons as follows. Let V denote the set of vertices of the obstacles; each vertex of a convex obstacle is reflex. Let π be a permutation on V. Process the vertices in the order π. For a vertex $v \in V$, draw a directed line segment (called *extension*) that starts from the vertex

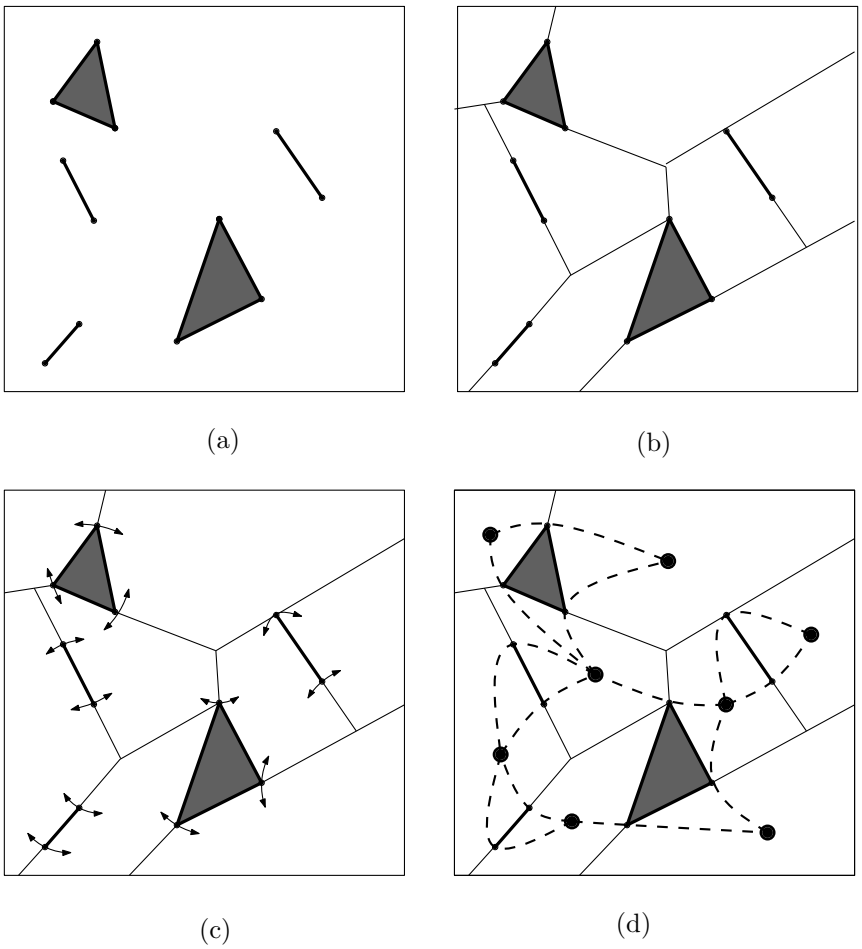


Figure 5.1: (a) Five obstacles with a total of 12 vertices. (b) A convex partition. (c) An assignment σ. (d) The resulting dual graph.

along the angle bisector. For a line-segment obstacle, the extension is drawn along the supporting line. The extension ends when it hits another obstacle, a previous extension, or infinity (the bounding box). For k convex obstacles with a total of n vertices, this naïve algorithm produces a convex partition with $n-k+1$ cells, if no two extensions are collinear. For example, for n disjoint line segments (with $2n$ endpoints) in general position, we obtain $n+1$ cells. If the obstacles are in general position, then each vertex v is incident to exactly two cells, lying on opposite sides of the extension emanating from v. Hence the assignment σ is unique, and the choice of permutation π completely determines the dual graph $D(\pi)$. We call this a STRAIGHT-FORWARD convex partition, and a STRAIGHT-FORWARD dual

graph, which depends only on the permutation π of the vertices.

5.2 Related Problems

5.2.1 Disjoint Compatible Matchings

A *plane matching* is a set of n disjoint line segments in the plane, which is a perfect matching on the $2n$ endpoints. Two plane matchings on the same vertex set are *compatible* if there are no two edges that cross, and are *disjoint* if there is no shared edge. Aichholzer *et al.* [7] conjectured that for every plane matching on $4n$ vertices, there is a disjoint compatible plane matching. (*compatible geometric matchings conjecture*). They proved that their conjecture holds if the $2n$ segments in the matching admit a convex partition whose dual graph is the union of two edge-disjoint spanning trees, and the two endpoints of each segment corresponds to distinct spanning trees. Aichholzer *et al.* further conjectured for the $4n$ endpoints of $2n$ line segments in the plane, there is a permutation π such that $D(\pi)$ is the union of two edge-disjoint spanning trees (*two spanning trees conjecture*).

The conjecture would immediately imply that such a dual graph is 2-edge connected. Benbernou *et al.* [15] claimed that there is always a permutation π such that $D(\pi)$ is 2-edge connected—but there was a flaw in their argument [16].

In Section 5.4 we prove that such permutation π does **not** always exist, thus refute the *two spanning trees conjecture* of Aichholzer *et al.* [7].

5.2.2 Fault-Tolerant Wireless Networks

In today's information age, users of mobile phones, wireless data services, and GPS devices etc, have come to expect ubiquitous access to information—anywhere, anytime. The providers of these services try to meet the needs of their demanding customers or risk losing business. But it is difficult to provide uninterrupted service in the presence of obstacles such as tall buildings in a city (e.g. Manhattan) that may impede the signals from satellites or cell towers. Underground subways pose a similar problem. One possible solution is to install inexpensive radio devices on the boundaries of these obstacles, which can communicate with each other and provide connectivity to the users. We can model the obstacles as

convex polygons in the plane, and place radios on the boundaries of these obstacles. Our goal is to partition the service areas between radios such that (i) each radio serves clients in two designated convex areas, and (ii) between any two clients there are *at least two* disjoint sequences of radios such that consecutive radios are not occluded by obstacles. The purpose of the second condition is to provide fault-tolerance so that clients can still have access even when one of the radios responsible for the area fails. The communication network of these radio devices is exactly the dual graph defined above. If the radios are placed at the corners of all obstacles, then we can designate two convex regions to each radio with properties (i) and (ii). The result can also be applied to sensor networks in the sense of Tan *et al.* [70].

5.3 Results

- We show instances where **no** permutation π produces a STRAIGHT-FORWARD dual graph $D(\pi)$ that is 2-edge connected (Section 5.4). This is a counterexample to a conjecture by Aichholzer *et al.* [7].

- We show that for every finite set of disjoint convex polygons in the plane there is a convex partition (not necessarily STRAIGHT-FORWARD) and an assignment that produces a 2-edge connected dual graph (Section 5.5). The dual graph has the same number of nodes as in the case of STRAIGHT-FORWARD convex partition.

5.4 Counterexample for Two Spanning Trees Conjecture

Theorem 5.1 *For every $n \geq 15$, there are n disjoint line segments in the plane such that the dual graph $D(\pi)$ has a bridge (cut-edge—removing this edge disconnects the dual graph) for every permutation π.*

Proof. We show that for the 15 line segments in Fig. 5.2, every permutation π produces a STRAIGHT-FORWARD dual graph $D(\pi)$ with a bridge (cut-edge). Our construction consists of three rotationally symmetric copies of a configuration with 5 segments

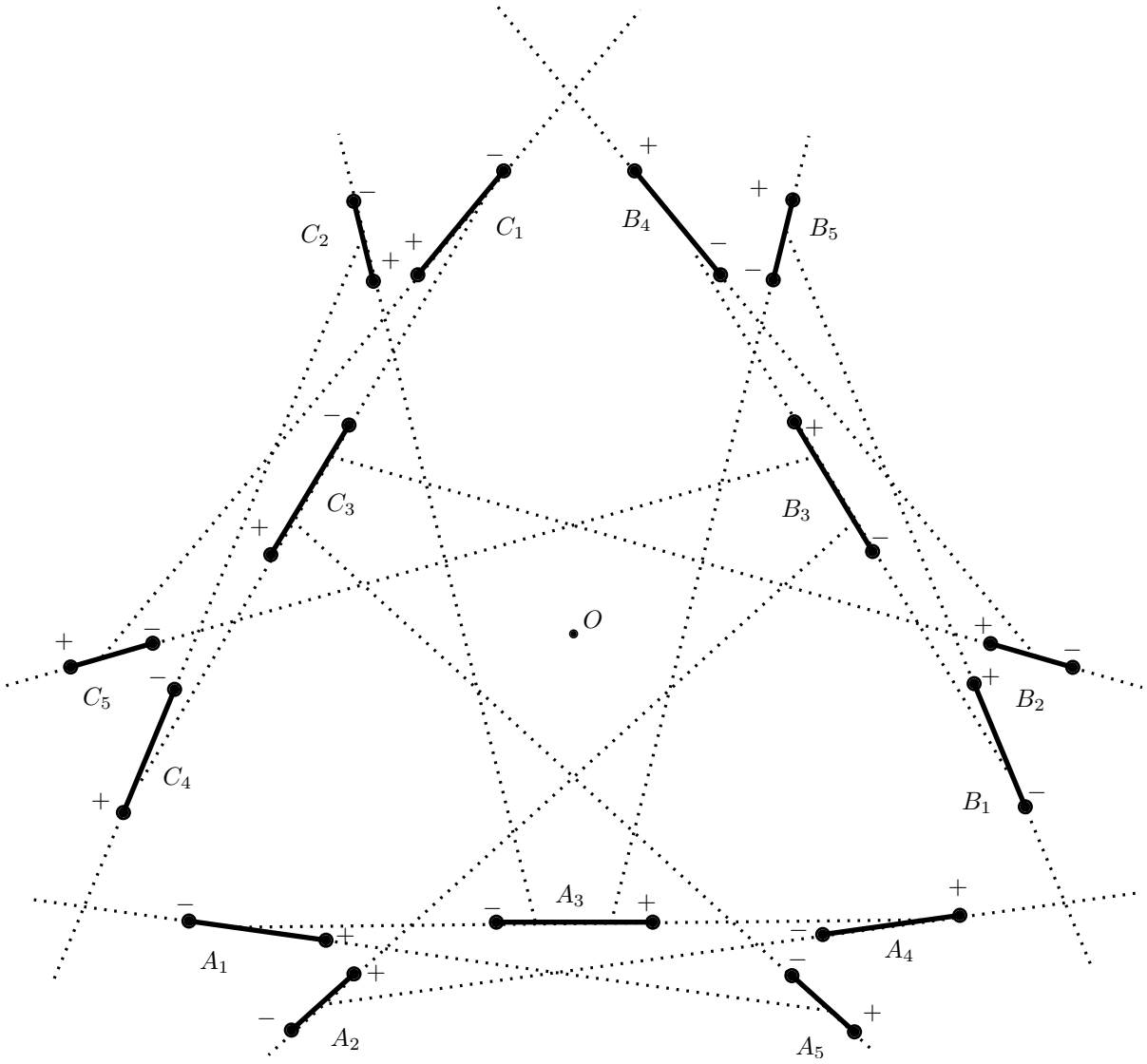
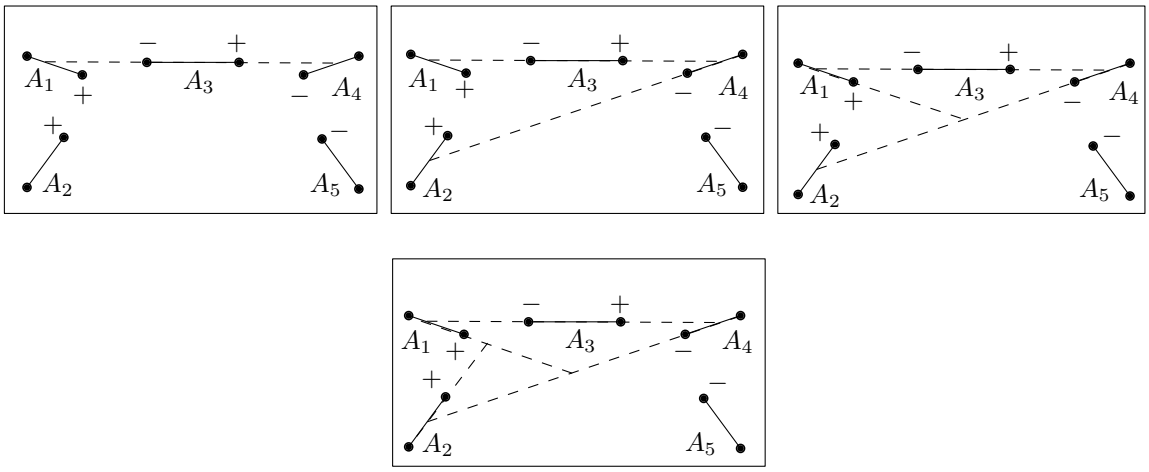


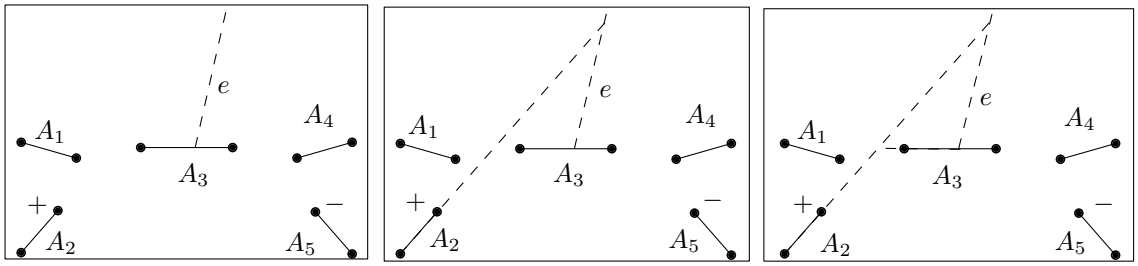
Figure 5.2: Counterexample with $n = 15$.

$\{A_1, A_2, \ldots, A_5\}$, which we call a *star structure*. We can generate larger constructions by adding segments whose supporting lines avoid the convex hull of this configuration.

In Fig. 5.2 the dotted lines represent the arrangement of all possible extensions of the given line segments. Denote the right endpoint of a segment by '$+$' and the left endpoint by '$-$' in the star structure A. The set of all possible permutations can be described in terms of only two cases by focusing on the star structure A. The structures B and C are copies of A rotated around the origin O.



Case 1: Extensions from A_3^- and A_3^+ hit the segments A_1 and A_4, respectively.



Case 2: The segment A_3 is hit by an extension e.

Figure 5.3: Permutations for the counterexample.

Case 1. The extensions from endpoints A_3^- and A_3^+ hit the segments A_1 and A_4, respectively; i.e. the extensions from endpoints A_2^+ and A_5^- terminate either at the extensions from endpoints A_3^- and A_3^+, respectively or earlier (Fig. 5.3). It can be easily verified that in this

case every permutation of the four endpoints $\{A_1^+, A_2^+, A_4^-, A_5^-\}$ produces a bridge in the dual graph. The same reasoning applies to the structures B and C because of symmetry.

Case 2. Therefore, to avoid a bridge in the dual graph, there must be at least one endpoint in each star structure whose extension goes beyond the structure. Since when two extensions meet in a STRAIGHT-FORWARD convex partition, one of the extensions must continue in a straight line, at least one of these three endpoints will have its extension hit a segment (A_3, B_3 or C_3) in a different structure. Assume w.l.o.g segment A_3 is hit by an extension e from either B_2^+ or C_5^-. Then an extension from either A_2^+ or A_5^- hits e, which together with A_3^- or A_3^+ creates a bridge in the dual graph. □

5.5 Constructing a Convex Partition

We showed in Section 5.4 that in some instances, no STRAIGHT-FORWARD dual graph is 2-edge connected. In this section we present an algorithm that produces a convex partition of the free space between disjoint convex polygonal obstacles, with a 2-edge connected dual graph. We will start from an arbitrary STRAIGHT-FORWARD convex partition, and apply a sequence of *local modifications*, if necessary, until the dual graph becomes 2-edge connected. Our local modifications will not change the number of cells. We define a class of convex partitions (DIRECTED-FOREST) that includes all STRAIGHT-FORWARD convex partitions and is closed under the local modifications we propose.

The basis for local modifications is a simple idea. In a STRAIGHT-FORWARD convex partition, extensions are created sequentially (each vertex emits a directed ray) and whenever two directed extensions meet at a *Steiner* vertex v (defined below), the earlier extension continues in its original direction, and the later one terminates. Here, however, we allow the two directed extensions to merge and continue as one edge in any direction that maintains the convexity of all the angles incident to v (Fig. 5.4(a)). Merged extensions provide considerable flexibility.

Definition 5.1 *For a given set S of disjoint obstacles, the class of* DIRECTED-FOREST *convex partitions is defined as follows: The free space $\mathbb{R}^2 \setminus (\bigcup S)$ is partitioned into convex cells by* directed edges *(including* directed rays *going to infinity). Each endpoint of a*

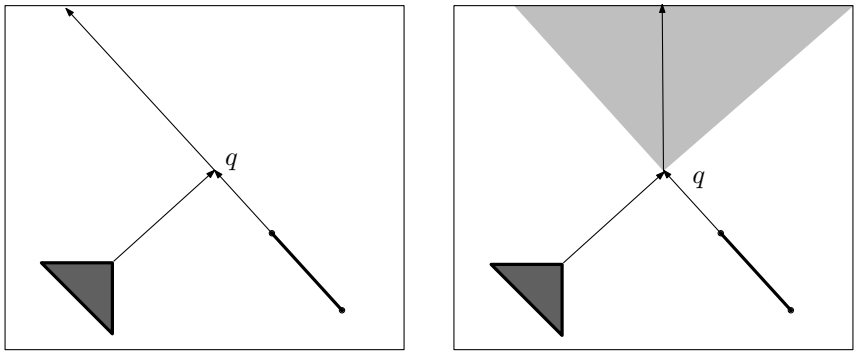


Figure 5.4: (a) Two incoming extensions meet at q. (b) The merged extension may continue in any direction within the opposing wedge.

directed edge is either a vertex of S or a Steiner vertex *(lying in the interior of the free space, or on the boundary of an obstacle). We require that*

- *every vertex in V (a vertex of an obstacle) emits exactly one outgoing edge;*
- *every Steiner point in the interior of the free space is incident to exactly one outgoing edge;*
- *no Steiner point on a convex obstacle is incident to any outgoing edge; and*
- *the directed edges do not form a cycle.*

It is easy to see that a STRAIGHT-FORWARD convex partition belongs to the class of DIRECTED-FOREST convex partitions.

Proposition 5.1 *There is an obstacle vertex on the boundary of every cell.*

Proof. Consider a directed edge on the boundary of a cell. Follow directed edges in reverse orientation along the boundary. Since directed edges cannot form a cycle, and the out-degree of every Steiner vertex is at most one, there must be at least one obstacle vertex on the boundary of the cell. □

In a DIRECTED-FOREST, we can also follow directed edges (in forward direction) from any vertex in V to an obstacle or to infinity, since the out-degree of each vertex is always

exactly one, unless the vertex lies on the boundary of an obstacle or at infinity. For connected components of extensions (directed edges), we use the concept of *extension trees* introduced by Bose *et al.* [19].

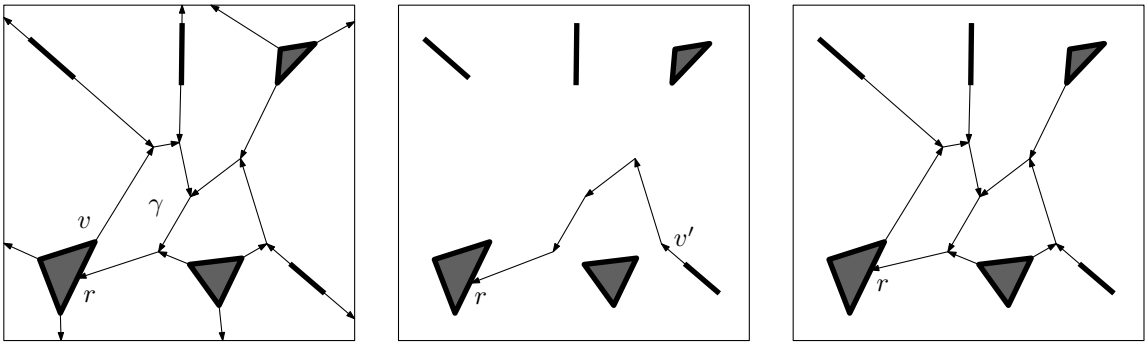


Figure 5.5: (a) A convex partition formed by directed line segments. The extended path γ originates at v and terminates at r, two points on the same obstacle. The edge at v is a bridge in the dual graph, and γ is called *forbidden*. (b) A single extended-path emitted by v'. (c) A single extension tree rooted at r.

Definition 5.2 *The* **extended-path** *of a vertex $v \in V$ is a directed path along directed edges starting from v and ending on an obstacle or at infinity. Its (relative) interior is disjoint from all obstacles.*

Definition 5.3 *An* **extension tree** *is the union of all extended-paths that end at the same point, which is called the* **root** *of the extension tree. The* **size** *of an extension tree is the number of extended-paths included in the tree.*

A vertex $v \in V$ may be incident to more than two cells. It is incident to $\ell + 2$ cells if it is incident to ℓ incoming edges. In our construction, we let σ assign a vertex v of an obstacle to the two cells adjacent to the unique outgoing edge incident to v. With this convention, a bridge in the dual graphs $D(C, \sigma)$ can be characterized by a forbidden pattern (see Fig. 5.5(a)).

Definition 5.4 *An extended-path starting at $v \in V$ is called* **forbidden** *if it ends at the obstacle incident to v. A* **forbidden** *extended-path, together with the boundary of the incident obstacle, forms a simple closed curve, which encloses a bounded region.*

Lemma 5.1 *A dual graph $D(C, \sigma)$ of a* DIRECTED-FOREST *convex partition is 2-edge connected iff no vertex $v \in V$ emits a forbidden extended-path.*

Proof. First we show that a forbidden extended-path implies a bridge in the dual graph. Let γ be a forbidden extended-path, starting from vertex v of an obstacle, and ending at point r on the boundary of the same obstacle (see Figures 5.5(a), 5.6, 5.7). Extended-path γ together with the obstacle boundary between v and r forms a simple closed curve and partitions the free space into two regions R_1 and R_2, each of which is the union of some convex cells. Let V_1 and V_2 be the set of nodes in the dual graph corresponding to the convex cells in these regions, respectively. Point v is the only obstacle vertex along γ. If an edge e of the dual graph connects some node in V_1 to a node in V_2, then e corresponds to a vertex of an obstacle whose unique outgoing edge is part of γ. But v is the only such vertex. This implies that there is a bridge in the dual graph, whose removal disconnects V_1 from V_2.

Next we show that a bridge in the dual graph implies a forbidden extended-path. Assume that V_1 and V_2 form a partition of V in $D(C, \sigma)$ such that V_1 and V_2 are connected by a bridge e. The two node sets correspond to two regions, R_1 and R_2, in the free space. Let β be boundary separating the two regions. We first show that one of these regions is bounded.

Suppose for contradiction that both regions R_1 and R_2 are unbounded. Note that β must contain at least two directed edges of the convex partition that go to infinity. Since every Steiner vertex in the interior of the free space has an outgoing edge, β must contain at least two extended-paths. Hence β contains at least two vertices of some obstacles, and the adjacent outgoing edges. Thus there are at least two edges in the dual graph between the node sets V_1 and V_2, therefore, e is not a bridge.

Now assume without loss of generality that the region R_1 is bounded, thus the separating boundary β is a closed curve. If we pick an arbitrary directed extension along β and follow β in reverse direction, then we arrive to a segment endpoint v. Assume that v corresponds to the bridge e. Then we arrive to the same segment endpoint v starting from any directed extension along β. This means that all directed edges along β are in the extended-path of v. Since β is a closed curve, the extended-path of v must end on the boundary of the obstacle incident to v, thus it is a forbidden extended-path. □

Corollary 5.1 *An extension tree with its root at infinity cannot contain a forbidden extended-path.*

5.5.1 Convex Partitioning Algorithm

We construct a convex partition as follows. We first create a STRAIGHT-FORWARD convex partition, which is in the class of DIRECTED-FOREST convex partitions. Let $\mathcal{T}$ denote the set of extension trees. Each extension tree may contain one or more forbidden extended-paths. If an extension tree $t \in \mathcal{T}$ contains a forbidden extended-path γ, then we continuously deform t with a sequence of local modifications until a vertex of an obstacle collides with the relative interior of t (subroutine FLEXTREE(t)). At that time, t splits into two extension trees t_1 and t_2 such that each of these two trees is strictly smaller in size than t. An extension tree of size one is a straight-line extension, and cannot contain a forbidden extended-path. Since the number of extended-paths is fixed (equal to the number of vertices in V), eventually no extension tree contains any forbidden extended-path, and we obtain a convex partition whose dual graph has no bridges by Lemma 5.1.

For a finite set S of disjoint convex polygonal obstacles in the plane, the main loop of our partition algorithm is CREATECONVEXPARTITION(S). The algorithm calls the subroutine FLEXTREE(t) for every extension tree that contains a forbidden extended-path, which in turn calls subroutine EXPAND(t, γ) for a forbidden extended path γ, as described in Section 5.5.2.

Algorithm 6 CREATECONVEXPARTITION(S)

Given: A set S of disjoint convex polygons having n vertices in total.
Create a STRAIGHT-FORWARD convex partition.
Let $\mathcal{T}$ be set of extension trees in the partition.
while there is an extension tree $t \in \mathcal{T}$ containing a forbidden extended-path **do**
  FLEXTREE(t)
end while

Algorithm 7 FLEXTREE(t)

Let γ be a forbidden extended-path contained in t.
while γ is still a forbidden extended-path **do**
  $(t, \gamma) =$ EXPAND(t, γ)
end while
Let $v' \in V$ be a vertex of an obstacle where the extended-path γ now terminates.
Split tree t into two extension trees t_1 and t_2. Subtree t_1 consists of the extended-paths that terminate at the original endpoint of γ. Subtree t_2 consists of the extended-paths that now terminate at v'.

5.5.2 Local Modifications: EXPAND(t, γ)

Consider a forbidden extended-path γ contained in an extension tree $t \in \mathcal{T}$. Path γ starts from a vertex $v \in V$, and ends at a root r lying on the boundary of the obstacle $s \in S$ incident to v. Path γ together with the portion of the boundary of s between v and r bounds a simple polygon P (does not contain s in its interior).

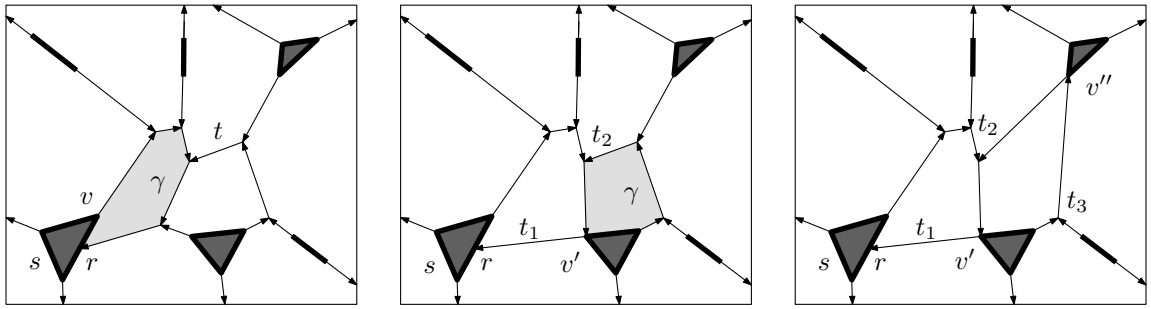


Figure 5.6: (a) An extension tree t with a forbidden extended-path. (b) After deforming and splitting t into two trees, t_2 contains a forbidden extended-path. (c) Deforming and splitting t_2 eliminates all forbidden extended-paths.

We continuously deform the boundary of P, together with extension tree t, until it collides with a new vertex $v' \in V$ that is not incident to s. Similar continuous motion arguments have been used for proving combinatorial properties in [9, 13, 57]. We deform P in a sequence of local modifications, or *step*s. Each step involves two adjacent edges of the polygon P. The *vertices of* P are v, r and the Steiner points where P has an interior angle different from $180°$. Steiner vertices where P has an interior angle of $180°$ are considered

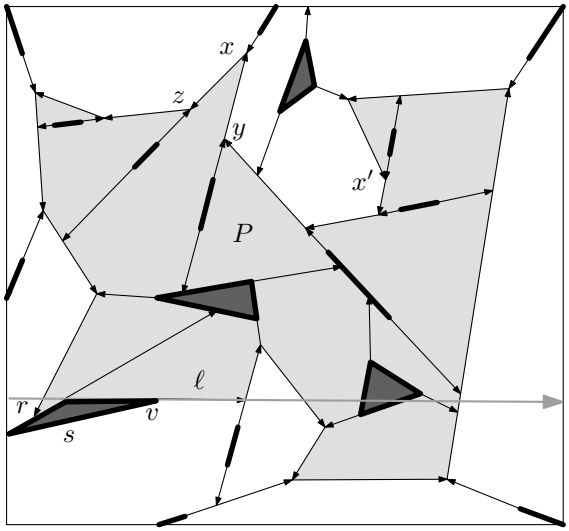


Figure 5.7: Polygon P corresponding to a forbidden extended-path $v, \ldots, r$; convex vertex x; inflexible edges xy and xz; reflex vertex x'.

interior points of edges of P. Each step of the deformation will (i) expand the interior of the polygon P, (ii) keep r a vertex of P, and (iii) maintain a valid DIRECTED-FOREST convex partition. The third condition implies, in particular, that every cell has to remain convex. Also, since the interior of P is expanding, some cells in the exterior of P (and adjacent to P) will shrink—we ensure that all cells adjacent to P have a nonempty interior.

Where to perform a local deformation step? The polygon P is modified either at a convex vertex x on the convex hull of P or at a reflex vertex x' (with special properties). These vertices x and x' are calculated at the start of each local deformation step.

Consider the edge of the obstacle s that is incident to the point v, and is part of the boundary of the polygon P. Let ℓ be the supporting line through this edge. The obstacle s lies completely in one of the closed halfplanes bounded by ℓ (since s is convex). Let x be a vertex of P furthest away from the supporting line ℓ in the other halfplane. Clearly, x is a convex vertex of P (interior angle less than $180°$), otherwise, it will not be the furthest. The goal is to expand the polygon P by modifying the edges xy and xz incident to x. Imagine grabbing the vertex x and pulling it away from the polygon P stretching the edges xy and xz. But this expansion can only occur if none the edges xy and xz are *inflexible*. An edge of P is *inflexible* if there is a convex cell in the interior of P that has an angle of $180°$ at

one of the two endpoints of the edge. Since x is a convex vertex, the edge xy or xz can be inflexible iff some convex cell has an angle of 180° at y or z, respectively (Fig. 5.7).

In the case when at least one of the edges incident to x is inflexible, local modification of P takes place at a reflex vertex x'. Assume w.l.o.g xy is inflexible. Then y must be a reflex vertex of P (every inflexible edge of P is incident to a reflex vertex). Starting with the reflex vertex y, move along the boundary of P in the direction away from x. Let x' be the first reflex vertex encountered such that one of the edges incident to x' is flexible. It is not difficult to verify that there is always one such vertex x' (Proposition 5.2).

Proposition 5.2 *If x is incident to an inflexible edge, then there is a reflex polygonal chain along P of length ≥ 1 that includes this inflexible edge and terminates at a reflex vertex x' of P that has exactly one flexible edge.* □

How to perform a local deformation step? Local deformation of P takes place either at a convex vertex x (Case 1 and 2), or at a reflex vertex x' (Case 3). Since the number of cells in the convex partition must remain the same, it is necessary to check for the collapse of a cell in the exterior of P (Case 4).

Case 1. Both edges xy and xz of P incident to x are flexible, and there is an edge wx in the opposing wedge of $\angle yxz$. Fig. 5.8(a). Then continuously move x along xw towards w while stretching the edges xy and xz.

Case 2. Both edges xy and xz of P incident to x are flexible, and there is no edge in the opposing wedge of $\angle yxz$. Fig. 5.8(b). Let ℓ_x be a line parallel to ℓ passing through x, and let w be a neighbor of x on the opposite side of ℓ_x. Assume that z and w are on the same side of the angle bisector of $\angle yxz$. Then split x into two vertices x_1 and x_2. Now x_1 remains fixed at x and x_2 moves continuously along xw towards w stretching the edge x_2z.

Case 3. At least one edge incident to x is inflexible; then there is a reflex vertex x' such that edge $x'z'$ is inflexible, and $x'y'$ is flexible. Fig. 5.8(c). Continuously move x' along $x'z'$ towards z' while stretching the edge $x'y'$.

Case 4. A further $\varepsilon > 0$ stretching of some edge ab to position ab', where vertex b continuously moves along segment bb', would collapse a cell in the exterior of P. Fig. 5.8(d). Then the triangle $\Delta abb'$ lies in the free space and ab' contains a side of an obstacle $s' \neq s$

(cf. Proposition 5.3 below). Let $v' \in ab'$ be the vertex of this obstacle that lies closer to a. Stretch edge ab of P into the path (a, v', b).

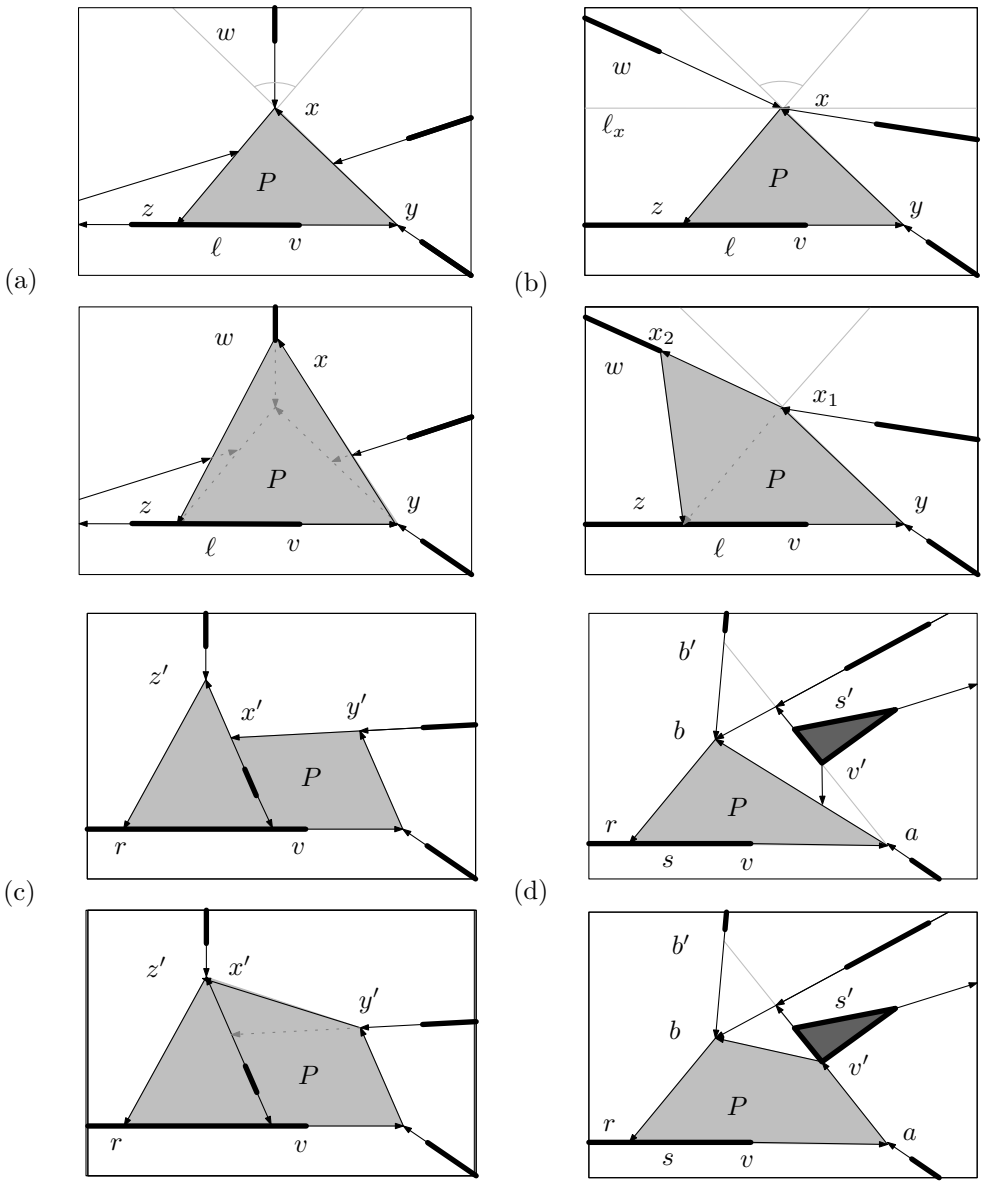


Figure 5.8: Three local operations: (a) Convex vertex x, incoming edge w in the wedge. (b) Convex vertex x, no incoming edge in the wedge. (c) Reflex vertex x. (d) The case of a collapsing cell.

When to stop a local deformation step? Continuously deform one or two edges of P, at either a convex vertex x or a reflex vertex x', until one of the following conditions occurs:

- an angle of a convex cell interior to P or an angle of P becomes $180°$;
- two vertices of the polygon P collide;
- one of the edges of P collides either in its interior or at its endpoint with a vertex v of an obstacle;
- a further $\varepsilon > 0$ deformation would collapse a cell in the exterior of P.

Since a local deformation step does not always terminate in a collision with an obstacle vertex v', the subroutine FLEXTREE(t) decides at the end of each step whether more local modifications are needed.

5.5.3 Correctness of the Algorithm

We prove that we can eliminate all forbidden extended-paths and obtain a DIRECTED-FOREST convex partition with a 2-edge connected dual graph. Let t be a extension tree, containing a forbidden extended-path γ starting from $v \in V$ and ending at root r. First we show that in EXPAND(t, γ), the four cases cover all possibilities.

Proposition 5.3 *If a further $\varepsilon > 0$ deformation of some edge ab to position ab', where b continuously moves along segment bb', would collapse a cell in the exterior of P, then the triangle $\Delta abb'$ lies in the free space and segment ab' contains a side of an obstacle $s' \neq s$.*

Proof. A continuous deformation of ab to ab', where b' moves along segment bb', sweeps triangle $\Delta abb'$. Hence the interior of this triangle cannot contain any obstacle. Assume that cell $c \in C$ would collapse if ab reaches position ab'. By Proposition 5.1, there is a vertex $v' \in V$ on the boundary of cell c, and so v' must lie on the segment ab'. Note that no extended-path can reach v' from the triangle $\Delta abb'$. Hence the only two edges along the boundary of c incident to v' are the extensions emitted by a side of the obstacle s' containing v'. It follows that segment ab' contains a side of obstacle s'.

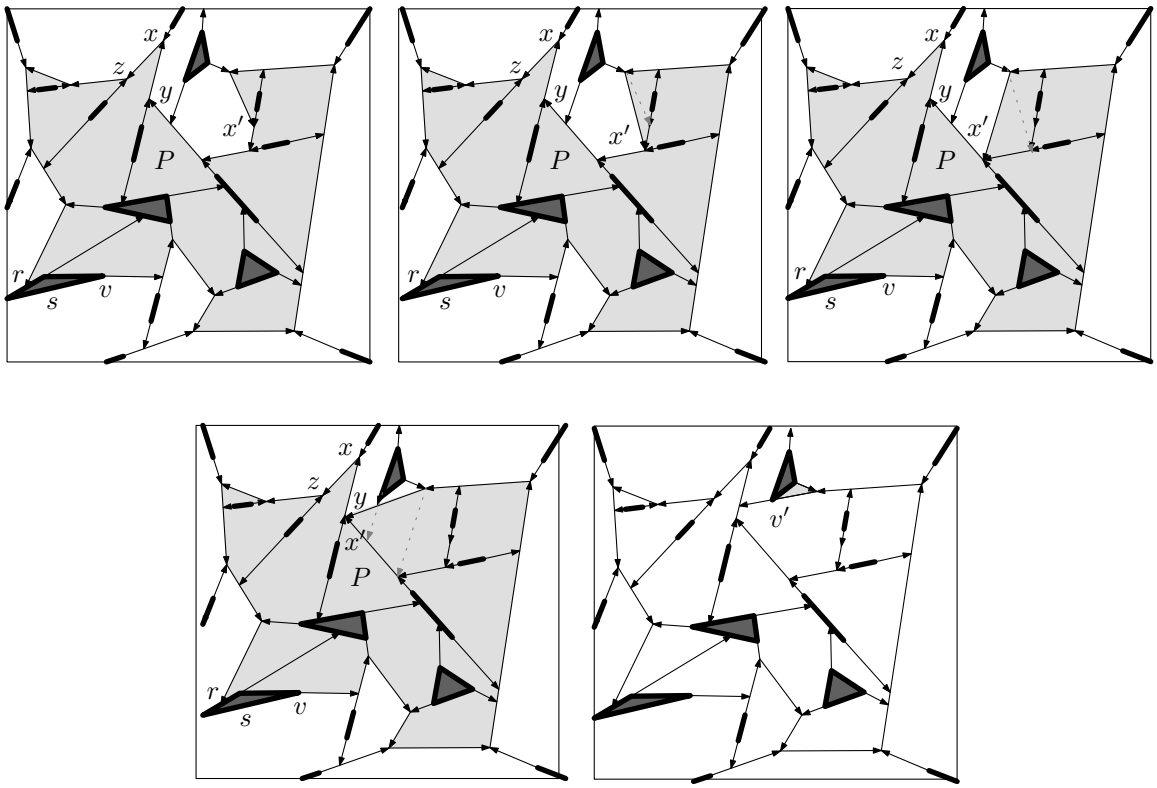


Figure 5.9: Polygon P is continuously deformed until an obstacle vertex appears on the boundary.

It remains to be shown that $s' \neq s$ (that is, v and v' are vertices of distinct obstacles). In Case 1–2, $b = x$ is the convex vertex of P that lies furthest from the supporting line ℓ, and b moves continuously away from ℓ. Therefore both b and b' are in the open halfplane bounded by ℓ, and so edge ab' cannot contain an edge of the obstacle s. In Case 3, $b = x'$ is a reflex vertex of P and it moves continuously along a reflex chain along the boundary of P between x' and x (c.f., Proposition 5.2). Since x is the furthest point from the supporting line ℓ of s, the reflex chain between x and x' is separated from s by a line. Segment ab' lies in the convex hull of the reflex chain, and so it cannot contain a side of s. □

Proposition 5.4 *Subroutine* EXPAND(t, γ) *(i) increases the interior of polygon P, (ii) keeps r as a vertex of P, and (iii) maintains a valid* DIRECTED-FOREST *convex partition. Furthermore,* EXPAND(t, γ) *modifies directed edges of the extension tree t only.* □

Lemma 5.2 *The subroutine* FLEXTREE(t) *modifies an extension tree* $t \in \mathcal{T}$*, with a forbidden extended-path* γ*, in a finite number of* EXPAND(t, γ) *steps until an obstacle vertex* $v' \in V$ *appears along* γ*.*

Proof. FLEXTREE(t) repeatedly calls EXPAND(t, γ) for a forbidden extended-path γ. We associate an integer $\mathrm{count}(t, \gamma)$ to t and γ and show that EXPAND(t, γ) either deforms t to collide with an obstacle $s \neq s'$ or $\mathrm{count}(t, \gamma)$ strictly decreases. This implies that FLEXTREE(t) terminates in a finite number of steps.

Let k denote the size of t (i.e., the number of extended-paths in t). Then t has at most $k - 1$ Steiner vertices in the free space, since each corresponds to the merging of two or more extended-paths. Let k_{ex} be the number of Steiner vertices of t in the exterior of P, let r_P be the number of vertices of P, let f_P be the number of flexible edges of P, and let m_P be the number of directed edges in t that are incident to vertex x of P from the exterior of P. Then let $\mathrm{count}(t, \gamma) = 2k \cdot k_{\text{ex}} + r_P + f_P + 2m_P$. Recall that a Steiner vertex where P has an internal angle of $180°$ is not a vertex of P. The vertices of P are v, r and Steiner vertices in the interior of the free space where P has a non-straight internal angle, hence $r_p, f_P, m_P < k$.

Consider a sequence of EXPAND(t, γ) steps where t does not collide with an obstacle. Since in Case 4, a vertex $v' \in V$ appears in the relative interior of t, we may assume that only Case 1–3 are applied. Case 1–3 expand the interior of polygon P, and the directed edges in the exterior of P are not deformed. Hence k_{ex} never increases, and it decreases if P expands and reaches a Steiner point in the exterior of P.

Now consider a sequence of EXPAND(t, γ) steps where k_{ex} remains fixed and Case 4 does not apply. Then m_P can only decrease in Case 1–3. Case 2 initially introduces a new edge of P (increasing r_P and f_P by one each) but it also decreases m_P by at least one. Case 1 and 3 never increase r_P or f_P. In Case 1–3, the deformation step terminates when an interior angle of a convex cell within P becomes $180°$ (and an edge becomes inflexible, decreasing f_P) or an interior angle of P becomes $180°$ (and P loses a vertex, decreasing r_P). In both events, $r_P + f_P$ decreases by at least one. Therefore, $\mathrm{count}(t, \gamma) = 2k \cdot k_{\text{ex}} + r_P + f_P + 2m_P$ strictly decreases in every step EXPAND(t, γ), until the relative interior of t collides with an obstacle. □

Theorem 5.2 *For every finite set of disjoint convex polygonal obstacles in the plane, there is a convex partition and an assignment σ such that the dual graph $D(C, \sigma)$ is 2-edge connected. For k convex polygonal obstacles with a total of n vertices, the convex partition consists of $n - k + 1$ convex cells.*

Proof. The convex partitioning algorithm first creates a STRAIGHT-FORWARD convex partition for the given set of disjoint polygonal obstacles. For k disjoint obstacles with a total of n vertices, it consists of $n-k+1$ convex cells. The extensions in the convex partition can be represented as a set of extension trees $\mathcal{T}$. We showed in Lemma 5.1 that there is a bridge in the dual graph iff some extension tree contains a forbidden extended-path. Subroutine FLEXTREE(t) splits every extension tree t containing a forbidden extended-path into two smaller trees. (The extended-paths in t are distributed between the two resultant trees.) An extension tree that consists of a single extended-path is a straight-line extension, and cannot be forbidden (a straight-line extension emitted from a vertex of an obstacle cannot hit the same obstacle, since each obstacle is convex.) Therefore, after at most $\frac{|V|}{2}$ calls to FLEXTREE(t), no extended-path is forbidden, and so the dual graph of the convex partition is 2-edge connected. □

Corollary 5.2 *For every finite set of disjoint line segments in the plane, there is a convex partition and an assignment σ such that the dual graph $D(C, \sigma)$ is 2-edge connected. For n disjoint line segments, the convex partition consists of $n + 1$ convex cells.*

5.6 Conclusion

We have shown that for every finite set of convex polygonal obstacles, there is a convex partition with a 2-edge connected dual graph. Such a dual graph can be used in the design of fault-tolerant wireless or sensor networks in the presence of polygonal obstacles. We have presented a polynomial time algorithm for constructing the dual graph. Our algorithm can easily be implemented in $O(n^3)$ time for obstacles with a total of n vertices, however we believe that the runtime can be substantially reduced by a more careful analysis and by using specialized data structures. For comparison, for any given permutation π the STRAIGHT-FORWARD convex partition C_π can be computed in $O(n \lg^2 n)$ time [50], and

if the permutation π is not specified, then a STRAIGHT-FORWARD convex partition can be computed in $O(n \lg n)$ time by a sweep-line algorithm (in a left-to-right sweep followed by a right-to-left sweep).

The question regarding the existence of convex partitions whose dual graph can be edge-partitioned into two disjoint spanning trees remains open. An affirmative answer to this question will settle the disjoint compatible matching conjecture by Aichholzer *et al.*. An interesting related problem is whether there always exists a convex partition that has a bi-connected (2-vertex connected) dual graph.

Tan *et al.* [70] computes STRAIGHT-FORWARD convex partition in a distributed manner, which makes the algorithm suitable for sensor networks. However, it remains to be seen whether the algorithm to produce convex partitions with 2-edge connected dual graphs presented in this paper could be modified to work in a distributed manner. A related question is how to support efficient insertion and deletion of convex polygonal obstacles. In other words: how *local* the local modifications really are?

Chapter 6

Conclusion

In thesis we bridge the gap between lower and upper bounds of simplex range emptiness queries by proving lower bounds in the partition graph model. The partition graph model covers data structures that recursively partition the space e.g. data structures based on the simplicial partition trees. Since geometric partitioning is a very important technique for building data structures, these lower bounds apply to a very broad class of data structures. The lower bounds are very severe e.g. a linear-storage ($O(n)$) data structure for simplex emptiness on a planar point set of 1 trillion points must spend 1 million units ($\Omega(\sqrt{n})$) of time answering an emptiness query. If there is an algorithm/application which generates a large number of such queries, $\Omega(\sqrt{n})$ query time will result in a severe bottleneck. To make things worse, simplex emptiness reduces to a host of geometric problems, thus implying lower bounds for these problems as well.

However, we can turn this argument on his head: if there is an algorithm generating large number of queries, may be there is an opportunity to exploit the structure in the query sequence. Thus instead of designing general data structures that do not rely on the peculiarities of a particular algorithm and are useful in a variety of applications, we need to construct data structures specific to the particular algorithm/application. In this thesis we improved the performance of Paterson and Yao's classical randomized auto-partition algorithm by developing a new data structure for ray shooting-and-insertion in the free space between disjoint polygonal obstacles. The idea of creating specialized data structures is not new; one well-known example is that of using Fibonacci heaps for Prim's minimum

spanning tree algorithm. However, the idea needs to be pursued more often for geometric algorithms especially when the algorithmic performance is critical such as in computer graphics, geographic information system, and computer-aided design and engineering. We hope this thesis succeeds in drawing the attention of computational geometry community towards the need for creating application-specific data structures.

Bibliography

[1] AGARWAL, P., AND ERICKSON, J. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, vol. 223 of *Contemp. Math.* AMS, 1999, pp. 1–56.

[2] AGARWAL, P. K. Range searching. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, 1997, ch. 31, pp. 575–598.

[3] AGARWAL, P. K. Range searching. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds., 2 ed. CRC Press LLC, Boca Raton, FL, 2004, ch. 36, pp. 809–838.

[4] AGARWAL, P. K., BASCH, J., GUIBAS, L. J., HERSHBERGER, J. E., AND ZHANG, L. Deformable free space tilings for kinetic collision detection. *Int. J. Robotics Research 21*, 3 (Mar 2002), 179–197.

[5] AGARWAL, P. K., AND SHARIR, M. Applications of a new space-partitioning technique. *Discrete Comput. Geom. 9* (1993), 11–38.

[6] AGARWAL, P. K., AND VAN KREVELD, M. Implicit point location in arrangements of line segments, with an application to motion planning. *Int. J. Comput. Geom. Appl. 4*, 4 (1994), 369–383.

[7] AICHHOLZER, O., BEREG, S., DUMITRESCU, A., GARCÍA, A., HUEMER, C., HURTADO, F., KANO, M., MÁRQUEZ, A., RAPPAPORT, D., SMORODINSKY, S., SOUVAINE, D., URRUTIA, J., AND WOOD, D. R. Compatible geometric matchings. *Comput. Geom. Theory Appl. 42*, 6-7 (2009), 617–626.

[8] AL-JUBEH, M., HOFFMANN, M., ISHAQUE, M., SOUVAINE, D. L., AND TÓTH, C. D. Convex partitions with 2-edge connected dual graphs. In *COCOON* (2009), pp. 192–204.

[9] ANDRZEJAK, A., ARONOV, B., HAR-PELED, S., SEIDEL, R., AND WELZL, E. Results on k-sets and j-facets via continuous motion. In *Proc. 14th Symp. Comput. Geom.* (New York, NY, 1998), ACM, pp. 192–199.

[10] ARONOV, B., BOSE, P., DEMAINE, E. D., GUDMUNDSSON, J., IACONO, J., LANGERMAN, S., AND SMID, M. H. M. Data structures for halfplane proximity queries and incremental Voronoi diagrams. In *Proc. 7th Latin Amer. Sympos. Theor. Inf.* (2006), vol. 3887 of *LNCS*, Springer, pp. 80–92.

[11] ARYA, S., MALAMATOS, T., AND MOUNT, D. M. On the importance of idempotence. In *Proc. Sympos. Theory of Computing* (2006), ACM Press, pp. 564–573.

[12] ARYA, S., MOUNT, D. M., AND XIA, J. Tight lower bounds for halfspace range searching. In *Symposium on Computational Geometry* (2010), pp. 29–37.

[13] BANCHOFF, T. F. Global geometry of polygons I: The theorem of Fabricius-Bjerre. *Proc. AMS 45* (1974), 237–241.

[14] BASCH, J., ERICKSON, J., GUIBAS, L. J., HERSHBERGER, J., AND ZHANG, L. Kinetic collision detection between two simple polygons. *Comput. Geom. 27*, 3 (2004), 211–235.

[15] BENBERNOU, N., DEMAINE, E. D., DEMAINE, M. L., HOFFMANN, M., ISHAQUE, M., SOUVAINE, D. L., AND TÓTH, C. D. Disjoint segments have convex partitions with 2-edge connected dual graphs. In *Proc. CCCG* (2007), pp. 13–16.

[16] BENBERNOU, N., DEMAINE, E. D., DEMAINE, M. L., HOFFMANN, M., ISHAQUE, M., SOUVAINE, D. L., AND TÓTH, C. D. Erratum for "disjoint segments have convex partitions with 2-edge connected dual graphs". In *Proc. CCCG* (2008), p. 223.

[17] BENBERNOU, N., ISHAQUE, M., AND SOUVAINE, D. L. Data structures for restricted triangular range searching. In *CCCG* (2008).

[18] BERG, M. D., CHEONG, O., KREVELD, M. V., AND OVERMARS, M. *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.

[19] BOSE, P., HOULE, M. E., AND TOUSSAINT, G. T. Every set of disjoint line segments admits a binary tree. *Discrete Comput. Geom. 26*, 3 (2001), 387–410.

[20] BRASS, P., AND KNAUER, C. On counting point-hyperplane incidences. *Comput. Geom. theory Appl. 25*, 1-2 (2003), 13–20.

[21] BRÖNNIMANN, H., CHAZELLE, B., AND PACH, J. How hard is half-space range searching? *Discrete Comput. Geom. 10* (1993), 143–155.

[22] CARLSSON, J. G., ARMBRUSTER, B., AND YE, Y. Finding equitable convex partitions of points in a polygon efficiently. *ACM Transactions on Algorithms* (2010). to appear.

[23] CHAN, T. M. Optimal partition trees. In *Symposium on Computational Geometry* (2010), pp. 1–10.

[24] CHAZELLE, B. On the convex layers of a planar set. *IEEE Trans. Inform. Theory IT-31*, 4 (July 1985), 509–517.

[25] CHAZELLE, B. An algorithm for segment-dragging and its implementation. *Algorithmica 3* (1988), 205–221.

[26] CHAZELLE, B. Functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput. 17*, 3 (1988), 427–462.

[27] CHAZELLE, B. Lower bounds on the complexity of polytope range searching. *JAMS 2* (1989), 637–666.

[28] CHAZELLE, B. Lower bounds for off-line range searching. *Discrete & Computational Geometry 17*, 1 (1997), 53–65.

[29] CHAZELLE, B., AND DOBKIN, D. P. *Optimal Convex Decompositions*, vol. 2. 1985, pp. 63–133.

[30] CHAZELLE, B., EDELSBRUNNER, H., GRIGNI, M., GUIBAS, L. J., HERSHBERGER, J., SHARIR, M., AND SNOEYINK, J. Ray shooting in polygons using geodesic triangulations. *Algorithmica 12* (1994), 54–68.

[31] CHAZELLE, B., AND GUIBAS, L. J. Fractional cascading: I. a data structuring technique. *Algorithmica 1*, 2 (1986), 133–162.

[32] CHAZELLE, B., GUIBAS, L. J., AND LEE, D. T. The power of geometric duality. *BIT 25*, 1 (1985), 76–90.

[33] CHAZELLE, B., AND LIU, D. Lower bounds for intersection searching and fractional cascading in higher dimension. *J. Comput. Syst. Sci. 68*, 2 (2004), 269–284.

[34] CHAZELLE, B., AND ROSENBERG, B. Simplex range reporting on a pointer machine. *Comput. Geom. Theory Appl. 5*, 5 (1996), 237–247.

[35] CHAZELLE, B., SHARIR, M., AND WELZL, E. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica 8* (1992), 407–429.

[36] DAESCU, O., MI, N., SHIN, C.-S., AND WOLFF, A. Farthest-point queries with geometric and combinatorial constraints. *Comput. Geom. Theory Appl. 33*, 3 (2006), 174–185.

[37] DAESCU, O., MI, N., SHIN, C.-S., AND WOLFF, A. Farthest-point queries with geometric and combinatorial constraints. *Comput. Geom. Theory Appl. 33*, 3 (2006), 174–185.

[38] EDELSBRUNNER, H. *Algorithms in combinatorial geometry*. Springer, New York, 1987.

[39] ERICKSON, J. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom. 16*, 4 (1996), 389–418.

[40] ERICKSON, J. Space-time tradeoffs for emptiness queries. *SIAM J. Comput. 29*, 6 (2000), 1968–1996.

[41] FREDMAN, M. L. A lower bound on the complexity of orthogonal range queries. *J. ACM 28*, 4 (1981), 696–705.

[42] GANGULI, A., CORTES, J., AND BULLO, F. Multirobot rendezvous with visibility sensors in nonconvex environments. *IEEE Transactions on Robotics* (Aug. 2007). (Submitted Nov 2006) Conditionally Accepted.

[43] GHOSH, S. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, NY, USA, 2007.

[44] GOODRICH, M. T., AND TAMASSIA, R. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *J. Algorithms 23* (1997), 51–73.

[45] GOSWAMI, P. P., DAS, S., AND NANDY, S. C. Simplex range searching and k nearest neighbors of a line segment in 2d. In *SWAT* (London, UK, 2002), Springer-Verlag, pp. 69–79.

[46] HAUSSLER, D., AND WELZL, E. Epsilon-nets and simplex range queries. *Discrete Comput. Geom. 2* (1987), 127–151.

[47] HERSHBERGER, J., AND SURI, S. Applications of a semi-dynamic convex hull algorithm. *BIT 32* (1992), 249–267.

[48] HERSHBERGER, J., AND SURI, S. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms 18*, 3 (1995), 403–431.

[49] HOFFMANN, M., AND TÓTH, C. D. Segment endpoint visibility graphs are hamiltonian. *Comput. Geom. 26*, 1 (2003), 47–68.

[50] ISHAQUE, M., SPECKMANN, B., AND TÓTH, C. D. Shooting permanent rays among disjoint polygons in the plane. In *SCG '09: Proc. 25th Symp. Comput. Geom.* (New York, NY, USA, 2009), ACM, pp. 51–60.

[51] ISHAQUE, M., AND TÓTH, C. D. Relative convex hulls in semi-dynamic subdivisions. In *Proc. 16th European Sympos. on Algorithms* (2008), vol. 5193 of *LNCS*, Springer, pp. 780–792.

[52] KANEKO, A., AND KANO, M. Perfect partitions of convex sets in the plane. *Discrete & Computational Geometry 28*, 2 (2002), 211–222.

[53] KEIL, M. Polygon decomposition. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000, pp. 491–518.

[54] KEIL, M., AND SNOEYINK, J. On the time bound for convex decomposition of simple polygons. *Internat. J. Comput. Geom. Appl. 12* (2002), 181–192.

[55] KIRKPATRICK, D. G., AND SPECKMANN, B. Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. In *Symposium on Computational Geometry* (2002), pp. 179–188.

[56] KNUTH, D. E. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

[57] KRUMME, D. W., RAFALIN, E., SOUVAINE, D. L., AND TÓTH, C. D. Tight bounds for connecting sites across barriers. *Discrete and Computational Geometry 40*, 3 (2008), 377–394.

[58] LIEN, J.-M., AND AMATO, N. M. Approximate convex decomposition of polygons. *Comput. Geom. 35*, 1-2 (2006), 100–123.

[59] LINGAS, A. The power of non-rectilinear holes. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming* (London, UK, 1982), Springer-Verlag, pp. 369–383.

[60] MATOUŠEK, J. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom. 10* (1993), 157–182.

[61] MATOUŠEK, J. Geometric range searching. *ACM Comput. Surv. 26*, 4 (1994), 421–461.

[62] MITCHELL, J. S. B. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[63] OVERMARS, M. H., AND VAN LEEUWEN, J. Maintenance of configurations in the plane. *J. Comput. Syst. Sci. 23* (1981), 166–204.

[64] PATERSON, M. S., AND YAO, F. F. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom. 5* (1990), 485–503.

[65] PĂTRAŞCU, M. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computing (STOC)* (2007), pp. 40–46.

[66] PELLEGRINI, M. *Ray shooting and lines in space*, 2 ed. CRC Press LLC, Boca Raton, FL, 2004, ch. 37, pp. 239–256.

[67] SKLANSKY, J., CHAZIN, R. L., AND HANSEN, B. J. Minimum perimeter polygons of digitized silhouettes. *IEEE Trans. Comput. C-21* (1972), 260–268.

[68] SPECKMANN, B. *Kinetic Data Structures for Collision Detection*. PhD thesis, University of British Columbia, Vancouver, 2001.

[69] SZEMERDI, E., AND TROTTER, W. T. Extremal problems in discrete geometry. *Combinatorica 3* (1983), 381–392.

[70] TAN, G., BERTIER, M., AND KERMARREC, A.-M. Convex partition of sensor networks and its use in virtual coordinate geographic routing. In *INFOCOM* (2009), pp. 1746–1754.

[71] TARJAN, R. E. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences 18*, 2 (1979), 110 – 127.

[72] TARJAN, R. E. *Data structures and network algorithms*. SIAM, Philadelphia, 1983.

[73] TÓTH, C. D. Binary space partition for line segments with a limited number of directions. *SIAM J. Comput. 32*, 2 (2003), 307–325.

[74] TÓTH, C. D. A note on binary plane partitions. *Discrete Comput. Geom. 30*, 1 (2003), 3–16.

[75] TÓTH, C. D. Binary plane partitions for disjoint line segments. In *Proc. 25th Sympos. on Comput. Geom.* (Aaarhus, 2009), ACM Press, pp. 71–79.

[76] TOUSSAINT, G. T. Shortest path solves translation separability of polygons. Report SOCS-85.27, School Comput. Sci., McGill Univ., Montreal, PQ, 1985.

[77] TOUSSAINT, G. T. An optimal algorithm for computing the relative convex hull of a set of points in a polygon. In *Signal Processing III: Theories and Applications*. Proc. 3rd European Signal Process. Conf., 1986, pp. 853–856.

[78] WILLARD, D. E. Multidimensional search trees that provide new types of memory reductions. *J. ACM 34*, 4 (1987), 846–858.

[79] YAO, A. C.-C. On the complexity of maintaining partial sums. *SIAM J. Comput. 14*, 2 (1985), 277–288.